

**Stavros Konstantinidis  
Giovanni Pighizzini (Eds.)**

**LNCS 10952**

# **Descriptive Complexity of Formal Systems**

**20th IFIP WG 1.02 International Conference, DCFS 2018  
Halifax, NS, Canada, July 25–27, 2018  
Proceedings**



**ifip**



**Springer**

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, Lancaster, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Zurich, Switzerland*

John C. Mitchell

*Stanford University, Stanford, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

C. Pandu Rangan

*Indian Institute of Technology Madras, Chennai, India*

Bernhard Steffen

*TU Dortmund University, Dortmund, Germany*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbrücken, Germany*

More information about this series at <http://www.springer.com/series/7407>

Stavros Konstantinidis · Giovanni Pighizzini (Eds.)

# Descriptive Complexity of Formal Systems

20th IFIP WG 1.02 International Conference, DCFS 2018  
Halifax, NS, Canada, July 25–27, 2018  
Proceedings

*Editors*

Stavros Konstantinidis  
Department of Mathematics  
and Computing Science  
Saint Mary's University  
Halifax, NS  
Canada

Giovanni Pighizzini  
Dipartimento di Informatica e Comunicazi  
Universita degli Studi di Milano  
Milan  
Italy

ISSN 0302-9743                      ISSN 1611-3349 (electronic)  
Lecture Notes in Computer Science  
ISBN 978-3-319-94630-6              ISBN 978-3-319-94631-3 (eBook)  
<https://doi.org/10.1007/978-3-319-94631-3>

Library of Congress Control Number: 2018947360

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

© IFIP International Federation for Information Processing 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG  
part of Springer Nature  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

This volume contains the papers presented at the 20th International Conference on Descriptive Complexity of Formal Systems (DCFS 2018), which was held at Saint Mary's University in Halifax, Nova Scotia, Canada, during July 25–27, 2018. It was jointly organized by the Working Group 1.02 on Descriptive Complexity of the International Federation for Information Processing (IFIP) and by the Department of Mathematics and Computing Science at Saint Mary's University.

The DCFS conference series is an international venue for the dissemination of new results related to all aspects of descriptive complexity including, but not limited to:

- Automata, grammars, languages, and other formal systems; various modes of operations and complexity measures
- Succinctness of description of objects, state-explosion-like phenomena
- Circuit complexity of Boolean functions and related measures
- Size complexity of formal systems
- Structural complexity of formal systems
- Trade-offs between computational models and mode of operation
- Applications of formal systems – for instance in software and hardware testing, in dialogue systems, in systems modelling or in modelling natural languages – and their complexity constraints
- Co-operating formal systems
- Size or structural complexity of formal systems for modelling natural languages
- Complexity aspects related to the combinatorics of words
- Descriptive complexity in resource-bounded or structure-bounded environments
- Structural complexity as related to descriptive complexity
- Frontiers between decidability and undecidability
- Universality and reversibility
- Nature-motivated (bio-inspired) architectures and unconventional models of computing
- Blum static (Kolmogorov/Chaitin) complexity, algorithmic information

DCFS became an IFIP working conference in 2016, continuing the former Workshop on Descriptive Complexity of Formal Systems, which was a merger in 2002 of two other workshops: FDSR (Formal Descriptions and Software Reliability) and DCAGRS (Descriptive Complexity of Automata, Grammars and Related Structures). DCAGRS was previously held in Magdeburg (1999), London (2000), and Vienna (2001). FDSR was previously held in Paderborn (1998), Boca Raton (1999), and San Jose (2000). Since 2002, DCFS has been successively held in London, Ontario, Canada (2002), Budapest, Hungary (2003), London, Ontario, Canada (2004), Como, Italy (2005), Las Cruces, New Mexico, USA (2006), Nový Smokovec, High Tatras, Slovakia (2007), Charlottetown, Prince Edward Island, Canada (2008), Magdeburg, Germany (2009), Saskatoon, Canada (2010), Giessen, Germany (2011), Braga,

Portugal (2012), London, Ontario, Canada (2013), Turku, Finland (2014), Waterloo, Ontario, Canada (2015), Bucharest, Romania (2016), and Milan, Italy (2017).

The submission and refereeing process was supported by the EasyChair conference system. In total, 24 papers were submitted by authors in 14 different countries. Each paper was reviewed by at least three Program Committee members. The Program Committee selected 19 papers for presentation at the conference and publication in this volume. There were three invited talks by:

- Jacques Sakarovitch, CNRS/Paris Diderot University and Telecom ParisTech, France
- Peter Selinger, Dalhousie University, Halifax, Canada
- Ludwig Staiger, Martin-Luther-Universität, Halle, Germany

We are very thankful to all invited speakers, contributing authors, Program Committee members, and external referees for their valuable contributions towards the realization of DCFS 2018.

We also thank Alfred Hofmann and Anna Kramer of Springer for their guidance during the process of publishing this volume.

Finally, we are indebted to the Organizing Committee members Ms. Rose Daurie, Dr. Paul Muir, Dr. Juraj Šebej, and the graduate students of the Department of Mathematics and Computing Science at Saint Mary's University, Halifax.

We look forward to the next DCFS in Košice, Slovakia.

July 2018

Stavros Konstantinidis  
Giovanni Pighizzini

# Organization

## Steering Committee

Cezar Câmpeanu	University of Prince Edward Island, Charlottetown, Canada
Erzsébet Csuhaj-Varjú	Eötvös Loránd University, Budapest, Hungary
Jürgen Dassow	Otto von Guericke University, Magdeburg, Germany
Helmut Jürgensen	Western University, London, Canada
Martin Kutrib (Chair)	Justus Liebig University, Giessen, Germany
Giovanni Pighizzini	University of Milan, Italy
Rogério Reis	University of Porto, Portugal

## Program Committee

Suna Bensch	Umeå University, Sweden
Francine Blanchet-Sadri	The University of North Carolina at Chapel Hill, USA
Cezar Câmpeanu	University of Prince Edward Island, Canada
Erzsébet Csuhaj-Varjú	Eötvös Loránd University, Budapest, Hungary
Michael J. Dinneen	The University of Auckland, New Zealand
Henning Fernau	University of Trier, Germany
Viliam Geffert	P.J. Šafárik University, Košice, Slovakia
Dora Giammarresi	University of Rome Tor Vergata, Italy
Yo-Sub Han	Yonsei University, South Korea
Szabolcs Iván	University of Szeged, Hungary
Galina Jirásková	Slovak Academy of Sciences, Slovakia
Christos Kapoutsis	Carnegie Mellon University, Qatar
Stavros Konstantinidis	Saint Mary's University, Halifax, Canada
Martin Kutrib	Justus Liebig University, Giessen, Germany
Sylvain Lombardy	LaBRI - CNRS - Institut Polytechnique de Bordeaux, France
Kalpna Mahalingam	Indian Institute of Technology, India
Andreas Malcher	Justus Liebig University, Giessen, Germany
Andreas Maletti	Universität Leipzig, Germany
Ian McQuillan	University of Saskatchewan, Canada
Victor Mitrană	Universidad Politécnică de Madrid, Spain
Nelma Moreira	University of Porto, Portugal
Jean Néraud	University of Rouen, France
Alexander Okhotin	St. Petersburg State University, Russia
Giovanni Pighizzini	University of Milan, Italy
George Rahonis	Aristotle University of Thessaloniki, Greece
Narad Rampersad	University of Winnipeg, Canada
Kai Salomaa	Queen's University, Kingston, Canada



Shinnosuke Seki	University of Electro-Communications, Tokyo, Japan
Petr Sosík	Silesian University, Czech Republic
Lynette Van Zijl	Stellenbosch University, South Africa
Mikhail Volkov	Ural State University, Russia
Abuzer Yakaryilmaz	University of Latvia, Riga, Latvia

## **Additional Reviewers**

Marcella Anselmo	Sang-Ki Ko
Mikhail Barash	Pavlos Marantidis
Simon Beier	Benedek Nagy
Michael Cadilhac	Paulina Paraponiari
Stefan Dück	Erik Paul
Kitti Gelle	Alexander Szabari
Gustav Grabolle	Matthias Wendlandt
Jozef Jirásek	Mansur Ziatdinov
Chris Keeler	

## **Sponsors**

AARMS (Atlantic Association for Research in the Mathematical Sciences)  
Academic Vice President, Saint Mary's University  
Dean of Science, Saint Mary's University  
Director of Computing and Data Analytics, Saint Mary's University  
Destination Halifax

# The Complexity of Carry Propagation for Successor Functions (Extended Abstract)

Valérie Berthé<sup>1</sup>, Christiane Frougny<sup>1</sup>, Michel Rigo<sup>2</sup>,  
and Jacques Sakarovitch<sup>3</sup>

<sup>1</sup> IRIF - UMR 8243 - CNRS/Paris Diderot University, France

<sup>2</sup> Université de Liège, Institut de Mathématiques, France

<sup>3</sup> IRIF - UMR 8243 - CNRS/Paris Diderot University  
and Telecom ParisTech, France

*Carry propagation* is the nightmare of school pupils and the headache of computer engineers: not only can the addition of two digits give rise to a carry, but this carry itself, when added to the next digits to the left<sup>1</sup> may give rise to another carry, and so on, and so forth, and this may happen for an arbitrarily long time. Since the beginnings of computer science, the evaluation of the carry propagation length has been the subject of many works and it is known that the average carry propagation length (or complexity) for addition of two uniformly distributed  $n$ -digits binary numbers is  $\log_2(n) + O(1)$  (see [5, 7, 10]).

We consider here the problem of carry propagation from a more theoretical perspective and in an apparently elementary case. We investigate the amortized carry propagation of the *successor function* in various numeration systems. The central case of integer base numeration system allows us to describe quickly what we mean. Let us take an integer  $p$  greater than 1 as a base. In the representations of the succession of the integers — which is exactly what the successor function does — the least digit changes at every step, the penultimate digit changes every  $p$  steps, the ante-penultimate digit changes every  $p^2$  steps, and so on and so forth ... As a result, the average carry propagation of the successor function, computed over the first  $N$  integers, should tend to the quantity

$$1 + \frac{1}{p} + \frac{1}{p^2} + \frac{1}{p^3} + \cdots = \frac{p}{p-1},$$

when  $N$  tends to infinity. It can be shown that it is indeed the case. Motivated by various works on non-standard numeration systems, we investigate the questions of evaluating and computing the amortized carry propagation in those systems. We thus consider several such numeration systems different from the classical integer base numeration systems: the greedy numeration systems and the beta-numeration systems, see [6], which are a particular case of the former, the rational base numeration systems [1] which are not greedy numeration systems, and the abstract numeration systems [8] which are a generalization of the classical *positional numeration systems*.

---

<sup>1</sup> We write numbers under MSDF (Most Significant Digit First) convention.

The approach of *abstract numeration systems* of [8], namely the study of a numeration system via the properties of the set of expansions of the natural integers is well-fit to this problem. Such systems consist of a totally ordered alphabet  $A$  of the non-negative integers  $\mathbb{N}$  and a language  $L$  of  $A^*$ , ordered by the *radix order* deduced from the ordering on  $A$ . The representation of an integer  $n$  is then the  $(n + 1)$ -th word of  $L$  in the radix order. This definition is consistent with every classical standard and non-standard numeration system.

Given a system defined by a language  $L$  ordered by radix order, we denote by  $\text{cp}_L(i)$  the carry propagation in the computation from the representation of  $i$  in  $L$  to that of  $i + 1$ . The (*amortized*) *carry propagation* of  $L$ , which we denote by  $\text{CP}_L$ , is the limit, if it exists, of the mean of the carry propagation at the first  $N$  words of  $L$ :

$$\text{CP}_L = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=0}^{N-1} \text{cp}_L(i). \quad (1)$$

A further hypothesis is to consider *prefix-closed* and *right-extensible* languages, called ‘PCE’ languages in the sequel: every left-factor of a word of  $L$  is a word of  $L$  and every word of  $L$  is a left-factor of a longer word of  $L$ . Hence,  $L$  is the branch language of an infinite labelled tree  $\mathcal{T}_L$  and, once again, every classical standard and non-standard numeration system meets that hypothesis.

We first prove two easy properties of the carry propagation of PCE languages. First,  $\text{CP}_L$  does not depend upon the labelling of  $\mathcal{T}_L$ , but only on its ‘shape’ which is completely defined by the infinite sequence of the degrees of the nodes visited in a breadth-first traversal  $\mathcal{T}_L$ , and which is called *signature* of  $\mathcal{T}_L$  (or of  $L$ ) in [9]. For instance, the signature of the representation language in base  $p$  is the constant sequence  $p^\omega$ . Second, we call *local growth rate* of a language  $L$ , and we denote by  $\gamma_L$ , the limit, if it exists, of the ratio  $\mathbf{u}_L(\ell + 1)/\mathbf{u}_L(\ell)$ , where  $\mathbf{u}_L(\ell)$  is the number of words of  $L$  of length  $\ell$ . If  $\text{CP}_L$  exists, then  $\gamma_L$  exists and it holds:

$$\text{CP}_L = \frac{\gamma_L}{\gamma_L - 1}. \quad (2)$$

Examples show that  $\gamma_L$  may exist without  $\text{CP}_L$  exist. By virtue of this equality, the *computation* of  $\text{CP}_L$  is usually not an issue, the problem lies in proving its *existence*. We develop three different methods of existence proof, whose domains of application are pairwise incomparable: *combinatorial*, *algebraic*, and *ergodic*, and which are built upon very different mathematical backgrounds.

A combinatorial method shows that languages with *eventually periodic signature* have a carry propagation. These languages are essentially the rational base numeration systems (including the integer base numeration systems), possibly with non-canonical alphabets of digits [9].

We next consider the *rational abstract numeration systems*, that is, those systems which are defined by languages accepted by *finite automata*. Examples of such systems are the Fibonacci numeration system, more generally, beta-numeration systems where beta is a *Parry number* [6], and other systems different from beta-numeration. By means of a property of rational power series with positive coefficients which is reminiscent of

Perron-Frobenius Theorem, we prove that *the carry propagation of a rational pce language  $L$  exists if  $L$  has a local growth rate and all its quotients also have a local growth rate.*

The definition of carry propagation (Eq. 1) inevitably reminds of Ergodic Theorem. We then consider the *greedy numeration systems*. The language of greedy expansions in such a system is embedded into a compact set, and the successor function is extended as an action, called *odometer*, on that compactification. This gives a dynamical system, but Ergodic Theorem does not directly apply as the odometer is not continuous in general. Recently tools in ergodic theory [2] allow us to prove the existence of the carry propagation for greedy systems with exponential growth, and thus for beta-numeration in general.

This work was indeed motivated by a paper where the *amortized (algorithmic) complexity* of the successor function for some beta-numeration systems was studied [3]. Whatever the chosen computation model, the (amortized) complexity is greater than the (amortized) carry propagation, hence can be seen as the sum of two quantities: the carry propagation itself and an *overload*. The study of carry propagation lead to quite unexpected and winding developments that form a subject on its own, leaving the evaluation of the overload to future works. A complete version of this present work [4] will appear soon.

## References

1. Akiyama, S., Frougny, C., Sakarovitch, J.: Powers of rationals modulo 1 and rational base number systems. *Israel J. Math.* **168**, 53–91 (2008)
2. Barat, G., Grabner, P.: Combinatorial and probabilistic properties of systems of numeration. *Ergod. Th. Dynam. Sys.* **36**, 422–457 (2016)
3. Barcucci, E., Pinzani, R., Poneti, M.: Exhaustive generation of some regular languages by using numeration systems. In: *Proceedings of the Words 2005*, pp. 119–127. No. 36 in *Monographies du LaCIM, UQaM* (2005)
4. Berthé, V., Frougny, C., Rigo, M., Sakarovitch, J.: On the concrete complexity of the successor function. Part I: The carry propagation (in preparation)
5. Burks, A.W., Goldstine, H.H., von Neumann, J.: Preliminary discussion of the logical design of an electronic computing instrument. In: Taub, A.H. (ed.) *Collected Works of John von Neumann*, vol. 5, pp. 34–79. Macmillan (1963)
6. Frougny, C., Sakarovitch, J.: Number representation and finite automata. In: Berthé, V., Rigo, M. (eds.) *Combinatorics, Automata and Number Theory*, pp. 34–107. Cambridge University (2010)
7. Knuth, D.E.: The average time for carry propagation. *Nederl. Akad. Wetensch. Indag. Math.* **40**, 238–242 (1978)
8. Lecomte, P., Rigo, M.: Abstract numeration systems. In: Berthé, V., Rigo, M. (eds.) *Combinatorics, Automata and Number Theory*, pp. 34–107. Cambridge University Press (2010)
9. Marsault, V., Sakarovitch, J.: Trees and languages with periodic signature. *Indagationes Math.* **28**, 221–246 (2017)
10. Pippenger, N.: Analysis of carry propagation in addition: an elementary approach. *J. Algorithms* **42**(2), 317–333 (2002)

# Contents

Finite Automata and Randomness . . . . .	1
<i>Ludwig Staiger</i>	
Properties of Right One-Way Jumping Finite Automata . . . . .	11
<i>Simon Beier and Markus Holzer</i>	
Word Problem Languages for Free Inverse Monoids . . . . .	24
<i>Tara Brough</i>	
Most Complex Deterministic Union-Free Regular Languages . . . . .	37
<i>Janusz A. Brzozowski and Sylvie Davies</i>	
Site-Directed Insertion: Decision Problems, Maximality and Minimality . . . . .	49
<i>Da-Jung Cho, Yo-Sub Han, Kai Salomaa, and Taylor J. Smith</i>	
Two-Way Automata over Locally Finite Semirings . . . . .	62
<i>Louis-Marie Dando and Sylvain Lombardy</i>	
A New Technique for Reachability of States in Concatenation Automata . . . . .	75
<i>Sylvie Davies</i>	
Forward Injective Finite Automata: Exact and Random Generation of Nonisomorphic NFAs . . . . .	88
<i>Miguel Ferreira, Nelma Moreira, and Rogério Reis</i>	
On the Generation of 2-Polyominoes . . . . .	101
<i>Enrico Formenti and Paolo Massazza</i>	
A Local Limit Property for Pattern Statistics in Bicomponent Stochastic Models . . . . .	114
<i>Massimiliano Goldwurm, Jianyi Lin, and Marco Vignati</i>	
Linear-Time Limited Automata . . . . .	126
<i>Bruno Guillon and Luca Prigioniero</i>	
Cover Complexity of Finite Languages . . . . .	139
<i>Stefan Hetzl and Simon Wolfsteiner</i>	
On the Grammatical Complexity of Finite Languages . . . . .	151
<i>Markus Holzer and Simon Wolfsteiner</i>	
State Grammars with Stores . . . . .	163
<i>Oscar H. Ibarra and Ian McQuillan</i>	

Error-Free Affine, Unitary, and Probabilistic OBDDs. . . . .	175
<i>Rishat Ibrahimov, Kamil Khadiev, Krišjānis Prūsis, and Abuzer Yakaryılmaz</i>	
State Complexity of Unambiguous Operations on Deterministic Finite Automata . . . . .	188
<i>Galina Jirásková and Alexander Okhotin</i>	
Cycle Height of Finite Automata. . . . .	200
<i>Chris Keeler and Kai Salomaa</i>	
Finite Automata with Undirected State Graphs . . . . .	212
<i>Martin Kutrib, Andreas Malcher, and Christian Schneider</i>	
Further Closure Properties of Input-Driven Pushdown Automata . . . . .	224
<i>Alexander Okhotin and Kai Salomaa</i>	
State Complexity Characterizations of Parameterized Degree-Bounded Graph Connectivity, Sub-Linear Space Computation, and the Linear Space Hypothesis . . . . .	237
<i>Tomoyuki Yamakami</i>	
<b>Author Index</b> . . . . .	251



# Finite Automata and Randomness

Ludwig Staiger<sup>(✉)</sup>

Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg,  
06099 Halle (Saale), Germany  
[staiger@informatik.uni-halle.de](mailto:staiger@informatik.uni-halle.de)

**Abstract.** The lecture surveys approaches using finite automata to define several notions of (automata-theoretic) randomness.

It focuses on the one hand on automata-theoretic randomness of infinite sequences in connection with automata-independent notions like disjunctivity and Borel normality.

On the other hand it considers the scale of relaxations of randomness (Borel normality and disjunctivity), that is, finite-state dimension and subword complexity and their interrelations.

**Keywords:** Finite automata · Infinite words · Betting automata  
Finite-state dimension · Subword complexity

## 1 Introduction

The (algorithmic) randomness of infinite sequences can be defined by means of computability. There have been three main approaches to the definition of algorithmically random sequences, namely

1. the measure-theoretic approach,
2. the unpredictability approach, and
3. the incompressibility (or complexity-theoretic) approach.

All these approaches are based on Turing machines and were shown to be equivalent in the case of Martin-Löf random sequences. We refer the reader to the textbooks [5, 9, 10, 13] for a complete history of Martin-Löf randomness and related topics.

After Martin-Löf's measure-theoretic approach [11] and Schnorr's unpredictability approach [16] already in the 1970s sequences random with respect to finite automata were considered. It turned out that two approaches equivalent in the algorithmic case yield different characterisations of sequences which might be called "random" in the automata case. The first approach is an adaptation of the betting or martingale approach of [16] to finite automata whereas the second – in an analogy to Martin-Löf's measure theoretic approach – uses a randomness definition via null sets definable by finite automata.

Here we present a brief survey on both randomness approaches for finite automata and their relaxations which result in the finite-state dimension on the one hand and in a connection to subword complexity on the other hand.

## 2 Notation

We introduce the notation used throughout the paper. By  $\mathbb{N} = \{0, 1, 2, \dots\}$  we denote the set of natural numbers. Let  $X = \{0, \dots, r-1\}$  be a finite alphabet of cardinality  $|X| = r \geq 2$ , and  $X^*$  be the set (monoid) of words on  $X$ , including the *empty word*  $e$ , and  $X^\omega$  be the set of infinite sequences ( $\omega$ -words) over  $X$ . As usual we refer to subsets  $W \subseteq X^*$  as languages and to subsets  $F \subseteq X^\omega$  as  $\omega$ -languages.

For  $w \in X^*$  and  $\eta \in X^* \cup X^\omega$  let  $w \cdot \eta$  be their *concatenation*. This concatenation product extends in an obvious way to subsets  $W \subseteq X^*$  and  $P \subseteq X^* \cup X^\omega$ . For a language  $W$  let  $W^* := \bigcup_{i \in \mathbb{N}} W^i$  be the *submonoid* of  $X^*$  generated by  $W$ , and by  $W^\omega := \{w_1 \cdots w_i \cdots : w_i \in W \setminus \{e\}\}$  we denote the set of infinite strings formed by concatenating words in  $W$ . Furthermore  $|w|$  is the *length* of the word  $w \in X^*$  and  $\mathbf{pref}(P)$  ( $\mathbf{infix}(P)$ ) is the set of all finite prefixes (infixes) of strings in  $P \subseteq X^* \cup X^\omega$ , in particular,  $\mathbf{pref}(P) \subseteq \mathbf{infix}(P)$ . We shall abbreviate  $w \in \mathbf{pref}(\eta)$  ( $\eta \in X^* \cup X^\omega$ ) by  $w \sqsubseteq \eta$ . If  $n \leq |p|$  then  $p[0..n]$  is the  $n$ -length prefix of  $p \in X^* \cup X^\omega$ .

A (*deterministic*) *finite automaton* over  $X$  is a quintuple  $\mathcal{A} = (X, Q, q_0, \delta, Q')$  where  $Q$  is a finite set of states,  $q_0 \in Q$  the initial state,  $\delta : Q \times X \rightarrow Q$  is the transition function, and  $Q' \subseteq Q$  is the set of final states. As usual  $\delta$  also denotes the continuation of  $\delta$  to  $Q \times X^*$  defined by  $\delta(q, e) := q$  and  $\delta(q, wx) := \delta(\delta(q, w), x)$ .

A language  $W \subseteq X^*$  is called *regular* if there is a finite automaton  $\mathcal{A}$  such that  $W = \{w : \delta(q_0, w) \in Q'\}$ .

## 3 Randomness by Martingales

If one is asked why a certain event is random then often will be the answer that the event be “unpredictable”. In particular, an  $\omega$ -word  $\xi = x_1x_2 \cdots$  should be random if one cannot win by betting on its digits given other (previous) digits. For automata this yields the following.

**Definition 1 (Betting automaton).**  $\mathcal{A} = (X, Q, \mathbb{R}_{\geq 0}, q_0, \delta, \nu)$  is a finite betting automaton :  $\iff$

1.  $(X, Q, q_0, \delta)$  is a finite automaton (without final states) and
2.  $\nu : Q \times X \rightarrow \mathbb{R}_{\geq 0}$  and  $\sum_{x \in X} \nu(q, x) \leq 1$ , for all  $q \in Q$ .

The automaton starts with capital  $\mathcal{V}_{\mathcal{A}}(e) = 1$ . After the history  $w \in X^*$  its capital is  $\mathcal{V}_{\mathcal{A}}(w)$  and the automaton bets  $\nu(\delta(q_0, w), x) \cdot \mathcal{V}_{\mathcal{A}}(w)$  on every  $x$  as the outcome of the next digit. Its reward is  $r \cdot \nu(\delta(q_0, w), x) \cdot \mathcal{V}_{\mathcal{A}}(w)$  ( $r = |X|$ ) for the next digit  $x$ . This results in the following capital function (or martingale).

$$\begin{aligned} \mathcal{V}_{\mathcal{A}}(e) &:= 1, \text{ and} \\ \mathcal{V}_{\mathcal{A}}(wx) &:= r \cdot \nu(\delta(q_0, w), x) \cdot \mathcal{V}_{\mathcal{A}}(w) \end{aligned} \tag{1}$$

In order to formulate the main result we need still the following notion.



**Definition 2 (Borel normal  $\omega$ -word).** An  $\omega$ -word  $\xi \in X^\omega$  is Borel normal iff every subword (infix)  $w \in X^*$  appears with the same frequency:

$$\forall w \left( \lim_{n \rightarrow \infty} \frac{|\{i : i \leq n \wedge \xi[0..i] \in X^* \cdot w\}|}{n} \right) = r^{-|w|}$$

Then Schnorr and Stimm [17] proved the following characterisation of  $\omega$ -words random w.r.t. finite betting automata.

**Theorem 1 ([17]).** If  $\xi \in X^\omega$  is Borel normal then for every finite automaton  $\mathcal{A}$  it holds

1.  $\forall^\infty n (n \in \mathbb{N} \rightarrow \mathcal{V}_{\mathcal{A}}(\xi[0..n]) = \mathcal{V}_{\mathcal{A}}(\xi[0..n+1]))$ ,
2.  $\exists \gamma (0 > \gamma \wedge \forall^\infty n (n \in \mathbb{N} \rightarrow \mathcal{V}_{\mathcal{A}}(\xi[0..n]) \leq r^{\gamma \cdot n})$ .

If  $\xi \in X^\omega$  is not Borel normal then there are a finite automaton  $\mathcal{A}$  and  $\gamma > 0$  such that

3.  $\exists^\infty n (n \in \mathbb{N} \rightarrow \mathcal{V}_{\mathcal{A}}(\xi[0..n]) \geq r^{\gamma \cdot n})$ .

Other recent approaches to relate Borel normality to finite automata can be found e.g. in [2, 3] or [20].

## 4 Finite-State Dimension

Next we turn to aspects of partial randomness via automaton definable martingales  $\mathcal{V}_{\mathcal{A}}$ . Finite-state dimension may be seen as the estimate of the maximally achievable exponent  $\gamma$  in Theorem 1.3. To this end we define for a betting automaton  $\mathcal{A}$  and a non-empty subset  $F \subseteq X^\omega$

$$\alpha_{\mathcal{A}}(F) := \inf \left\{ \alpha : \forall \xi (\xi \in F \rightarrow \limsup_{n \rightarrow \infty} \frac{\mathcal{V}_{\mathcal{A}}(\xi[0..n])}{r^{(1-\alpha) \cdot n}} > 0) \right\} \quad (2)$$

Observe that  $1 - \alpha$  corresponds to the exponent  $\gamma$ .

Then the *finite-state dimension* of  $F$  is obtained as

$$\dim_{\text{FS}}(F) := \sup \{ \alpha_{\mathcal{A}}(F) : \mathcal{A} \text{ is a finite automaton} \} \quad (3)$$

In this definition we followed Schnorr's approach via martingales and order functions (cf. [26]) rather than the one by  $s$ -gales in [6]. If we replace  $\lim \sup$  in Eq. (2) by  $\lim \inf$  we obtain the so called strong finite-state dimension which has similar properties [7].

As an immediate consequence of Theorem 1 we obtain that  $\dim_{\text{FS}}(\xi) = 1$  if and only if  $\xi$  is Borel normal. One possibility to obtain  $\omega$ -words of smaller finite-state dimension is by dilution (inserting blocks of zeros) of Borel normal ones. In this way one proves

**Lemma 1 ([6, Lemma 6.5]).** For every rational number  $t \in \mathbb{Q} \cap [0, 1]$  there is an  $\omega$ -word  $\xi$  such that  $\dim_{\text{FS}}(\xi) = t$ .

The papers [4, 6, 7] give several equivalent definitions of finite-state dimension in terms of information-lossless compression by finite-state machines, by log-loss rates of continuous measures<sup>1</sup> on  $X^*$ , or by block-entropy rates.

Combining the results of [6] with the ones of [18, 19] in [8] it was observed that finite-state dimension has also a characterisation via decompression by transducers.

**Definition 3 (Finite transducer).**  $\mathcal{M} = (X, Y, Q, q_0, \delta, \lambda)$  is a generalised sequential machine (or finite transducer) if and only if  $(X, Q, q_0, \delta)$  is a finite automaton without final states,  $Y$  is an alphabet and  $\lambda : Q \times X \rightarrow Y^*$

The transducer realises a prefix monotone mapping  $\varphi : X^* \rightarrow Y^*$  in the following way:

$$\varphi(e) := e, \text{ and } \varphi(wx) := \varphi(w) \cdot \lambda(\delta(q_0, w), x)$$

This mapping can be extended to  $\omega$ -words via  $\mathbf{pref}(\overline{\varphi}(\eta)) = \mathbf{pref}(\varphi(\mathbf{pref}(\eta)))$ , that is,  $\overline{\varphi}(\eta) := \lim_{v \rightarrow \eta} \varphi(v)$ .

We define the decompression rate  $\vartheta_{\mathcal{M}}(\eta)$  along an input  $\eta$  as follows.

**Definition 4 (Decompression along an input).**

$$\vartheta_{\mathcal{M}}(\eta) := \liminf_{n \rightarrow \infty} \frac{n}{|\varphi(\eta[0..n])|},$$

where  $\mathcal{M}$  is a finite transducer and  $\varphi$  its related mapping.

As the difference  $|\varphi(wx) - \varphi(w)|$  is bounded, this quantity measures in some sense the asymptotic amount of digits necessary to obtain the first  $\ell$  digits of the output.

Then the finite-state dimension of  $\xi \in X^\omega$  turns out to be the simultaneous best choice of a transducer  $\mathcal{M}$  with a suitable best input  $\eta$  generating  $\xi = \overline{\varphi}(\eta)$  (cf. [6, 8, 18, 19]).

**Theorem 2.** Consider the class  $\mathcal{K}_X$  of transducers  $\mathcal{M}$  having output alphabet  $Y = X$ . Then for all  $\xi \in X^\omega$  we have

$$\dim_{\text{FS}}(\xi) = \inf \{ \vartheta_{\mathcal{M}}(\eta) : \mathcal{M} \in \mathcal{K}_X \wedge \eta \in X^\omega \wedge \xi = \overline{\varphi}(\eta) \}.$$

We conclude this section by presenting a connection between the finite-state dimension of some set  $F \subseteq X^\omega$  and the entropy of regular languages  $W$  containing  $\mathbf{pref}(F)$  [4, Theorem 3.5].

The entropy (or entropy rate)  $H_W$  of a language  $W \subseteq X^*$  is defined as [4, 22]

$$H_W := \limsup_{n \rightarrow \infty} \frac{\log_r(1 + |W \cap X^n|)}{n}. \quad (4)$$

The entropy is monotone and stable, that is,  $H_{W \cup V} = \max\{H_W, H_V\}$ . It should be mentioned that  $H_W = H_{\mathbf{pref}(W)} = H_{\mathbf{infix}(W)}$ , for regular languages.

**Theorem 3 ([4]).**  $\dim_{\text{FS}}(F) \leq \inf \{ H_W : \mathbf{pref}(F) \subseteq W \wedge W \text{ is regular} \}$

<sup>1</sup> These measures were called predictors in [4].

## 5 Automaton Definable Null Sets

We start this section with introducing  $\omega$ -languages definable by finite automata. For more background see the books [15, 28] or the surveys [23, 27].

Let  $\mathcal{B} = (X, Q, \Delta, q_0, Q')$  be a non-deterministic (Büchi-)automaton. Then the sequence  $(q_i, \xi(i+1), q_{i+1})_{i \in \mathbb{N}}$  is a *run* of  $\mathcal{B}$  on the  $\omega$ -word  $\xi = \xi(1) \cdot \xi(2) \cdots$  provided  $(q_i, \xi(i+1), q_{i+1}) \in \Delta$  for all  $i \in \mathbb{N}$ . A run is called *successful* if infinitely many of the  $q_i$  are in the set of final states  $Q'$ .

The  $\omega$ -language  $L_\omega(\mathcal{B})$  defined by  $\mathcal{B}$  is then

$$L_\omega(\mathcal{B}) = \{\xi : \xi \in X^\omega \wedge \text{there is a successful run of } \mathcal{B} \text{ on } \xi\}.$$

**Definition 5 (Regular  $\omega$ -language).** *An  $\omega$ -language  $F \subseteq X^\omega$  is called regular if and only if  $F$  is accepted by a finite automaton*

The following properties of the class of regular (automaton definable)  $\omega$ -languages are well-known.

**Theorem 4.** *1. An  $\omega$ -language  $F \subseteq X^\omega$  is regular if and only if there are an  $n \in \mathbb{N}$  and regular languages  $W_i, V_i \subseteq X^*, i \leq n$ , such that  $F = \bigcup_{i=1}^n W_i \cdot V_i^\omega$ .  
2. The set of regular  $\omega$ -languages over  $X$  is closed under Boolean operations.  
3. If  $F \subseteq X^\omega$  is regular then  $\mathbf{pref}(F)$  and  $\mathbf{infix}(F)$  are regular languages.*

**Theorem 5.** *Let  $\mathcal{DB}$  be the class of  $\omega$ -languages accepted by deterministic Büchi automata. Then*

*1.  $\mathcal{DB}$  is a proper subclass of the class of regular  $\omega$ -languages, and  
2.  $\mathcal{DB}$  is closed under union and intersection but not under complementation.  
3. If  $W \subseteq X^*$  is a regular language then  $\{\xi : \xi \in X^\omega \wedge \mathbf{pref}(\xi) \subseteq W\} \in \mathcal{DB}$  and  $\{\xi : \xi \in X^\omega \wedge |\mathbf{pref}(\xi) \cap W| = \infty\} \in \mathcal{DB}$ .*

As measure on the space  $X^\omega$  we use the usual product measure  $\mu$  defined by its values on the cylinder sets  $\mu(w \cdot X^\omega) := r^{-|w|}$ . Then in [21, 24] the following characterisation of regular null sets via “forbidden subwords” is proved.

**Theorem 6.** *Let  $F$  be a regular  $\omega$ -language.*

*1. If  $F \in \mathcal{DB}$  then  $\mu(F) = 0$  if and only if there is word  $w \in X^*$  such that  $F \subseteq X^\omega \setminus X^* \cdot w \cdot X^\omega$ .  
2.  $\mu(F) = 0$  if and only if  $F \subseteq \bigcup_{w \in X^*} X^\omega \setminus X^* \cdot w \cdot X^\omega$ .*

*Remark 1.* Theorem 6 holds for a much larger class of finite measures on  $X^\omega$  including all non-degenerated product measures on  $X^\omega$  (cf. [21, 24, 29, 30]).

Now we can characterise those  $\omega$ -words which are not contained in a regular  $\omega$ -language of measure zero.

**Definition 6 (Disjunctivity).** *An  $\omega$ -word  $\xi \in X^\omega$  is called disjunctive (or rich or saturated) if and only if it contains every word  $w \in X^*$  as subword (infix).*

Consequently,  $\omega$ -words random w.r.t. finite automata in the sense of the measure theoretic approach are exactly the disjunctive ones. This allows us to compare both of the presented approaches of randomness.

**Proposition 1.** *Every Borel normal  $\omega$ -word is disjunctive, but there are disjunctive  $\omega$ -words which are not Borel normal, e.g. the  $\omega$ -word  $\zeta := \prod_{w \in X^*} 0^{|w|} \cdot w$ .*

## 6 Subword Complexity

The characterisation via “forbidden subwords” enables us to derive a notion of partial randomness similar to the finite-state dimension. To this end we use the entropy of languages defined in Eq. (4) and define for arbitrary  $P \subseteq X^* \cup X^\omega$

**Definition 7 (Subword complexity).**

$$\tau(P) := H_{\mathbf{infix}(P)}$$

In view of the inequality  $\mathbf{infix}(P) \cap X^{n+m} \subseteq (\mathbf{infix}(P) \cap X^n) \cdot (\mathbf{infix}(P) \cap X^m)$  which holds for  $\mathbf{infix}(P)$  the limit in Eq. (4) exists and equals

$$\tau(P) = \inf \left\{ \frac{\log_r(1 + |\mathbf{infix}(P) \cap X^n|)}{n} : n \in \mathbb{N} \right\}.$$

This value is also known as factor complexity in automata theory and topological entropy in symbolic dynamics.

The following is clear.

**Proposition 2.**  $0 \leq \tau(\xi) \leq 1$  and an  $\omega$ -word  $\xi \in X^\omega$  is disjunctive if and only if  $\tau(\xi) = 1$ .

For subword complexity one has for every possible value an  $\omega$ -word of exactly this complexity [12].

**Theorem 7.** For every  $t, 0 \leq t \leq 1$ , there is a  $\xi \in X^\omega$  such that  $\tau(\xi) = t$ .

Similar to Eq. (5.1.2) of [22] one can derive the following identity.

$$\tau(P) = \inf \{ H_W : W \subseteq X^* \wedge \mathbf{infix}(P) \subseteq W \wedge W \text{ is regular} \} \quad (5)$$

Now Theorem 3 yields the following relation to finite-state dimension.

$$\dim_{\text{FS}} F \leq \tau(F) \quad (6)$$

For certain regular  $\omega$ -languages  $F \subseteq X^\omega$  we have identity in Eq. (6).

**Proposition 3.** Let  $F \subseteq X^\omega$  be non-empty and regular.

1. Then  $\max_{\xi \in F} \tau(\xi)$  exists and  $\max_{\xi \in F} \tau(\xi) = \max_{\xi \in F} \dim_{\text{FS}} \{\xi\}$ .
2. If, moreover,  $F = \{\xi : \mathbf{pref}(\xi) \subseteq \mathbf{pref}(F)\}$  then  $\tau(F) = \dim_{\text{FS}} F$ .

## 7 Predicting Finite Automata

A further feature of randomness of an  $\omega$ -word  $\xi$ , similar to the one mentioned for betting automata, is the impossibility of the exact prediction of the next symbol. Here Tadaki [25] proposed the following.

**Definition 8 (Predicting automaton).** A transducer  $\mathcal{A} = (X, X, Q, q_0, \delta, \lambda)$  is referred to as a predicting automaton if  $\lambda : Q \rightarrow \{e\} \cup X$  is a labelling of states.

**Definition 9 (Prediction).** A predicting automaton  $\mathcal{A} = (X, X, Q, q_0, \delta, \lambda)$  strongly predicts  $\xi \in X^\omega$  if and only if

1.  $\lambda(\delta(q_0, \xi[0..n-1])) = \xi(n)$  for infinitely many  $n \in \mathbb{N}$ , and
2. if  $\lambda(\delta(q_0, \xi[0..n-1])) \neq \xi(n)$  then  $\lambda(\delta(q_0, \xi[0..n-1])) = e$ .

Definition 9 is a strong requirement, it forces the automaton to make on input  $\xi$  infinitely many correct predictions and no incorrect ones. Here using the label  $\lambda(q) = e$  the automaton may skip. Nevertheless, in the binary case  $X = \{0, 1\}$  we have the following.

**Theorem 8.** 1. Let  $\mathcal{A} = (\{0, 1\}, \{0, 1\}, Q, q_0, \delta, \lambda)$  be a binary predicting automaton. If  $\mathcal{A}$  strongly predicts  $\xi \in \{0, 1\}^\omega$  then  $\xi$  is not disjunctive.  
 2. If  $\xi \in \{0, 1\}^\omega$  is disjunctive then no predicting automaton predicts  $\xi$ .

This theorem does not hold in the other cases when  $|X| \geq 3$ . Here we have to turn to “negative” prediction. We say that  $\mathcal{A}$  weakly predicts  $\xi$  provided  $\lambda(\delta(q_0, \xi[0..n-1])) \neq \xi(n)$  for infinitely many  $n \in \mathbb{N}$  and  $\lambda(\delta(q_0, \xi[0..n-1])) = e$  otherwise. Then we have.

**Theorem 9.** 1. Let  $\mathcal{A} = (X, X, Q, q_0, \delta, \lambda)$  be a binary predicting automaton. If  $\mathcal{A}$  weakly predicts  $\xi \in X^\omega$  then  $\xi$  is not disjunctive.  
 2. If  $\xi \in X^\omega$  is disjunctive then no predicting automaton weakly predicts  $\xi$ .

## 8 Finite-State Genericity

This section reviews some connections between disjunctivity and finite-state genericity. As in [1] we define the following.

**Definition 10.** Let  $\xi \in X^\omega$ .

1.  $\xi$  meets a function  $\psi : X^* \rightarrow X^*$  if  $w \cdot \psi(w) \sqsubset \xi$ .
2.  $\xi$  is finite-state generic if  $\xi$  meets every function  $\varphi$  realised by a finite transducer.

This can be interpreted in terms of the usual product topology on  $X^\omega$  which can be defined by the metric  $\varrho(\xi, \eta) := \sup\{r^{-n} : \xi(n) \neq \eta(n)\}$  where we agree on  $\sup \emptyset = 0$ . The cylinder sets  $w \cdot X^\omega$  are simultaneously open and closed balls

of diameter  $r^{-|w|}$ . The closure  $\mathcal{C}(F)$  of (smallest closed set containing) a set  $F \subseteq X^\omega$  obtains as  $\mathcal{C}(F) = \{\xi : \mathbf{pref}(\xi) \subseteq \mathbf{pref}(F)\}$ .

A subset  $F$  is *nowhere dense* if its closure does not contain a non-empty open subset, that is, for every  $w \in \mathbf{pref}(F)$  there is a continuation  $v \in X^*$  such that  $w \cdot v \cdot X^\omega \cap F = \emptyset$ , that is “ $v$  leads  $w$  to a hole” in  $F$ .

Then Definition 10.2 gives an indication that finite-state generic  $\omega$ -words avoid “finite-state nowhere dense” subsets of  $X^\omega$ . This is shown by Theorem 4.4 of [1].

**Theorem 10** ([1]). *An  $\omega$ -word  $\xi$  is disjunctive if and only if it is finite-state generic.*

Theorem 10 fits into the more general coincidence of measure and category for regular  $\omega$ -languages depicted in Fig. 1 (see [21, 24, 29, 30]). In the general case, however, the monograph [14] shows that measure and category (topological density) are two concepts which do not coincide.

	Measure	Category (Density)
very large	$\mu(F) = \mu(X^\omega)$	$F$ is residual (co-meagre)
large	$\mu(F) \neq 0$	$F$ is of 2 <sup>nd</sup> BAIRE category
small	$\mu(F) = 0$	$F$ is of 1 <sup>st</sup> BAIRE category (meagre)
very small	$\mu(\mathcal{C}(F)) = 0$	$F$ is nowhere dense

**Fig. 1.** Coincidence of measure and category for regular  $\omega$ -languages

As usual a subset  $F$  is *meagre* or of *first Baire category* if it is at most countable union of nowhere dense sets, a set is of *second Baire category* if it is not meagre, and it is *residual* if its complement is meagre. The first column of Fig. 1 presents a comparison of the sizes of  $F \subseteq X^\omega$ , and the rows indicate that for regular  $\omega$ -languages  $F \subseteq X^\omega$  properties of the same row coincide, e.g.  $\mu(F) = 0$  iff  $F$  is meagre.

## References

1. Ambos-Spies, K., Busse, E.: Automatic forcing and genericity: on the diagonalization strength of finite automata. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCs 2003. LNCS, vol. 2731, pp. 97–108. Springer, Heidelberg (2003). <https://doi.org/10.1007/3-540-45066-1-7>
2. Becher, V., Carton, O., Heiber, P.A.: Normality and automata. J. Comput. Syst. Sci. **81**(8), 1592–1613 (2015)
3. Becher, V., Heiber, P.A.: Normal numbers and finite automata. Theor. Comput. Sci. **477**, 109–116 (2013)
4. Bourke, C., Hitchcock, J.M., Vinodchandran, N.: Entropy rates and finite-state dimension. Theor. Comput. Sci. **349**(3), 392–406 (2005)

5. Calude, C.S.: Information and Randomness. An Algorithmic Perspective. Texts in Theoretical Computer Science. An EATCS Series, 2nd edn. Springer, Berlin (2002). <https://doi.org/10.1007/978-3-662-04978-5>. With forewords by Gregory J. Chaitin and Arto Salomaa
6. Dai, J.J., Lathrop, J.I., Lutz, J.H., Mayordomo, E.: Finite-state dimension. *Theor. Comput. Sci.* **310**(1–3), 1–33 (2004)
7. Doty, D., Lutz, J.H., Nandakumar, S.: Finite-state dimension and real arithmetic. *Inf. Comput.* **205**(11), 1640–1651 (2007)
8. Doty, D., Moser, P.: Finite-state dimension and lossy decompressors. *CoRR abs/cs/0609096* (2006). <http://arxiv.org/abs/cs/0609096>
9. Downey, R.G., Hirschfeldt, D.R.: Algorithmic Randomness and Complexity. Theory and Applications of Computability. Springer, New York (2010)
10. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and Its Applications. Texts and Monographs in Computer Science. Springer, New York (1993). <https://doi.org/10.1007/978-1-4757-3860-5>
11. Martin-Löf, P.: The definition of random sequences. *Inf. Control* **9**, 602–619 (1966)
12. Moldagaliyev, B., Staiger, L., Stephan, F.: On the values for factor complexity (2018, to appear)
13. Nies, A.: Computability and Randomness, Oxford Logic Guides, vol. 51. Oxford University Press, Oxford (2009). <https://doi.org/10.1093/acprof:oso/9780199230761.001.0001>
14. Oxtoby, J.C.: Measure and Category, Graduate Texts in Mathematics, vol. 2, 2nd edn. Springer, New York (1980). A survey of the analogies between topological and measure spaces
15. Perrin, D., Pin, J.E.: Infinite Words. Automata, Semigroups, Logic and Games. Elsevier/Academic Press, Amsterdam (2004)
16. Schnorr, C.P.: Zufälligkeit und Wahrscheinlichkeit. LNM, vol. 218. Springer, Heidelberg (1971). <https://doi.org/10.1007/BFb0112458>
17. Schnorr, C.P., Stimm, H.: Endliche Automaten und Zufallsfolgen. *Acta Inf.* **1**, 345–359 (1972)
18. Sheinwald, D., Lempel, A., Ziv, J.: On compression with two-way head machines. In: Storer, J.A., Reif, J.H. (eds.) Proceedings of the IEEE Data Compression Conference, DCC 1991, Snowbird, Utah, 8–11 April 1991, pp. 218–227. IEEE Computer Society (1991). <https://doi.org/10.1109/DCC.1991.213359>
19. Sheinwald, D., Lempel, A., Ziv, J.: On encoding and decoding with two-way head machines. *Inf. Comput.* **116**(1), 128–133 (1995)
20. Shen, A.: Automatic Kolmogorov complexity and normality revisited. In: Klasing, R., Zeitoun, M. (eds.) FCT 2017. LNCS, vol. 10472, pp. 418–430. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-55751-8\\_33](https://doi.org/10.1007/978-3-662-55751-8_33)
21. Staiger, L.: Reguläre Nullmengen. *Elektron. Informationsverarbeitung. Kybernetik* **12**(6), 307–311 (1976)
22. Staiger, L.: Kolmogorov complexity and Hausdorff dimension. *Inform. Comput.* **103**(2), 159–194 (1993). <https://doi.org/10.1006/inco.1993.1017>
23. Staiger, L.:  $\omega$ -languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 339–387. Springer, Berlin (1997). Beyond Words
24. Staiger, L.: Rich  $\omega$ -words and monadic second-order arithmetic. In: Nielsen, M., Thomas, W. (eds.) CSL 1997. LNCS, vol. 1414, pp. 478–490. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0028032>
25. Tadaki, K.: Phase transition and strong predictability. In: Ibarra, O.H., Kari, L., Kopecki, S. (eds.) UCNC 2014. LNCS, vol. 8553, pp. 340–352. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08123-6\\_28](https://doi.org/10.1007/978-3-319-08123-6_28)

26. Terwijn, S.A.: Complexity and randomness. *Rend. Semin. Mat.* **62**(1), 1–37 (2004). Torino
27. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science*, vol. B, pp. 133–191. Elsevier, Amsterdam (1990). *Formal Models and Semantics*
28. Trakhtenbrot, B.A., Barzdin̄, Y.M.: *Finite Automata*. North-Holland Publishing Co., Amsterdam (1973). *Behavior and Synthesis*, Translated from the Russian by D. Louvish, English translation edited by E. Shamir and L. H. Landweber, *Fundamental Studies in Computer Science*, vol. 1
29. Varacca, D., Völzer, H.: Temporal logics and model checking for fairly correct systems. In: 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12–15 August 2006, Seattle, WA, USA, Proceedings, pp. 389–398. IEEE Computer Society (2006). <https://doi.org/10.1109/LICS.2006.49>
30. Völzer, H., Varacca, D.: Defining fairness in reactive and concurrent systems. *J. ACM* **59**(3), Article no. 13, 37 (2012). <https://doi.org/10.1145/2220357.2220360>





# Properties of Right One-Way Jumping Finite Automata

Simon Beier and Markus Holzer<sup>(✉)</sup>

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

**Abstract.** Right one-way jumping finite automata (ROWJFAs), were recently introduced in [H. CHIGAHARA, S. Z. FAZEKAS, A. YAMAMURA: One-Way Jumping Finite Automata, *Internat. J. Found. Comput. Sci.*, 27(3), 2016] and are jumping automata that process the input in a discontinuous way with the restriction that the input head reads deterministically from left-to-right starting from the leftmost letter in the input and when it reaches the end of the input word, it returns to the beginning and continues the computation. We solve most of the open problems of these devices. In particular, we characterize the family of permutation closed languages accepted by ROWJFAs in terms of Myhill-Nerode equivalence classes. Using this, we investigate closure and non-closure properties as well as inclusion relations to other language families. We also give more characterizations of languages accepted by ROWJFAs for some interesting cases.

## 1 Introduction

Jumping finite automata [11] are a machine model for discontinuous information processing. Roughly speaking, a jumping finite automaton is an ordinary finite automaton, which is allowed to read letters from anywhere in the input string, not necessarily only from the left of the remaining input. In a series of papers [1, 6, 7, 13] different aspects of jumping finite automata were investigated, such as, e.g., inclusion relations, closure and non-closure results, decision problems, computational complexity of jumping finite automata problems, etc. Shortly after the introduction of jumping automata, a variant of this machine model was defined, namely (right) one-way jumping finite automata [3]. There the device moves the input head deterministically from left-to-right starting from the leftmost letter in the input and when it reaches the end of the input word, it returns to the beginning and continues the computation. As in the case of ordinary jumping finite automata inclusion relations to well-known formal language families, closure and non-closure results under standard formal language operations were investigated. Nevertheless, a series of problems on right one-way jumping automata (ROWJFAs) remained open in [3]. This is the starting point of our investigation.

First we develop a characterization of (permutation closed) languages that are accepted by ROWJFAs in terms of the Myhill-Nerode relation. It is shown that the permutation closed language  $L$  belongs to **ROWJ**, the family of all languages accepted by ROWJFAs, if and only if  $L$  can be written as the *finite union* of Myhill-Nerode equivalence classes. Observe, that the overall number of equivalence classes can be infinite. This result nicely contrasts the characterization of regular languages, which requires that the overall number of equivalence classes is finite. The characterization allows us to identify languages that are *not* accepted by ROWJFAs, which are useful to prove non-closure results on standard formal language operations. In this way we solve all of the open problems from [3] on the inclusion relations of ROWJFAs languages to other language families and on their closure properties. It is shown that the family **ROWJ** is an anti-abstract family of languages (anti-AFL), that is, it is not closed under any of the operations  $\lambda$ -free homomorphism, inverse homomorphism, intersection with regular sets, union, concatenation, or Kleene star. This is a little bit surprising for a language family defined by a deterministic automaton model. Although anti-AFLs are sometimes referred to an “unfortunate family of languages” there is linguistical evidence that such language families might be of crucial importance, since in [4] it was shown that the family of natural languages is an anti-AFL. On the other hand, the permutation closed languages in **ROWJ** almost form an anti-AFL, since this language family is closed under inverse homomorphism. Moreover, we obtain further characterizations of languages accepted by ROWJFAs. For instance, we show that

1. language  $wL$  is in **ROWJ** if and only if  $L$  is in **ROWJ**,
2. language  $Lw$  is in **ROWJ** if and only if  $L$  is regular, and
3. language  $L_1L_2$  is in **ROWJ** if and only if  $L_1$  is regular and  $L_2$  is in **ROWJ**, where  $L_1$  and  $L_2$  have to fulfil some further easy pre-conditions.

The latter result is in similar vein as a result in [9] on linear context-free languages, where it was shown that  $L_1L_2$  is a linear context-free language if and only if  $L_1$  is regular and  $L_2$  at most linear context free. Finally another characterization is given for letter bounded ROWJFA languages, namely, the language  $L \subseteq a_1^*a_2^*\dots a_n^*$  is in **ROWJ** if and only if  $L$  is regular. This result nicely generalizes the fact that every unary language accepted by an ROWJFA is regular.

## 2 Preliminaries

We assume the reader to be familiar with the basics in automata and formal language theory as contained, for example, in [10]. Let  $\mathbb{N} = \{0, 1, 2, \dots\}$  be the set of non-negative integers. We use  $\subseteq$  for inclusion, and  $\subset$  for proper inclusion. Let  $\Sigma$  be an alphabet. Then  $\Sigma^*$  is the set of all words over  $\Sigma$ , including the empty word  $\lambda$ . For a language  $L \subseteq \Sigma^*$  define the set  $\text{perm}(L) = \cup_{w \in L} \text{perm}(w)$ , where  $\text{perm}(w) = \{v \in \Sigma^* \mid v \text{ is a permutation of } w\}$ . Then a language  $L$  is called *permutation closed* if  $L = \text{perm}(L)$ . The length of a word  $w \in \Sigma^*$  is

denoted by  $|w|$ . For the number of occurrences of a symbol  $a$  in  $w$  we use the notation  $|w|_a$ . We denote the powerset of a set  $S$  by  $2^S$ . For  $\Sigma = \{a_1, a_2, \dots, a_k\}$ , the *Parikh-mapping*  $\psi : \Sigma^* \rightarrow \mathbb{N}^k$  is the function  $w \mapsto (|w|_{a_1}, |w|_{a_2}, \dots, |w|_{a_k})$ . A language  $L \subseteq \Sigma^*$  is called *semilinear* if its *Parikh-image*  $\psi(L)$  is a semilinear subset of  $\mathbb{N}^k$ , a definition of those can be found in [8].

The elements of  $\mathbb{N}^k$  can be partially ordered by the  $\leq$ -relation on vectors. For vectors  $\mathbf{x}, \mathbf{y} \in \mathbb{N}^k$  we write  $\mathbf{x} \leq \mathbf{y}$  if all components of  $\mathbf{x}$  are less or equal to the corresponding components of  $\mathbf{y}$ . The value  $\|\mathbf{x}\|$  is the maximum norm of  $\mathbf{x}$ , that is,  $\|(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k)\| = \max\{|\mathbf{x}_i| \mid 1 \leq i \leq k\}$ .

For  $v, w \in \Sigma^*$ , we say that  $v$  is a prefix of  $w$  if there is an  $x \in \Sigma^*$  with  $w = vx$ . Moreover,  $v$  is a sub-word of  $w$  if there are  $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_{n+1} \in \Sigma^*$  with  $v = x_1x_2 \cdots x_n$  and  $w = y_1x_1y_2x_2 \cdots y_nx_ny_{n+1}$ , for some  $n \geq 0$ . A language  $L \subseteq \Sigma^*$  is called prefix-free if and only if there are no words  $v, w \in L$  such that  $v \neq w$  and  $v$  is a prefix of  $w$ .

For an alphabet  $\Sigma$  and a language  $L \subseteq \Sigma^*$ , let  $\sim_L$  be the *Myhill-Nerode equivalence relation* on  $\Sigma^*$ . So, for  $v, w \in \Sigma^*$ , we have  $v \sim_L w$  if and only if, for all  $u \in \Sigma^*$ , the equivalence  $vu \in L \Leftrightarrow wu \in L$  holds. For  $w \in \Sigma^*$ , we call the equivalence class  $[w]_{\sim_L}$  positive if and only if  $w \in L$ . Otherwise, the equivalence class  $[w]_{\sim_L}$  is called negative.

A *deterministic finite automaton*, a DFA for short, is defined as a tuple  $A = (Q, \Sigma, R, s, F)$ , where  $Q$  is the finite set of states,  $\Sigma$  is the finite input alphabet,  $\Sigma \cap Q = \emptyset$ ,  $R$  is a partial function from  $Q \times \Sigma$  to  $Q$ ,  $s \in Q$  is the start state, and  $F \subseteq Q$  is the set of final states. The elements of  $R$  are referred to a rules of  $A$  and we write  $py \rightarrow q \in R$  instead of  $R(p, y) = q$ . A configuration of  $A$  is a string in  $Q\Sigma^*$ . A DFA makes a transition from configuration  $paw$  to configuration  $qw$  if  $pa \rightarrow q \in R$ , where  $p, q \in Q$ ,  $a \in \Sigma$ , and  $w \in \Sigma^*$ . We denote this by  $paw \vdash_A qw$  or just  $paw \vdash qw$  if it is clear which DFA we are referring to. In the standard manner, we extend  $\vdash$  to  $\vdash^n$ , where  $n \geq 0$ . Let  $\vdash^+$  and  $\vdash^*$  denote the transitive closure of  $\vdash$  and the transitive-reflexive closure of  $\vdash$ , respectively. Then, the language accepted by  $A$  is  $L(A) = \{w \in \Sigma^* \mid \exists f \in F : sw \vdash^* f\}$ . We say that  $A$  accepts  $w \in \Sigma^*$  if  $w \in L(A)$  and that  $A$  rejects  $w$  otherwise. The family of languages accepted by DFAs is referred to as **REG**.

A *jumping finite automaton*, a JFA for short, is a tuple  $A = (Q, \Sigma, R, s, F)$ , where  $Q$ ,  $\Sigma$ ,  $R$ ,  $s$ , and  $F$  are the same as in the case of DFAs. A configuration of  $A$  is a string in  $\Sigma^*Q\Sigma^*$ . The binary jumping relation, symbolically denoted by  $\curvearrowright_A$ , over  $\Sigma^*Q\Sigma^*$  is defined as follows. Let  $x, z, x', z'$  be strings in  $\Sigma^*$  such that  $xz = x'z'$  and  $py \rightarrow q \in R$ . Then, the automaton  $A$  makes a jump from  $xpyz$  to  $x'qz'$ , symbolically written as  $xpyz \curvearrowright_A x'qz'$  or just  $xpyz \curvearrowright x'qz'$  if it is clear which JFA we are referring to. In the standard manner, we extend  $\curvearrowright$  to  $\curvearrowright^n$ , where  $n \geq 0$ . Let  $\curvearrowright^+$  and  $\curvearrowright^*$  denote the transitive closure of  $\curvearrowright$  and the transitive-reflexive closure of  $\curvearrowright$ , respectively. Then, the language accepted by  $A$  is  $L(A) = \{w \mid u, v \in \Sigma^*, \exists f \in F : usv \curvearrowright^* f\}$ . We say that  $A$  accepts  $w \in \Sigma^*$  if  $w \in L(A)$  and that  $A$  rejects  $w$  otherwise. Let **JFA** be the family of all languages that are accepted by JFAs.

A *right one-way jumping finite automaton*, a ROWJFA for short, is a tuple  $A = (Q, \Sigma, R, s, F)$ , where  $Q$ ,  $\Sigma$ ,  $R$ ,  $s$ , and  $F$  are defined as in a DFA. A configuration of  $A$  is a string in  $Q\Sigma^*$ . The right one-way jumping relation, symbolically denoted by  $\circlearrowright_A$ , over  $Q\Sigma^*$  is defined as follows. For  $p \in Q$  we set

$$\Sigma_p = \Sigma_{R,p} = \{b \in \Sigma \mid pb \rightarrow q \in R \text{ for some } q \in Q\}.$$

Now, let  $pa \rightarrow q \in R$ ,  $x \in (\Sigma \setminus \Sigma_p)^*$ , and  $y \in \Sigma^*$ . Then, the ROWJFA  $A$  makes a jump from the configuration  $pxay$  to the configuration  $qyx$ , symbolically written as  $pxay \circlearrowright_A qyx$  or just  $pxay \circlearrowright qyx$  if it is clear which ROWJFA we are referring to. In the standard manner, we extend  $\circlearrowright$  to  $\circlearrowright^n$ , where  $n \geq 0$ . Let  $\circlearrowright^+$  and  $\circlearrowright^*$  denote the transitive closure of  $\circlearrowright$  and the transitive-reflexive closure of  $\circlearrowright$ , respectively. The language accepted by  $A$  is  $L(A) = \{w \in \Sigma^* \mid \exists f \in F : sw \circlearrowright^* f\}$ . We say that  $A$  accepts  $w \in \Sigma^*$  if  $w \in L(A)$  and that  $A$  rejects  $w$  otherwise. Let **ROWJ** be the family of all languages that are accepted by ROWJFAs. Furthermore, for  $n \geq 0$ , let **ROWJ<sub>n</sub>** be the class of all languages accepted by ROWJFAs with at most  $n$  accepting states.

Besides the above mentioned language families let **FIN**, **DCF**, **CF**, and **CS** be the families of finite, deterministic context-free, context-free, and context-sensitive languages. Moreover, we are interested in permutation closed language families. These language families are referred to by a prefix **p**. E.g., **pROWJ** denotes the language family of all permutation closed **ROWJ** languages.

Sometimes, for a DFA  $A$ , we will also consider the relations  $\rightsquigarrow$  and  $\circlearrowright$ , that we get by interpreting  $A$  as a JFA or a ROWJFA. The following three languages are associated to  $A$ :

- $L_D(A)$  is the language accepted by  $A$ , interpreted as an ordinary DFA.
- $L_J(A)$  is the language accepted by  $A$ , interpreted as an JFA.
- $L_R(A)$  is the language accepted by  $A$ , interpreted as an ROWJFA.

From a result in [12] and from [3, Theorem 10], we get

$$L_D(A) \subseteq L_R(A) \subseteq L_J(A) = \text{perm}(L_D(A)). \quad (1)$$

As a consequence, we have **JFA** = **pJFA**. Next we give an example of a DFA  $A$  with  $L_D(A) \subset L_R(A) \subset L_J(A)$ .

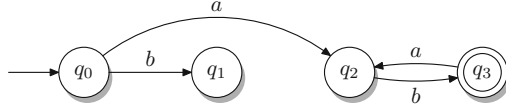
*Example 1.* Let  $A$  be the DFA

$$A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, R, q_0, \{q_3\}),$$

where  $R$  consists of the rules  $q_0b \rightarrow q_1$ ,  $q_0a \rightarrow q_2$ ,  $q_2b \rightarrow q_3$ , and  $q_3a \rightarrow q_2$ . The automaton  $A$  is depicted in Fig. 1.

It holds  $L_D(A) = (ab)^+$  and

$$L_J(A) = \text{perm}((ab)^+) = \{w \in \{a, b\}^+ \mid |w|_a = |w|_b\}.$$



**Fig. 1.** The automaton  $A$  with  $L_D(A) \subset L_R(A) \subset L_J(A)$ .

To show how ROWJFAs work, we give an example computation of  $A$ , interpreted as an ROWJFA, on the input word  $aabbba$ :

$$q_0 a a b b b a \circlearrowleft q_2 a b b b a \circlearrowleft q_3 b b a a \circlearrowleft q_2 a b b \circlearrowleft q_3 b a \circlearrowleft q_2 b \circlearrowleft q_3$$

That shows  $aabbba \in L_R(A)$ . Analogously, one can see that every word that contains the same number of  $a$ 's and  $b$ 's and that begins with an  $a$  is in  $L_R(A)$ . On the other hand, no other word can be accepted by  $A$ , interpreted as an ROWJFA. So, we get  $L_R(A) = \{w \in a\{a, b\}^* \mid |w|_a = |w|_b\}$ . Notice that this language is non-regular and not closed under permutation.  $\square$

The following basic property will be used later on.

**Lemma 2.** *Let  $A = (Q, \Sigma, R, s, F)$  be a DFA. Consider two words  $v, w \in \Sigma^*$ , states  $p, q \in Q$ , and an  $n \geq 0$  with  $pv \circlearrowleft^n qw$ . Then, there is a word  $x \in \Sigma^*$  such that  $xw$  is a permutation of  $v$ , and  $px \vdash^n q$ .*

*Proof.* We prove this by induction on  $n$ . If  $n = 0$ , we have  $pv = qw$  and just set  $x = \lambda$ . Now, assume  $n > 0$  and that the lemma is true for the relation  $\circlearrowleft^{n-1}$ . We get a state  $r \in Q$ , a symbol  $a \in \Sigma_r$ , and words  $y \in (\Sigma \setminus \Sigma_r)^*$  and  $z \in \Sigma^*$  such that  $w = zy$  and  $pv \circlearrowleft^{n-1} ryaz \circlearrowleft qw$ . By the induction hypothesis, there is an  $x' \in \Sigma^*$  such that  $x'yaz$  is a permutation of  $v$ , and  $px' \vdash^{n-1} r$ . Set  $x = x'a$ . Then, the word  $xw = x'azy$  is a permutation of  $x'yaz$ , which is a permutation of  $v$ . Furthermore, we get  $px = px'a \vdash^{n-1} ra \vdash q$ . This proves the lemma.  $\square$

### 3 A Characterization of Permutation Closed Languages Accepted by ROWJFAS

By the Myhill-Nerode theorem, a language  $L$  is regular if and only if the Myhill-Nerode relation  $\sim_L$  has only a finite number of equivalence classes. Moreover, the number of equivalence classes equals the number of states of the minimal DFA accepting  $L$ , see for example [10]. We can give a similar characterization for permutation closed languages that are accepted by an ROWJFA.

**Theorem 3.** *Let  $L$  be a permutation closed language and  $n \geq 0$ . Then, the language  $L$  is in  $\mathbf{ROWJ}_n$  if and only if the Myhill-Nerode relation  $\sim_L$  has at most  $n$  positive equivalence classes.*

*Proof.* First, assume that  $L$  is in **ROWJ<sub>n</sub>** and let  $A = (Q, \Sigma, R, s, F)$  be a DFA with  $|F| \leq n$  and  $L_R(A) = L$ . Consider  $v, w \in L$  and  $f \in F$  with  $sv \circ^* f$  and  $sw \circ^* f$ . Lemma 2 shows that there are permutations  $v'$  and  $w'$  of  $v$  and  $w$  with  $sv' \vdash^* f$  and  $sw' \vdash^* f$ . Because language  $L$  is closed under permutation we have  $v \sim_L v'$  and  $w \sim_L w'$ . Now, let  $u \in \Sigma^*$ . Thus  $sv'u \circ^* fu$  and  $sw'u \circ^* fu$ . That gives us

$$v'u \in L \Leftrightarrow (\exists g \in F : fu \circ^* g) \Leftrightarrow w'u \in L.$$

We have shown  $v \sim_L v' \sim_L w' \sim_L w$ . From  $L = \bigcup_{f \in F} \{w \in \Sigma^* \mid sw \circ^* f\}$ , we get  $|L / \sim_L| \leq |F| \leq n$ , which means that  $\sim_L$  has at most  $n$  positive equivalence classes.

Assume now that  $\sim_L$  has at most  $n$  positive equivalence classes and let  $\Sigma = \{a_1, a_2, \dots, a_k\}$  be an alphabet with  $L \subseteq \Sigma^*$ . Set  $L_\lambda = L \cup \{\lambda\}$ . Define the map  $S : L_\lambda / \sim_L \rightarrow 2^{\mathbb{N}^k}$  through  $[w] \mapsto \{\mathbf{x} \in \mathbb{N}^k \setminus \mathbf{0} \mid \psi^{-1}(\psi(w) + \mathbf{x}) \subseteq L\}$ . The definition of  $\sim_L$  and the fact that  $L$  is closed under permutation make the map  $S$  well-defined. Consider the relation  $\leq$  on  $\mathbb{N}^k$ . For each  $[w] \in L_\lambda / \sim_L$ , let  $M([w])$  be the set of minimal elements of  $S([w])$ . So, for every  $[w] \in L_\lambda / \sim_L$  and  $\mathbf{x} \in S([w])$ , there is an  $\mathbf{x}_0 \in M([w])$  such that  $\mathbf{x}_0 \leq \mathbf{x}$ . Due to [5] each subset of  $\mathbb{N}^k$  has only a finite number of minimal elements, so the sets  $M([w])$  are finite. For  $i \in \{1, 2, \dots, k\}$ , let  $\pi_i : \mathbb{N}^k \rightarrow \mathbb{N}$  be the canonical projection on the  $i$ th factor and set

$$m_i = \max \left( \bigcup_{[w] \in L_\lambda / \sim_L} \{\pi_i(\mathbf{x}) \mid \mathbf{x} \in M([w])\} \right),$$

where  $\max(\emptyset)$  should be 0. We have  $m_i < \infty$ , for all  $i \in \{1, 2, \dots, k\}$ , because of  $|L_\lambda / \sim_L| \leq n + 1$ . Let

$$Q = \left\{ q_{[wv] \sim_L} \mid w \in L_\lambda, v \in \Sigma^* \text{ with } |v|_{a_i} \leq m_i, \text{ for all } i \in \{1, 2, \dots, k\} \right\}$$

be a set of states. The finiteness of  $L_\lambda / \sim_L$  implies that  $Q$  is also finite. Set

$$F = \left\{ q_{[w] \sim_L} \mid w \in L \right\} \subseteq Q.$$

We get  $|F| = |L / \sim_L| \leq n$ . Define the partial mapping  $R : Q \times \Sigma \rightarrow Q$  by  $R(q_{[y] \sim_L}, a) = q_{[ya] \sim_L}$ , if  $q_{[ya] \sim_L} \in Q$ , and  $R(q_{[y] \sim_L}, a)$  be undefined otherwise, for  $a \in \Sigma$  and  $y \in \Sigma^*$  with  $q_{[y] \sim_L} \in Q$ . Consider the DFA  $A = (Q, \Sigma, R, q_{[\lambda] \sim_L}, F)$ . We will show that  $L_R(A) = L$ .

First, let  $y \in L_R(A)$ . Then, there exists  $w \in L$  with  $q_{[\lambda] \sim_L} y \circ^* q_{[w] \sim_L}$ . From Lemma 2 it follows that there is a permutation  $y'$  of  $y$  with  $q_{[\lambda] \sim_L} y' \vdash^* q_{[w] \sim_L}$ . Now, the definition of  $R$  tells us  $y' \sim_L w$ . We get  $y' \in L$  and also  $y \in L$ , because  $L$  is closed under permutation. That shows  $L_R(A) \subseteq L$ .

Now, let  $y \in \Sigma^* \setminus L_R(A)$ . There are two possibilities:

1. There is a  $w \in \Sigma^* \setminus L$  with  $q_{[w] \sim_L} \in Q$  such that  $q_{[\lambda] \sim_L} y \circ^* q_{[w] \sim_L}$ . Then, there is a permutation  $y'$  of  $y$  with  $q_{[\lambda] \sim_L} y' \vdash^* q_{[w] \sim_L}$ . We get  $y' \sim_L w$ . It follows  $y' \notin L$ , which gives us  $y \notin L$ .

2. There is a  $w \in L_\lambda$ , a  $v \in \Sigma^*$  with  $|v|_{a_i} \leq m_i$ , for all  $i \in \{1, 2, \dots, k\}$ , and a  $z \in (\Sigma \setminus \Sigma_{q_{[wv] \sim_L}})^+$  such that  $q_{[\lambda] \sim_L} y \overset{\circ}{\circlearrowleft} q_{[wv] \sim_L} z$ . By Lemma 2 there is a  $y' \in \Sigma^*$  such that  $y'z$  is a permutation of  $y$  and  $q_{[\lambda] \sim_L} y' \vdash^* q_{[wv] \sim_L}$ . We get  $y' \sim_L wv$ . Set

$$U = \bigcup_{t \in \Sigma^*} \{u \in \Sigma^* \mid ut \in \text{perm}(v) \text{ and } wu \in L_\lambda\}.$$

We have  $\lambda \in U$ . Let  $u_0 \in U$  such that  $|u_0| = \max(\{|u| \mid u \in U\})$  and let  $t_0 \in \Sigma^*$  such that  $u_0 t_0 \in \text{perm}(v)$ . It follows that  $|t_0|_{a_i} \leq |v|_{a_i} \leq m_i$ , for all  $i \in \{1, 2, \dots, k\}$ , and that there exists *no*  $\mathbf{x} \in M([wu_0] \sim_L)$  with  $\mathbf{x} \leq \psi(t_0)$ . Otherwise, we would have an  $x' \in \psi^{-1}(\mathbf{x})$  which is a non-empty sub-word of  $t_0$  such that  $wu_0 x' \in L$ , which implies  $u_0 x' \in U$ . However, this is a contradiction to the maximality of  $|u_0|$ . That shows that there is no  $\mathbf{x} \in M([wu_0] \sim_L)$  with  $\mathbf{x} \leq \psi(t_0)$ . Let now  $\mathbf{x}_0 \in M([wu_0] \sim_L)$ . There exists a  $j \in \{1, 2, \dots, k\}$  with  $|t_0|_{a_j} < \pi_j(\mathbf{x}_0) \leq m_j$ . Because of  $|t_0|_{a_i} \leq m_i$ , for all  $i$  with  $i \in \{1, 2, \dots, k\}$ , and  $z \in (\Sigma \setminus \Sigma_{q_{[wv] \sim_L}})^+ = (\Sigma \setminus \Sigma_{q_{[wu_0 t_0] \sim_L}})^+$ , we get  $|z|_{a_j} = 0$ . That gives  $|t_0 z|_{a_j} < \pi_j(\mathbf{x}_0)$  and that  $\psi(t_0 z) \geq \mathbf{x}_0$  is false. So, we have shown  $\psi(t_0 z) \notin S([wu_0] \sim_L)$ , which implies  $wu_0 t_0 z \notin L$ . From  $wu_0 t_0 z \sim_L wvz \sim_L y'z \sim_L y$ , it follows that  $y \notin L$ .

We have seen  $L_R(A) = L$ . This shows that  $L$  is in **ROWJ<sub>n</sub>**.  $\square$

The previous theorem allows us to determine for a lot of interesting languages whether they belong to **ROWJ** or not.

**Corollary 4.** *Let  $L$  be a permutation closed language. Then, the language  $L$  is in **ROWJ** if and only if the Myhill-Nerode relation  $\sim_L$  has only a finite number of positive equivalence classes.*  $\square$

An application of the last corollary is the following.

**Lemma 5.** *The language  $L = \{w \in \{a, b\}^* \mid |w|_b = 0 \vee |w|_b = |w|_a\}$  is not included in **ROWJ**.*

*Proof.* The language  $L$  is closed under permutation. For  $\sim_L$ , the positive equivalence classes  $[a^0], [a^1], \dots$  are pairwise different, since  $a^n b^m \in L$  if and only if  $m \in \{0, n\}$ . Corollary 4 tells us that  $L$  is not in **ROWJ**.  $\square$

There are counterexamples for both implications of Corollary 4, if we do not assume that the language  $L$  is closed under permutation. For instance, set  $L = \{a^n b^n \mid n \geq 0\}$ , which was shown to be not in **ROWJ** in [3]. Then, the positive equivalence classes of  $\sim_L$  are  $[\lambda]$  and  $[ab]$ . On the other hand, we have:

**Lemma 6.** *There is a language  $L$  in **ROWJ** such that  $\sim_L$  has an infinite number of positive equivalence classes.*  $\square$

From Corollary 4 we conclude the following equivalence.

**Corollary 7.** *Let  $L$  be a permutation closed **ROWJ** language over the alphabet  $\Sigma$ . Then, the language  $L$  is regular if and only if  $\Sigma^* \setminus L$  is in **ROWJ**.  $\square$*

The previous corollary gives us:

**Lemma 8.** *The language  $\{w \in \{a, b\}^* \mid |w|_a \neq |w|_b\}$  is not in **ROWJ**.  $\square$*

Having the statement of Theorem 3, it is natural to ask, which numbers arise as the number of positive equivalence classes of the Myhill-Nerode relation  $\sim_L$  of a permutation closed language  $L$ . The answer is, that all natural numbers arise this way, even if we restrict ourselves to some special families:

**Theorem 9.** *For each  $n > 0$ , there is a permutation closed language which is (1) finite, (2) regular, but infinite, (3) context-free, but non-regular, (4) non-context-free such that the corresponding Myhill-Nerode relation has exactly  $n$  positive equivalence classes.  $\square$*

The previous theorem, together with Theorem 3, implies that the language families **ROWJ<sub>n</sub>** form a proper hierarchy, even if we only consider languages out of special language families:

**Corollary 10.** *For all  $n \geq 0$ , we have  $\mathbf{ROWJ}_n \cap \mathbf{FAM} \subset \mathbf{ROWJ}_{n+1} \cap \mathbf{FAM}$ , where **FAM** is either  $2^{\Sigma^*}$ , **FIN**,  $\mathbf{REG} \setminus \mathbf{FIN}$ ,  $\mathbf{CF} \setminus \mathbf{REG}$ , or  $\mathbf{CS} \setminus \mathbf{CF}$ . The statement remains valid if restricted to permutation closed languages.  $\square$*

## 4 Inclusion Relations Between Language Families

We investigate inclusion relations between **ROWJ** and other important languages families. The following relations were given in [3]: (1)  $\mathbf{REG} \subset \mathbf{ROWJ}$ , (2) **ROWJ** and **CF** are incomparable, and (3)  $\mathbf{ROWJ} \not\subset \mathbf{JFA}$ . It was stated as an open problem if  $\mathbf{JFA} \subset \mathbf{ROWJ}$ . We can answer this using Lemma 5:

**Theorem 11.** *The language families **ROWJ** and **JFA** are incomparable.  $\square$*

For the complexity of **ROWJ**, we get that the language family **ROWJ** is included in both of the complexity classes  $\mathbf{DTIME}(n^2)$  and  $\mathbf{DSPACE}(n)$ . This implies that **ROWJ** is properly included in **CS**. Moreover, we find the following relations:

**Theorem 12.** *We have (1)  $\mathbf{ROWJ} \subset \mathbf{CS}$ , (2) **ROWJ** and **DCF** are incomparable, and (3) every language in **ROWJ** is semilinear.  $\square$*

For permutation closed language families the next theorem applies.

**Theorem 13.** *We have  $\mathbf{pFIN} \subset \mathbf{pREG} \subset \mathbf{pDCF} \subset \mathbf{pCF} \subset \mathbf{pJFA} = \mathbf{JFA} \subset \mathbf{pCS}$  and  $\mathbf{pREG} \subset \mathbf{pROWJ} \subset \mathbf{JFA}$ . Furthermore, the family **pROWJ** is incomparable to **pDCF** and to **pCF**. We have  $\mathbf{pROWJ} \subset \mathbf{ROWJ}$ .  $\square$*



### 5 Closure Properties of ROWJ and pROWJ

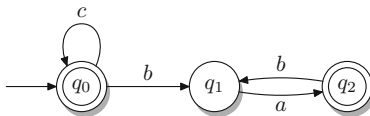
We consider closure properties of the language families **ROWJ** and **pROWJ**. Our results are summarized in Table 1. Here we only show that **ROWJ** is *not* closed under inverse homomorphism, while the permutation closed language family **pROWJ** is closed under this operation. The proofs of the remaining closure and non-closure results will be given in the journal version of the paper.

**Table 1.** Closure properties of **ROWJ** and **pROWJ**. The gray shaded results are proven in this paper. The non-shaded closure properties for **REG** are folklore. For **ROWJ** the closure/non-closure results can be found in [3] and that for the language family **JFA** in [1,6,7,12].

Closed under	Language family			
	REG	pROWJ	ROWJ	JFA
Union	yes	no	no	yes
Union with reg. lang.	yes	no	no	no
Intersection	yes	yes	no	yes
Intersection with reg. lang.	yes	no	no	no
Complementation	yes	no	no	yes
Reversal	yes	yes	no	yes
Concatenation	yes	no	no	no
Right conc. with reg. lang.	yes	no	no	no
Left conc. with reg. lang.	yes	no	no	no
Left conc. with prefix-free reg. lang.	yes	no	yes	no
Kleene star or plus	yes	no	no	no
Homomorphism	yes	no	no	no
Inv. homomorphism	yes	yes	no	yes
Substitution	yes	no	no	no
Permutation	no	yes	no	yes

**Theorem 14.** *The family ROWJ is not closed under inverse homomorphism.*

*Proof.* Let  $A$  be the ROWJFA  $A = (\{q_0, q_1, q_2\}, \{a, b, c\}, R, q_0, \{q_0, q_2\})$ , where  $R$  consists of the rules  $q_0c \rightarrow q_0$ ,  $q_0b \rightarrow q_1$ ,  $q_1a \rightarrow q_2$ , and  $q_2b \rightarrow q_1$ . The



**Fig. 2.** The ROWJFA  $A$  satisfying  $L(A) \cap \{ac, b\}^* = \{(ac)^n b^n \mid n \geq 0\}$ .

ROWJFA  $A$  is depicted in Fig. 2. Let  $h : \{a, b\}^* \rightarrow \{a, b, c\}^*$  be the homomorphism, given by  $h(a) = ac$  and  $h(b) = b$ . We have  $h(\{a, b\}^*) = \{ac, b\}^*$ .

Let now  $\lambda \neq w \in L(A) \cap \{ac, b\}^*$ , which implies  $|w|_b > 0$ . When  $A$  reads  $w$ , it reaches the first occurrence of the symbol  $b$  in state  $q_0$ . After reading this  $b$ , the automaton is in state  $q_1$ . Now, no more  $c$  can be read. So, we get  $w \in (ac)^+ b^+$ . Whenever  $A$  is in state  $q_2$ , it has read the same number of  $a$ 's and  $b$ 's. This gives us  $w \in \{(ac)^n b^n \mid n > 0\}$ . That shows  $L(A) \cap \{ac, b\}^* \subseteq \{(ac)^n b^n \mid n \geq 0\}$ .

On the other hand, for  $n > 0$ , we have

$$q_0(ac)^n b^n \circlearrowleft^n q_0 b^n a^n \circlearrowleft^2 q_2 a^{n-1} b^{n-1} \circlearrowleft^2 q_2 a^{n-2} b^{n-2} \circlearrowleft^2 \dots \circlearrowleft^2 q_2 ab \circlearrowleft^2 q_2.$$

This implies  $L(A) \cap \{ac, b\}^* = \{(ac)^n b^n \mid n \geq 0\}$ . We get

$$\begin{aligned} h^{-1}(L(A)) &= h^{-1}(L(A) \cap h(\{a, b\}^*)) = h^{-1}(L(A) \cap \{ac, b\}^*) \\ &= h^{-1}(\{(ac)^n b^n \mid n \geq 0\}) = \{a^n b^n \mid n \geq 0\}. \end{aligned}$$

In [3] it was shown that this language is not in **ROWJ**.  $\square$

For the language family **pROWJ** the situation w.r.t. the closure under inverse homomorphisms is exactly the other way around.

**Theorem 15.** *Let  $\Gamma$  and  $\Sigma$  be alphabets and  $h : \Gamma^* \rightarrow \Sigma^*$  be a homomorphism. Furthermore let  $L \subseteq \Sigma^*$  be in **pROWJ<sub>n</sub>**, for some  $n \geq 0$ . Then, the language  $h^{-1}(L)$  is also in **pROWJ<sub>n</sub>**.*

*Proof.* It is not difficult to see that the family of permutation closed languages is closed under inverse homomorphism. So, the language  $h^{-1}(L)$  is closed under permutation. Theorem 3 gives us  $|L/\sim_L| \leq n$ . From  $L = \bigcup_{S \in L/\sim_L} S$ , we get  $h^{-1}(L) = \bigcup_{S \in L/\sim_L} h^{-1}(S)$ . Consider now an element  $S \in L/\sim_L$ , two words  $v, w \in h^{-1}(S)$ , and an arbitrary  $u \in \Gamma^*$ . Because of  $h(v), h(w) \in S$ , we have  $h(v) \sim_L h(w)$ . It follows that

$$vu \in h^{-1}(L) \Leftrightarrow h(v)h(u) \in L \Leftrightarrow h(w)h(u) \in L \Leftrightarrow wu \in h^{-1}(L).$$

We have shown  $v \sim_{h^{-1}(L)} w$ . So, we get  $|h^{-1}(L)/\sim_{h^{-1}(L)}| \leq |L/\sim_L| \leq n$ , which by Theorem 3 implies that  $h^{-1}(L)$  is in **pROWJ<sub>n</sub>**.  $\square$

Thus we immediately get:

**Corollary 16.** *The family **pROWJ** is closed under inverse homomorphism.  $\square$*

## 6 More on Languages Accepted by ROWJFAs

In Corollary 4 a characterization of the permutation closed languages that are in **ROWJ** was given. In this section, we characterize languages in **ROWJ** for some cases where the considered language does not need to be permutation closed.

**Theorem 17.** *For an alphabet  $\Sigma$ , let  $w \in \Sigma^*$  and  $L \subseteq \Sigma^*$ . Then, the language  $wL$  is in **ROWJ** if and only if  $L$  is in **ROWJ**.*

*Proof.* If  $L$  is in **ROWJ**, then  $wL$  is also in **ROWJ**, because the language family **ROWJ** is closed under concatenation with prefix-free languages from the left. Now assume that  $wL$  is in **ROWJ** and  $L \neq \emptyset$ . We may also assume that  $|w| = 1$ . The general case follows from this special case *via* a trivial induction over the length of  $w$ . Thus, let  $w = a$  for an  $a \in \Sigma$  and let  $A = (Q, \Sigma, R, s, F)$  be a DFA with  $L_R(A) = aL$ . In the following, we will show *via* a contradiction that the value  $R(s, a)$  is defined. Assume that  $R(s, a)$  is undefined and let  $v$  be an arbitrary word out of  $L$ . Because  $av \in L_R(A)$ , there is a symbol  $b \in \Sigma_s$ , two words  $x \in (\Sigma \setminus \Sigma_s)^*$  and  $y \in \Sigma^*$ , and a state  $p \in F$  such that  $v = xby$  and  $saxby \circlearrowleft R(s, b)ya \circlearrowleft^* p$ . This gives us  $sbyax \vdash R(s, b)ya \circlearrowleft^* p$ , which implies  $byax \in L_R(A) = aL$ . However, this is a contradiction, because  $b \neq a$ . So, the value  $R(s, a)$  is defined.

Consider the DFA  $B = (Q, \Sigma, R, R(s, a), F)$ . For a word  $z \in \Sigma^*$ , we have  $z \in L_R(B)$  if and only if  $az \in L_R(A) = aL$ , because of  $saz \vdash R(s, a)z$ . That gives us  $L_R(B) = L$  and we have shown that  $L$  is in **ROWJ**.  $\square$

From the previous theorem and Corollary 4 we get:

**Corollary 18.** *For an alphabet  $\Sigma$ , let  $w \in \Sigma^*$  and let  $L \subseteq \Sigma^*$  be a permutation closed language. Then, the set  $wL$  is in **ROWJ** if and only if the Myhill-Nerode relation  $\sim_L$  has only a finite number of positive equivalence classes.*  $\square$

Next, we will give a characterization for the concatenation  $Lw$  of a language  $L$  and a word  $w$ . To do so, we need the following lemma. It treats the case of an ROWJFA that is only allowed to jump over one of the input symbols.

**Lemma 19.** *Let  $A = (Q, \Sigma, R, s, F)$  be a DFA with a symbol  $a \in \Sigma$  such that  $R(q, b)$  is defined for all  $(q, b) \in Q \times (\Sigma \setminus \{a\})$ . Then,  $L_R(A)$  is regular.*  $\square$

Our characterization for languages of the form  $Lw$  generalizes a result from [3], which says that the language  $\{va \mid v \in \{a, b\}^*, |v|_a = |v|_b\}$  is not in **ROWJ**:

**Theorem 20.** *For an alphabet  $\Sigma$ , let  $w \in \Sigma^*$  be a non-empty word and  $L \subseteq \Sigma^*$ . Then, the language  $Lw$  is in **ROWJ** if and only if  $L$  is regular.*  $\square$

Now, we consider the case of two languages over disjoint alphabets.

**Theorem 21.** *For disjoint alphabets  $\Sigma_1$  and  $\Sigma_2$ , let  $L_1 \subseteq \Sigma_1^*$  and  $L_2 \subseteq \Sigma_2^*$  with  $L_1 \neq \emptyset \neq L_2 \neq \{\lambda\}$  such that  $L_1L_2$  is in **ROWJ**. Then, the language  $L_1$  is regular and  $L_2$  is in **ROWJ**.*  $\square$

Adding prefix-freeness for  $L_1$ , we get an equivalence, by Theorem 21 and the closure of **ROWJ** under left-concatenation with prefix-free regular sets.

**Corollary 22.** *For disjoint alphabets  $\Sigma_1$  and  $\Sigma_2$ , let  $L_1 \subseteq \Sigma_1^*$  be a prefix-free language and  $L_2 \subseteq \Sigma_2^*$  be an arbitrary language with  $L_1 \neq \emptyset \neq L_2 \neq \{\lambda\}$ . Then, the language  $L_1L_2$  is in **ROWJ** if and only if  $L_1$  is regular and  $L_2$  is in **ROWJ**.  $\square$*

The previous corollary directly implies the following characterization:

**Corollary 23.** *For disjoint alphabets  $\Sigma_1$  and  $\Sigma_2$ , let  $L_1 \subseteq \Sigma_1^*$  be a prefix-free language and  $L_2 \subseteq \Sigma_2^*$  be a permutation closed language with  $L_1 \neq \emptyset \neq L_2 \neq \{\lambda\}$ . Then, the language  $L_1L_2$  is in **ROWJ** if and only if  $L_1$  is regular and the Myhill-Nerode relation  $\sim_{L_2}$  has only a finite number of positive equivalence classes.  $\square$*

If a non-empty language and a non-empty permutation closed language over disjoint alphabets are separated by a symbol, we get the following result:

**Corollary 24.** *For disjoint alphabets  $\Sigma_1$  and  $\Sigma_2$ , let  $L_1 \subseteq \Sigma_1^*$  be a non-empty language and  $L_2 \subseteq \Sigma_2^*$  be a non-empty permutation closed language. Furthermore, let  $a \in \Sigma_2$ . Then, the language  $L_1aL_2$  is in **ROWJ** if and only if  $L_1$  is regular and the Myhill-Nerode relation  $\sim_{L_2}$  has only a finite number of positive equivalence classes.  $\square$*

For an alphabet  $\Sigma = \{a_1, a_2, \dots, a_n\}$ , the family of subsets of  $a_1^*a_2^* \dots a_n^*$  is kind of a counterpart of the family of permutation closed languages over  $\Sigma$ . In a language  $L \subseteq a_1^*a_2^* \dots a_n^*$ , for each word  $w \in L$ , no other permutation of  $w$  is in  $L$ . We can characterize the subsets of  $a_1^*a_2^* \dots a_n^*$  that are in **ROWJ**.

**Theorem 25.** *Let  $\{a_1, a_2, \dots, a_n\}$  be an alphabet and  $L \subseteq a_1^*a_2^* \dots a_n^*$ . Then, the language  $L$  is in **ROWJ** if and only if  $L$  is regular.  $\square$*

## References

1. Beier, S., Holzer, M., Kutrib, M.: Operational state complexity and decidability of jumping finite automata. In: Charlier, É., Leroy, J., Rigo, M. (eds.) DLT 2017. LNCS, vol. 10396, pp. 96–108. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-62809-7\\_6](https://doi.org/10.1007/978-3-319-62809-7_6)
2. Bensch, S., Bordihn, H., Holzer, M., Kutrib, M.: On input-revolving deterministic and nondeterministic finite automata. Inform. Comput. **207**(11), 1140–1155 (2009)
3. Chigahara, H., Fazekas, S., Yamamura, A.: One-way jumping finite automata. Int. J. Found. Comput. Sci. **27**(3), 391–405 (2016)
4. Culy, C.: Formal properties of natural language and linguistic theories. Linguist. Philos. **19**, 599–617 (1996)
5. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with  $n$  distinct prime factors. Am. J. Math. **35**(4), 413–422 (1913)
6. Fernau, H., Paramasivan, M., Schmid, M.L.: Jumping finite automata: characterizations and complexity. In: Drewes, F. (ed.) CIAA 2015. LNCS, vol. 9223, pp. 89–101. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-22360-5\\_8](https://doi.org/10.1007/978-3-319-22360-5_8)

7. Fernau, H., Paramasivan, M., Schmid, M.L., Vorel, V.: Characterization and complexity results on jumping finite automata (2015). <http://arxiv.org/abs/1512.00482>
8. Ginsburg, S., Spanier, E.H.: Bounded ALGOL-like languages. *Trans. AMS* **113**, 333–368 (1964)
9. Greibach, S.A.: An infinite hierarchy of context-free languages. *J. ACM* **16**(1), 91–106 (1969)
10. Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading (1979)
11. Meduna, A., Zemek, P.: Jumping finite automata. *Int. J. Found. Comput. Sci.* **23**(7), 1555–1578 (2012)
12. Meduna, A., Zemek, P.: Chapter 17 Jumping finite automata. In: *Regulated Grammars and Automata*, pp. 567–585. Springer, New York (2014). [https://doi.org/10.1007/978-1-4939-0369-6\\_17](https://doi.org/10.1007/978-1-4939-0369-6_17)
13. Vorel, V.: Basic properties of jumping finite automata (2015). <http://arxiv.org/abs/1511.08396v2>



# Word Problem Languages for Free Inverse Monoids

Tara Brough<sup>(✉)</sup>

Centro de Matemática e Aplicações, Faculdade de Ciências e Tecnologia,  
Universidade Nova de Lisboa, 2829–516 Caparica, Portugal  
t.brough@fct.unl.pt

**Abstract.** This paper considers the word problem for free inverse monoids of finite rank from a language theory perspective. It is shown that no free inverse monoid has context-free word problem; that the word problem of the free inverse monoid of rank 1 is both 2-context-free (an intersection of two context-free languages) and ETOL; that the co-word problem of the free inverse monoid of rank 1 is context-free; and that the word problem of a free inverse monoid of rank greater than 1 is not poly-context-free.

**Keywords:** Word problems · Co-word problems · Inverse monoids  
ETOL languages · Stack automata · Poly-context-free languages

## 1 Introduction

The word problem of a finitely generated semigroup is, informally, the problem of deciding whether two words over a given finite generating set represent the same element of the semigroup. Although it is undecidable [23], even for finitely presented groups [2, 21], there has been much study (especially for groups) of word problems that are in some sense ‘easily’ decidable, for example by having low space or time complexity, or being in certain low-complexity language classes.

For groups, the obvious formalisation of the word problem is as the set of all words over the set of generators and their inverses representing the identity element, since two words  $u$  and  $v$  represent the same element iff  $uv^{-1}$  represents the identity. This has been generalised to semigroups in two ways: the first, which we call the *word problem* of a semigroup  $S$  with respect to finite generating set  $A$  is the set  $WP(S, A) = \{u\#v^{\text{rev}} \mid u =_S v, u, v \in A^+\}$  (where  $\#$  is a symbol not in  $A$  and  $v^{\text{rev}}$  denotes the reverse of  $v$ ); the second, the *two-tape word problem* of

---

T. Brough—The author was supported by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through an FCT post-doctoral fellowship (SFRH/BPD/121469/2016) and the projects UID/Multi/04621/2013 (CEMAT-CIÊNCIAS) and UID/MAT/00297/2013 (Centro de Matemática e Aplicações).

$S$  with respect to  $A$ , is the relation  $\iota(S, A) = \{(u, v) \in A^+ \times A^+ \mid u =_S v\}$ . Monoid versions of these are obtained by replacing  $A^+$  with  $A^*$ . The word problem has been studied in [7, 12, 13] and the two-tape word problem in [4, 22].

A semigroup  $S$  is *inverse* if for every  $x \in S$  there is a *unique*  $y \in S$  such that  $xyx = x$  and  $xyx = y$ . The classes of inverse semigroups and inverse monoids each form varieties of algebras and hence contain free objects. The free inverse monoid on a set  $X$  is denoted  $\text{FIM}(X)$ ; if  $|X| = k$  then we also use the notation  $\text{FIM}_k$ , and  $k$  is called the *rank* of  $\text{FIM}_k$ . All results in this paper are stated for free inverse monoids, but are equally true for free inverse semigroups, since in a free inverse monoid the only representative of the identity is the empty word  $\epsilon$ .

Word problems of free inverse monoids have already been studied from a time and space complexity perspective: they are recognisable in linear time and in logarithmic space [18]. The aim of this paper is to understand these word problems from a language-theoretic perspective. All free inverse monoid word problems are context-sensitive, since this is equivalent to recognisability in linear space. Our main goal is thus to determine in which subclasses of the context-sensitive languages the free inverse monoid word problem lies. Before summarising the results, we introduce several of the language classes considered. All classes mentioned here are closed under inverse generalised sequential machine mappings, and hence the property of having word problem in any of these classes is closed under change of finite generating set.

The non-closure of the class  $\mathcal{CF}$  of context-free languages under complementation and intersection [15] leads naturally to the definition of the classes of *coCF* and *poly-CF* languages, being respectively the classes of complements and finite intersections of context-free languages. A language is called *k-CF* if it is an intersection of  $k$  context-free languages. Groups with *coCF* word problem were studied in [14], and groups with poly-context-free word problem in [3]. For groups, having *coCF* word problem is equivalent to the *co-word problem* (the complement of the word problem, or abstractly the problem of deciding whether two words represent *different* elements) being context-free. For monoids, we generalise this terminology on the abstract rather than technical level: the complement of the word problem is not an algebraically interesting language, so we define  $\text{coWP}(M, X) = \{u\#v^{\text{rev}} \mid u =_M v, u, v \in X^*\}$ . The two-tape co-word problem is the complement of the two-tape word problem.

*Stack automata*, introduced in [10], are a generalisation of pushdown automata that allow the contents of the stack to be examined in ‘read-only’ mode. They are a special case of the *nested stack automata* introduced slightly later by Aho [1] to recognise indexed languages. The *checking stack languages* are recognised by the more restricted *checking stack automata* [11], in which the stack contents can only be altered prior to commencing reading of the input.

*ETOL languages* are another subclass of indexed languages, standardly defined by *ETOL-systems*, which are essentially finite collections of ‘tables’ of context-free-grammar-type productions. These operate similarly to context-free grammars except that at each step in a derivation, productions all from the same

‘table’ must be applied to *every* nonterminal in the current string (each table is required to have productions from every nonterminal, though these of course may be trivial). The more restricted *EDTOL languages* have the further requirement that in each table of productions there be only one production from each nonterminal. An automaton model for ETOL languages was given in [25]: it consists of a checking stack with attached push-down stack, operating in such a way that the pointers of the two stacks move together. See [24] (especially Chapter V) for further information on ETOL languages and their many relatives.

In the rank 1 case our goal is achieved fairly comprehensively, with both types of word problem for  $\text{FIM}_1$  being shown to be  $2\text{-}\mathcal{CF}$  (but not context-free),  $\text{co-}\mathcal{CF}$  and a checking stack language (and hence ETOL). As far as the author is aware, this is the first known example of a semigroup with ETOL but not context-free word problem. This result is particularly interesting because of the long-standing open problem of whether the indexed languages – of which the ETOL languages form a subclass – give any additional power over context-free languages for recognising word problems of groups [9, 19]. In higher ranks we show that  $\text{WP}(\text{FIM}_k)$  for  $k \geq 2$  is not  $\text{poly-}\mathcal{CF}$ . We conjecture that the same is true for  $\iota(\text{FIM}_k)$ , and that neither version of the word problem is  $\text{co}\mathcal{CF}$  or indexed except in rank 1.

## 2 Background

### 2.1 Free Inverse Monoids

Recall that a monoid  $M$  is *inverse* if for every  $x \in M$  there is a unique  $y \in M$  such that  $xyx = x$  and  $xyy = y$ . The element  $y$  is called the *inverse* of  $x$  and is usually denoted  $x^{-1}$ . In this paper we will also often use the notation  $\bar{x}$  for the inverse of  $x$ . Given a set  $X$ , we use the notation  $X^{-1}$  for a set  $\{\bar{x} \mid x \in X\}$  of formal inverses for  $X$ , and  $X^\pm$  for  $X \cup X^{-1}$ . For an element  $x \in X^\pm$ , if  $x \in X$  then  $x^{-1} = \bar{x}$ , while if  $x = \bar{y}$  for  $y \in X$  then  $x^{-1} = y$ . We can extend this to define the inverse of a word  $w = w_1 \dots w_n$  with  $w_i \in X^\pm$  by  $w^{\text{inv}} = w_n^{-1} \dots w_1^{-1}$ .

For any set  $X$ , the *free inverse monoid*  $\text{FIM}(X)$  on  $X$  exists and is given by the monoid presentation

$$\text{FIM}(X) = \langle X^\pm \mid u = uu^{\text{inv}}u, uu^{\text{inv}}vv^{\text{inv}} = vv^{\text{inv}}uu^{\text{inv}} \ (u, v \in (X^\pm)^*) \rangle.$$

This presentation is not very useful for working with the word problem of free inverse monoids. A more powerful tool is given by *Munn trees* [20]: certain labelled directed finite trees that stand in one-to-one correspondence with the elements of  $\text{FIM}(X)$ , such that the product of two elements can easily be computed using their corresponding trees. To obtain the Munn tree for an element  $m \in \text{FIM}(X)$ , we use the Cayley graph  $\mathcal{G}(F(X), X)$  of the free group  $F(X)$ . This is a labelled directed tree with  $|X|$  edges labelled by the elements of  $X$  entering and leaving each vertex. (The Cayley graph also has its vertices labelled by the elements of  $G$ , but these are not needed for our purposes.) Given any word  $w \in (X^\pm)^*$  representing  $m$ , choose any vertex of  $\mathcal{G}(F(X), X)$  as the start vertex



and label it  $\alpha$ . From vertex  $\alpha$ , then trace out the path defined by reading  $w$  from left to right, where for  $x \in X$ , follow the edge labelled  $x$  leading *from* the current vertex upon reading  $x$ , and follow the edge labelled  $\bar{x}$  leading *to* the current vertex upon reading  $\bar{x}$ . Mark the final vertex of the path thus traced as  $\omega$ , and remove all edges not traversed during reading of  $w$ . The result is the *Munn tree* of  $w$ , and the free inverse monoid relations ensure that two words produce the same Munn tree iff they represent the same element of  $\text{FIM}(X)$ .

To multiply two Munn trees, simply attach the start vertex of the second tree to the end vertex of the first tree, and identify any edges with the same label and direction issuing from or entering the same vertex. From this it can be seen that the *idempotents* (elements  $x$  such that  $x^2 = x$ ) in  $\text{FIM}(X)$  are those elements whose Munn trees have  $\alpha = \omega$ , and that these elements commute.

For a more detailed discussion, with diagrams, see [17, Section 6.4].

## 2.2 Word Problems of Inverse Monoids

Two notions of word problem for inverse monoids will occur throughout this paper. For an inverse monoid  $M$  with finite generating set  $X$ , the *word problem* of  $M$  with respect to  $X$  is the set

$$\text{WP}(M, X) = \{u\#v^{\text{inv}} \mid u, v \in (X^\pm)^*, u =_M v\},$$

while the *two-tape word problem* of  $M$  with respect to  $X$  is

$$\iota(M, X) = \{(u, v) \in (X^\pm)^* \times (X^\pm)^* \mid u =_M v\}.$$

If the generating set  $X$  is irrelevant, we may use the notation  $\text{WP}(M)$  or  $\iota(M)$ .

Each of these notions generalises the definition of the group word problem  $W(G, X)$  as the set of all words over  $X^\pm$  representing the identity. If  $M$  is a group, then  $W(G, X)$  and  $\text{WP}(G, X)$  are obtained from each other by very simple operations (deletion or insertion of a single  $\#$ ), and so membership in any ‘reasonable’ language class will not depend on whether we consider the group or inverse monoid word problem. For the two-tape word problem the generalisation is of a more algebraic nature:  $\iota(M, X)$  and  $W(G, X)$  are each the lift to  $(X^\pm)^*$  of the natural homomorphism from the free inverse monoid (respectively free group) on  $X$  to  $M$  (respectively  $G$ ). The kernel of a group homomorphism is a set, while the kernel of a semigroup homomorphism is a relation.

The word problem for semigroups in general has been studied in [12], where it is defined as the set of words  $u\#v^{\text{rev}}$  with  $u$  and  $v$  representing the same element. For inverse monoids, this is equivalent to the word problem considered here, since  $u\#v^{\text{inv}}$  is obtained from  $u\#v^{\text{rev}}$  by simply replacing every symbol after the  $\#$  by its inverse. This operation can be viewed as an inverse generalised sequential machine mapping, and thus all classes of languages we consider are closed under it (and hence all results in this paper hold for the definition in [12] as well). Note that it is still essential to include the ‘dividing symbol’  $\#$ : as an example, if  $F = \text{FIM}(X)$  and  $x \in X$ , then  $x\#\bar{x} \in \text{WP}(F, X)$ , but  $x\bar{x}\# \notin \text{WP}(F, X)$ .

### 3 The Rank 1 Case

Since the free group of rank 1 is isomorphic to  $(\mathbb{Z}, +)$ , Munn trees in the rank 1 case can be viewed as intervals of integers containing zero (the starting point  $\alpha$ ), with a marked point  $(\omega)$ . This allows elements of  $\text{FIM}_1$  to be represented by a 3-tuple of integers  $(-l, n, m)$  with  $l, n \in \mathbb{N}_0$  and  $-l \leq m \leq n$ , where  $[-l, n]$  is the interval spanned by the Munn tree and  $m$  is the marked point. Multiplication in this representation of  $\text{FIM}_1$  is given by

$$(-l, n, m)(-l', n', m') = (\min\{-l, m - l'\}, \max\{n, m + n'\}, m + m').$$

Equipped with this model of  $\text{FIM}_1$ , we can determine that free inverse monoids never have context-free word problem.

**Theorem 1.** *For any  $k \in \mathbb{N}$ , neither  $\text{WP}(\text{FIM}_k)$  nor  $\iota(\text{FIM}_k)$  is context-free.*

*Proof.* Suppose that  $\text{WP}(\text{FIM}_k, X)$  is context-free ( $X$  any finite generating set of  $\text{FIM}_k$ ). Then for any  $x \in X$ , the language  $L := \text{WP}(\text{FIM}_k, X) \cap x^* \bar{x}^* x^* \# \bar{x}^*$  is also context-free. For  $n \in \mathbb{N}$ , let  $w_n = x^n \bar{x}^n x^n \# \bar{x}^n$ , which is in  $L$  for all  $n \in \mathbb{N}$ . For  $n$  greater than the pumping length  $p$  of  $L$ , we can express  $w_n$  in the form  $uvwyz$  such that  $|vy| \geq 1$ ,  $|vwy| \leq p$ , and the strings  $v, y$  can simultaneously be ‘pumped’. Thus there must exist  $i, j \in \mathbb{N}_0$ , not both zero, such that all strings of one of the following three forms must be in  $L$  for  $m \geq -1$ :

$$x^{n+im} \bar{x}^{n+jm} x^n \# \bar{x}^n, \quad x^n \bar{x}^{n+im} x^{n+jm} \# \bar{x}^n, \quad x^n \bar{x}^n x^{n+im} \# \bar{x}^{n+jm}.$$

However, in all cases, some words of the given form are not in  $L$ :

Word form	Not in $L$ for
$x^{n+im} \bar{x}^{n+jm} x^n \# \bar{x}^n$	$(i \neq 0 \wedge m \geq 1) \vee (i = 0 \wedge j \neq 0 \wedge m \geq 1)$
$x^n \bar{x}^{n+im} x^{n+jm} \# \bar{x}^n$	as above
$x^n \bar{x}^n x^{n+im} \# \bar{x}^{n+jm}$	$(j \neq 0 \wedge m = -1) \vee (j = 0 \wedge i \neq 0 \wedge m \geq 1)$

Thus  $L$ , and hence  $\text{WP}(\text{FIM}_k, X)$ , is not context-free. The proof for  $\iota(\text{FIM}_k, X)$  is similar, by intersecting with  $(x^* \bar{x}^* x^*, \{x, \bar{x}\}^*)$  and using the pumping lemma on  $(x^n \bar{x}^n x^n, x^n)$  for sufficiently large  $n$ .  $\square$

For the remainder of this section, let  $\text{FIM}_1$  be generated by  $X = \{x\}$  and let  $Y = X^\pm = \{x, \bar{x}\}$ . For  $w \in Y^*$ , denote the image of  $w$  in  $\text{FIM}_1$  by  $\hat{w}$ . We define functions  $\lambda, \nu$  and  $\mu$  from  $Y^*$  to  $\mathbb{Z}$  by setting  $(-\lambda(w), \nu(w), \mu(w)) = \hat{w}$ . It will often be helpful to regard words in  $Y^*$  as paths in the integers starting at 0, with  $x$  representing a step in the positive direction and  $\bar{x}$  a step in the negative direction. We will refer to and visualise these directions as right (positive) and left (negative). Thus for  $w \in Y^*$  the path traced out by  $w$  has rightmost point  $\nu(w)$ , leftmost point  $-\lambda(w)$  and endpoint  $\mu(w)$ .

The idempotents in  $\text{FIM}_1$  are the elements  $(-l, n, 0)$  for  $l, n \in \mathbb{N}_0$ . We define the set of *positive idempotents*  $E^+ = \{(0, n, 0) \mid n \in \mathbb{N}_0\}$  and similarly the set of *negative idempotents*  $E^- = \{(-l, 0, 0) \mid l \in \mathbb{N}_0\}$  in  $\text{FIM}_1$ . (Note that in these

definitions, the identity  $(0, 0, 0)$  is counted as both a positive and a negative idempotent.) Grammars for the sets of positive and negative idempotents form an important building block in Theorem 2 (as well as in the ETOL grammar mentioned following Corollary 1).

**Lemma 1.** *Let  $Y = \{x, \bar{x}\}$  and  $L_{E^+} = \{w \in Y^* \mid \hat{w} \in E^+\}$ . Then  $L_{E^+}$  is generated by the context-free grammar  $\Gamma^+ = (\{S\}, Y, P^+, S)$  with  $P^+$  consisting of productions  $P_1 : S \rightarrow SS$ ,  $P_2 : S \rightarrow xS\bar{x}$  and  $P_3 : S \rightarrow \varepsilon$ . Similarly,  $L_{E^-} := \{w \in Y^* \mid \hat{w} \in E^-\}$  is generated by the context-free grammar  $\Gamma^- = (\{S\}, Y, P^-, S)$  with  $P^-$  the same as  $P^+$  except that  $P_2$  is replaced by  $P'_2 : S \rightarrow \bar{x}Sx$ .*

*Proof.* We can view  $L_{E^+}$  as the language of all paths starting and ending at 0 and never crossing to the left of 0. Concatenating two such paths gives another such path, so we have  $(L_{E^+})^* = L_{E^+}$ . Let  $L$  be the language of all paths in  $Y^*$  that start and end at 0 without visiting 0 in between. Then  $L_{E^+} = L^*$  and  $w \in Y^*$  is in  $L$  if and only if either  $w = \varepsilon$  or there exists  $v \in L_{E^+}$  such that  $w = xv\bar{x}$ . That is,  $L_{E^+} = (xL_{E^+}\bar{x})^*$ .

Let  $M$  be the language generated by  $\Gamma^+$ . We show by induction on the length of words that  $L_{E^+} = M$ . Note that for any  $w$  in  $L_{E^+}$  or  $M$  we have  $|w|_x = |w|_{\bar{x}}$ , so both languages consist of words of even length. To begin with,  $L_{E^+} \cap Y^0 = M \cap Y^0 = \{\varepsilon\}$ . Now suppose that  $L_{E^+} \cap Y^{2i} = M \cap Y^{2i}$  for all  $i < n$ . For  $w \in Y^{2n}$ , we have  $w \in L_{E^+}$  if and only if either  $w = w_1w_2$  or  $w = xv\bar{x}$  for some  $w_1, w_2, v \in L_{E^+}$ . By induction, this occurs iff  $w_1, w_2 \in M$  respectively  $v \in M$ , iff  $S \rightarrow SS \Rightarrow w_1w_2$  respectively  $S \rightarrow xS\bar{x} \Rightarrow xv\bar{x}$  in  $\Gamma^+$ , iff  $w \in M$ . Hence  $L_{E^+} = M$ . The language  $L_{E^-}$  is the reverse of  $L_{E^+}$ , and the grammars  $\Gamma^+$  and  $\Gamma^-$  are the reverse of one another, hence  $L_{E^-}$  is generated by  $\Gamma^-$ .  $\square$

A word  $u\#v^{\text{inv}}$  with  $u, v \in Y^*$  is in  $\text{WP}(\text{FIM}_1, X)$  if and only if it traces out a path starting and ending at 0 which reaches its rightmost and leftmost points each at least once before and at least once after the  $\#$ . If the minimum or maximum is achieved at the end of  $u$ , this counts as being achieved both before and after  $\#$ . (The path must end at 0 because if  $\hat{u} = \hat{v}$  then  $\hat{u}(\hat{v})^{-1}$  is an idempotent.) We now show that although the word problem of  $\text{FIM}_1$  is not context-free, it can be expressed as an intersection of two context-free languages.

**Theorem 2.**  *$\text{WP}(\text{FIM}_1)$  and  $\iota(\text{FIM}_1)$  are both  $2\text{-CF}$ .*

*Proof.* Let  $X = \{x\}$ ,  $Y = \{x, \bar{x}\}$  and  $L = \text{WP}(\text{FIM}_1, X)$ . We can express  $L$  as the intersection of the following two languages:

$$L_\nu = \{u\#v^{\text{inv}} \mid u, v \in Y^*, \nu(u) = \nu(v) \wedge \mu(u) = \mu(v)\}$$

and

$$L_\lambda = \{u\#v^{\text{inv}} \mid u, v \in Y^*, \lambda(u) = \lambda(v) \wedge \mu(u) = \mu(v)\}.$$

We will show that  $L_\nu$  and  $L_\lambda$  are each context-free and hence  $L$  is  $2\text{-CF}$ . Since  $L_\lambda$  is simply the reverse of  $L_\nu$ , it suffices to prove that  $L_\nu$  is context-free.

Let  $\Gamma_\nu = (V, \Sigma, P, S)$  be the context-free grammar with nonterminals  $V = \{S, T, Z, Z'\}$ , terminals  $\Sigma = \{x, \bar{x}, \#\}$  and productions  $P$  as follows:

$$\begin{aligned} S &\rightarrow ZSZ \mid xS\bar{x} \mid T \\ T &\rightarrow ZTZ \mid \bar{x}Tx \mid \# \\ Z &\rightarrow ZZ \mid \bar{x}Zx \mid \varepsilon. \end{aligned}$$

Any derivation in  $\Gamma_\nu$  can be expressed as

$$S \Rightarrow \alpha S \beta \rightarrow \alpha T \beta \Rightarrow u_1 u_2 \# v_2 v_1, \quad (1)$$

where  $\alpha \Rightarrow u_1$ ,  $\beta \Rightarrow v_1$  and  $T \Rightarrow u_2 \# v_2$ .

For any  $\alpha' \in \{Z, x\}^*$  and  $\beta' \in \{Z, \bar{x}\}^*$  with  $|\alpha'|_x = |\beta'|_{\bar{x}}$ , there is a partial derivation in  $\Gamma_\nu$ , not involving the production  $S \rightarrow T$ , from  $S$  to  $\alpha' S \beta'$ . Conversely, any partial derivation from  $S$  not involving  $S \rightarrow T$  results in a string  $\alpha S \beta$  in which  $\alpha$  and  $\beta$  can be derived from some such  $\alpha'$  and  $\beta'$  respectively.

Let  $\alpha \in \{Z, x\}^*$  and  $w \in Y^*$  with  $\alpha \Rightarrow^* w$ . By Lemma 1, the subwords of  $w$  produced from instances of  $Z$  in  $\alpha$  evaluate to negative idempotents, and so have no effect on  $\nu(w)$  or  $\mu(w)$ , whereas each  $x$  in  $\alpha$  increases both  $\nu(w)$  and  $\mu(w)$  by 1. Hence  $\nu(w) = \mu(w) = |\alpha|_x$ . Thus a pair of words  $u_1$  and  $v_1$  can appear in the derivation (1) if and only if  $\nu(u_1) = \mu(u_1) = \mu(v_1^{\text{inv}}) = \mu(v_1^{\text{inv}})$ . Similarly, it can be shown that  $T \Rightarrow u_2 \# v_2$  if and only if  $\nu(u_2) = \nu(v_2^{\text{inv}}) = 0$  and  $\mu(u_2) = \mu(v_2^{\text{inv}}) \leq 0$ .

Hence  $S \Rightarrow u \# v$  if and only if we can write  $u = u_1 u_2$  and  $v = v_2 v_1$  such that there exist  $l_1, l_2, l'_1, l'_2, m, n \in \mathbb{N}_0$  with

$$\begin{aligned} u_1 &=_{\text{FIM}_1} (-l_1, n, n) & u_2 &=_{\text{FIM}_1} (-l_2, -m, 0) \\ v_1^{\text{inv}} &=_{\text{FIM}_1} (-l'_1, n, n) & v_2^{\text{inv}} &=_{\text{FIM}_1} (-l'_2, -m, 0). \end{aligned}$$

If  $u \# v \in L_\nu$ , then we can express  $u$  and  $v$  in this way by setting  $u_1$  to be the shortest prefix of  $u$  such that  $\nu(u_1) = \nu(u)$  and  $v_1^{\text{inv}}$  the shortest prefix of  $v^{\text{inv}}$  such that  $\nu(v_1^{\text{inv}}) = \nu(v^{\text{inv}})$ . Conversely, supposing we can express  $u$  and  $v$  in this way, we have

$$\begin{aligned} u &=_{\text{FIM}_1} (-l_1, n, n)(-l_2, n, -m) = (-i, n, n - m) \\ v^{\text{inv}} &=_{\text{FIM}_1} (-l'_1, n, n)(-l'_2, 0, -m) = (-j, n, n - m) \end{aligned}$$

for some  $i, j \in \mathbb{N}_0$ . That is,  $u \# v \in L_\nu$ .

Hence  $L_\nu$  is generated by  $\Gamma_\nu$  and is context-free, and therefore  $L_\nu^{\text{rev}} = L_\lambda$  is also context-free. Thus  $\text{WP}(\text{FIM}_1, X) = L_\nu \cap L_\lambda$  is  $2\text{-CF}$ .

For variety, we give an automaton proof for  $\iota(\text{FIM}_1, X)$ . Define sublanguages  $L_\nu^t$  and  $L_\lambda^t$  of  $Y^* \times Y^*$  analogously to  $L_\nu$  and  $L_\lambda$ . Let  $x_1 = (x, \epsilon)$ ,  $x_2 = (\epsilon, x)$  and define  $\bar{x}_1, \bar{x}_2$  similarly. Reading  $x_i$  or  $\bar{x}_i$  means that we read an  $x$  or  $\bar{x}$  from the  $i$ -th tape and nothing from the other tape. Define a pushdown automaton  $\mathcal{A}_\nu$  with states  $q_0, q_1$  by the following transitions for  $i = 1, 2$  ( $Z$  is the bottom-of-stack

symbol):

$$\begin{array}{ll}
(q_0, Z, (x, x)) \mapsto (q_0, Z) & (q_1, Z, (\bar{x}, \bar{x})) \mapsto (q_1, Z) \\
(q_0, Z, \bar{x}_i) \mapsto (q_0, Y_i Z) & (q_1, Z, \bar{x}_i) \mapsto (q_1, Y_i Z) \\
(q_0, Y_i, \bar{x}_i) \mapsto (q_0, Y_i Y_i) & (q_1, Y_i, \bar{x}_i) \mapsto (q_1, Y_i Y_i) \\
(q_0, Y_i, x_i) \mapsto (q_0, \epsilon) & (q_1, Y_i, x_i) \mapsto (q_1, \epsilon). \\
(q_0, Z, \epsilon) \mapsto (q_1, Z) &
\end{array}$$

The language  $\mathcal{A}_\nu$  accepts by empty stack consists of all pairs  $(u, v)$  where  $u, v \in (E^-x)^n(E^-\bar{x})^kE^-$  for some  $n, k \in \mathbb{N}_0$ , which is precisely the language  $L_\nu^t$ . Switching the roles of  $x$  and  $\bar{x}$  in  $\mathcal{A}_\nu$  gives rise to a pushdown automaton  $\mathcal{A}_\lambda$  accepting  $L_\lambda^t$ . Hence  $\iota(\text{FIM}_1, X)$  is also  $2\text{-CF}$ .  $\square$

**Theorem 3.** *Both versions of the co-word problem of  $\text{FIM}_1$  are context-free.*

*Proof.* Let  $K = \text{coWP}(\text{FIM}_1, X) = \{u\#v^{\text{inv}} \mid u, v \in Y^*, u \neq_{\text{FIM}_1} v\}$ . A word  $w = u\#v$  with  $u, v \in Y^*$  is in  $K$  if and only if the path traced out by  $w$  starting at 0 either does not end at 0, or its minimum or maximum value is not achieved both before and after  $\#$  (recall that this includes not being achieved at the end of  $u$ ). Thus a context-free grammar for  $K$  with start symbol  $S$  is given by the following productions:

$$\begin{array}{ll}
S \rightarrow M \mid U \mid D & U \rightarrow ZxU\bar{x}Z \mid xE\bar{x}Z\# \mid \#ZxE\bar{x}Z \\
M \rightarrow ExA \mid E\bar{x}B & D \rightarrow Z'\bar{x}DxZ' \mid \bar{x}ExZ'\# \mid \#Z'\bar{x}ExZ' \\
A \rightarrow xA \mid EAE \mid \epsilon & Z \rightarrow ZZ \mid \bar{x}Zx \mid \epsilon \\
B \rightarrow \bar{x}B \mid EBE \mid \epsilon & Z' \rightarrow Z'Z' \mid xZ'\bar{x} \mid \epsilon. \\
E \rightarrow ZE \mid Z'E \mid \epsilon
\end{array}$$

$M$  generates all  $u\#v^{\text{inv}}$  with  $\mu(u) \neq \mu(v)$ ;  $U$  generates all  $u\#v^{\text{inv}}$  with  $wv^{\text{inv}}$  idempotent but  $\nu(u) \neq \nu(v)$ , and  $D$  does the same as  $U$  but for  $\lambda$  instead of  $\nu$ .

The two-tape co-word problem of  $\text{FIM}_1$  with respect to  $X$  is the language  $M = \{(u, v) \in Y^* \times Y^* \mid u \neq_{\text{FIM}_1} v\}$ . A pushdown automaton recognising  $M$  can be expressed as the union of automata  $\mathcal{B}_\mu, \mathcal{B}_\nu, \mathcal{B}_\lambda$ . The automaton  $\mathcal{B}_\mu$  checks that  $|u|_x - |u|_{\bar{x}} \neq |v|_x - |v|_{\bar{x}}$  for input  $(u, v)$ , and thus accepts all pairs with  $\mu(u) \neq \mu(v)$ . The automaton  $\mathcal{B}_\nu$  has states  $q_0, q_1, q_2, f$ , with  $f$  being the unique final state, input symbols  $x_i, \bar{x}_i$  (as in the proof of Theorem 2) and transitions:

$$\begin{array}{lll}
(q_0, x_1, Z) \rightarrow (q_0, XZ) & (q_0, \epsilon, *) \rightarrow (q_1, *) & (q_2, x_2, Z) \rightarrow (f, Z) \\
(q_0, x_1, X) \rightarrow (q_0, XX) & (q_1, \epsilon, Y) \rightarrow (q_1, \epsilon) & (q_2, x_2, X) \rightarrow (q_2, \epsilon) \\
(q_0, x_1, Y) \rightarrow (q_0, \epsilon) & (q_1, x_2, X) \rightarrow (q_2, \epsilon) & (q_2, x_2, Y) \rightarrow (q_2, \epsilon) \\
(q_0, \bar{x}_1, *) \rightarrow (q_0, Y*) & (q_1, \bar{x}_2, X) \rightarrow (q_2, YX) & (q_2, \bar{x}_2, *) \rightarrow (q_2, Y*) \\
& & (q_2, \epsilon, X) \rightarrow (f, X),
\end{array}$$

where  $Z$  is the bottom-of-stack marker and  $*$  denotes any stack symbol  $(X, Y, Z)$ . In state  $q_0$ ,  $X^{\nu(u)}Y^{\nu(u)-\mu(u)}$  is placed on the stack. State  $q_1$  removes all  $Y$ 's. State

$q_2$  then checks  $\nu(v)$  against  $\nu(u)$ , moving to the final state  $f$  if we either find that  $\nu(v) > \nu(u)$  or nondeterministically if  $\nu(v') < \nu(u)$  for the prefix  $v'$  of  $v$  read so far, in which case  $\mathcal{B}_\nu$  accepts if there is no further input. Thus  $\mathcal{B}_\nu$  accepts the language of all  $(u, v)$  with  $\nu(u) \neq \nu(v)$ . The automaton  $\mathcal{B}_\lambda$  is obtained by swapping the roles of  $x_i$  and  $\bar{x}_i$  in  $\mathcal{B}_\nu$ , and accepts  $(u, v)$  with  $\lambda(u) \neq \lambda(v)$ .  $\square$

Given the model of elements of  $\text{FIM}_1$  as marked intervals in  $\mathbb{Z}$ , stack automata provide possibly the most natural class of automata to consider as acceptors of its word problem. It turns out that it suffices to use a checking stack automaton.

**Theorem 4.** *WP(FIM<sub>1</sub>) and  $\iota(\text{FIM}_1)$  are each recognised by a checking stack automaton.*

*Proof.* The idea of the checking stack automaton for WP(FIM<sub>1</sub>) is to use the stack contents as a model for an interval of integers  $[-l, n]$  (chosen nondeterministically before beginning to read the input), and check for input  $u\#v^{\text{inv}}$  whether  $\hat{u} = \hat{v} = (-l, n, m)$  for some  $m \in [-l, n]$ . Following the set-up phase, the stack contents will always be of the form  $L^lOR^n$  for some  $l, n \in \mathbb{N}_0$ , with the leftmost symbol  $L$  or  $O$  being marked with a superscript  $-$ , and the rightmost symbol  $O$  or  $R$  being marked with a superscript  $+$ . Such a string represents a guess that the input string  $u\#v^{\text{inv}}$  will have  $\lambda(u) = \lambda(v) = l$  and  $\nu(u) = \nu(v) = n$ . For  $\alpha \in L^*OR^*$ , we denote the string ‘ $\alpha$  with marked endpoints’ by  $[\alpha]$ . For example,  $[LLORR] = L^-LORR^+$  and  $[O] = O^\pm$ . Before beginning to consume the input, the stack marker is moved to  $O^{(+,-,\pm)}$ .

During the checking phase, the automaton  $\mathcal{A}$  moves up and down the stack, tracing out the path given by  $u\#v^{\text{inv}}$ , accepting if and only if three conditions are satisfied: (i) both the left and right endpoints of  $[\alpha]$  are reached at least once before and after the  $\#$ ; (ii) the automaton never attempts to move beyond the endpoints of  $[\alpha]$ ; and (iii) the automaton ends with the stack marker pointing at  $O^{(+,-,\pm)}$ . If during the set-up phase the string  $[L^lOR^n]$  was placed on the stack, then the set of words accepted by  $\mathcal{A}$  following that particular set-up will be the language  $\{u\#v^{\text{inv}} \mid u, v \in Y^*, \lambda(u) = \lambda(v) = l, \nu(u) = \nu(v) = n, \mu(u) = \nu(v)\}$ .

More formally, the checking transitions of  $\mathcal{A}$  are described as follows, using states  $\{q_i, q_i^+, q_i^-, q_i^*, f \mid i = 1, 2\}$ , with  $f$  the unique final state. Following the setup phase (which can be achieved non-deterministically using two states),  $\mathcal{A}$  is in state  $q_1$ , with stack contents  $[\alpha]$  for some string  $\alpha \in L^*OR^*$ , and the stack marker pointing at the symbol corresponding to  $O$  in  $[\alpha]$ . The symbol  $\$$  is an end-of-input marker, standardly included in the definition of stack automata. Let  $\Delta^- = \{O^-, R^-\}$  and  $\Delta^+ = \{O^+, L^+\}$ . The left-hand side of a transition represents the current automaton configuration (state, input, stack symbol). The right-hand side has first component the state to be moved to, and second component the direction in which to move the stack marker (with  $-$  denoting no change). The full set of stack symbols is  $\Gamma = \{L^{(+,-)}, O^{(+,-,\pm)}, R^{(+,-)}\}$ . For  $i = 1, 2$ :

$$\begin{aligned}
(q_1, \#, O^\pm) &\mapsto (q_2^*, -), \\
(q_i, x, C) &\mapsto (q_i, \uparrow), \quad C \in \{L, O, R\} \\
(q_i, \bar{x}, C) &\mapsto (q_i, \downarrow), \quad C \in \{L, O, R\} \\
(q_i, x, C) &\mapsto (q_i^-, \uparrow), \quad C \in \Delta^- \quad (q_i^-, x, C) \mapsto (q_i^*, \uparrow), \quad C \notin \{R^+, O^+, O^\pm\} \\
(q_i, \bar{x}, C) &\mapsto (q_i^+, \downarrow), \quad C \in \Delta^+ \quad (q_i^+, \bar{x}, C) \mapsto (q_i^*, \downarrow), \quad C \notin \{L^-, O^-, O^\pm\} \\
(q_i^+, x, C) &\mapsto (q_i^*, \uparrow), \quad C \in \Delta^- \quad (q_1^+, \#, C) \mapsto (q_2, -), \quad C \in \Gamma \\
(q_i^-, \bar{x}, C) &\mapsto (q_i^*, \downarrow), \quad C \in \Delta^+ \quad (q_2^*, \$, C) \mapsto (f, -), \quad C \in \{O, O^-, O^+, O^\pm\}.
\end{aligned}$$

Note that these transitions involve no push or pop operations, so  $\mathcal{A}$  is a checking stack automaton. Now assume that  $\mathcal{A}$  has reached the reading phase, with stack contents  $[L^l O R^n]$  for some  $l, n \in \mathbb{N}_0$ . Let  $L_{(l,n)}$  denote the language of all words accepted by  $\mathcal{A}$  from this configuration. The case  $(l, n) = (0, 0)$  is degenerate, since  $[O] = O^\pm$  and the only path from  $q_0$  to  $f$  in this case is on input  $\#\$, which is exactly as desired since the empty word is the only representative of the identity  $(0, 0, 0)$  in  $Y^*$ . Henceforth assume at least one of  $l$  or  $n$  is non-zero.$

With few exceptions, the automaton moves up the stack on input  $x$  and down on  $\bar{x}$ . The exceptions are when this would otherwise result in moving beyond the top or bottom of the stack. In these cases there are no transitions defined and so the automaton fails. Thus for  $w \in Y^*$  the stack marker traces out the path given by  $w$ , provided this path remains within the interval  $[-l, n]$ .

When the automaton is in state  $q_i$  and has reached the top of the stack (indicated by a symbol in  $\Delta^+$ ), on the next input it either fails (on  $x$ ) or moves to state  $q_i^+$  (on  $\bar{x}$ ). Similarly, after reaching the bottom of the stack (symbols in  $\Delta^-$ ), the automaton either fails (on  $\bar{x}$ ) or moves to  $q_i^-$  (on  $x$ ). Following either of these events, the automaton will move to state  $q_i^*$  after reaching the opposite end of the stack, provided it does not fail. Thus being in state  $q_0^*$  indicates that the automaton has read some  $u' \in Y^*$  with  $\lambda(u') = l$  and  $\nu(u') = n$ .

The only transition on the symbol  $\#$  is from state  $q_0^*$  to  $q_1$  (regardless of stack symbol), and the only transition on  $\$$  is from  $q_1^*$  to the final state  $f$  and requires the automaton to be pointing at  $O^{(+,-)}$  (both of these transitions leave the stack unchanged). Hence  $L_{(l,n)}$  contains exactly those words in  $Y^* \# Y^* \$$  which trace out a path in  $[-l, n]$  starting and ending at  $O^{(+,-)}$  which visit the top and bottom of the stack each at least once before and after the  $\#$ ; that is,  $L_{(l,n)}$  consists of all  $u \# v^{\text{inv}} \$$  such that  $u \# v^{\text{inv}}$  is in  $\text{WP}(\text{FIM}_1, X)$  and  $\lambda(u) = l$ ,  $\nu(u) = n$ , as desired. Since the language accepted by  $\mathcal{A}$  is  $\bigcup_{l,n \in \mathbb{N}_0} L_{(l,n)}$ , we conclude that  $\mathcal{A}$  accepts  $\text{WP}(\text{FIM}_1, X)$ .

To recognise  $\iota(\text{FIM}_1, X)$ , we make a few small modifications to  $\mathcal{A}$ : in the setup phase, we additionally mark some symbol of the stack contents  $[\alpha]$  to denote a guess as to the location of  $\mu(u) = \mu(v)$  (where the input is  $(u \$, v \$)$ ). In states  $q_i, q_i^+, q_i^-, q_i^*$ , we read from the  $i$ -th tape ( $i = 1, 2$ ). On reaching the end symbol  $\$$  on each tape, we only proceed if the stack marker is pointing at the marked symbol. We introduce an intermediate state between  $q_1^*$  and  $q_2$  which returns the stack marker to the symbol corresponding to  $O$  in  $\alpha$ . In all other respects, the automaton behaves the same as  $\mathcal{A}$ . Thus the stack contents of the modified automaton represent an element  $(-l, n, m)$  of  $\text{FIM}_1$ , and with these stack contents the automaton accepts all  $(u, v)$  such that  $u$  and  $v$  both evaluate to  $(-l, n, m)$ . Evaluating over all possible stack contents yields  $\iota(\text{FIM}_1, X)$ .  $\square$

Note that the classes of 2- $\mathcal{CF}$  and checking stack languages are incomparable. The language  $\{ww \mid w \in A^*\}$  for  $|A| \geq 2$  is not poly- $\mathcal{CF}$  (an easy application of [3, Theorem 3.9]), but is accepted by a checking stack automaton that starts by putting a word  $w$  on the stack and then checks whether the input is  $ww$ . The language  $\{(ab^n)^n \mid n \in \mathbb{N}\}$  is not even indexed [15, Theorem 5.3], but is 3- $\mathcal{CF}$ .

Since E(D)TOL languages have been shown to describe various languages arising in group theory [5, 6] (but not word problems), it is worth noting the following.

**Corollary 1.** *WP(FIM<sub>1</sub>) and  $\iota$ (FIM<sub>1</sub>) are both ETOL.*

*Proof.* This follows from Theorem 4 and the fact that the class of checking stack languages is contained in the ETOL languages [25].  $\square$

The author has constructed nondeterministic ETOL grammars for both versions of the word problem of FIM<sub>1</sub> with 9 tables and 11 nonterminals. The nondeterminism arises from the fact that for any word  $w \in Y^*$ , we may insert idempotents arbitrarily at any point in  $w$  without changing the element represented, provided that these idempotents are not ‘large’ enough to change the value of  $\nu(w)$  or  $\lambda(w)$ .

**Conjecture 1.** *Neither WP(FIM<sub>1</sub>) nor  $\iota$ (FIM<sub>1</sub>) is EDTOL.*

## 4 Rank Greater Than 1

The word problem for inverse monoids in higher ranks is more complex from a language theory perspective.

**Lemma 2.** *For any  $k \geq 3$ , WP(FIM<sub>k</sub>) is not  $(k - 2)$ - $\mathcal{CF}$ .*

*Proof.* For any  $k \geq 2$ , let  $X_k = \{x_1, \dots, x_k, \bar{x}_1, \dots, \bar{x}_k\}$  and let

$$L_k = \{x_1^{m_1} \bar{x}_1^{m_1} \dots x_k^{m_k} \bar{x}_k^{m_k} \# x_1^{n_1} \bar{x}_1^{n_1} \dots x_k^{n_k} \bar{x}_k^{n_k} \mid m_i, n_i \in \mathbb{N}_0\}.$$

Let  $W_k = \text{WP}(\text{FIM}_k, X_k)$ . Then  $W_k$  consists of all those words in  $L_k$  with  $m_i = n_i$  for all  $i$  (since idempotents in FIM<sub>k</sub> commute). By [3, Theorem 3.12],  $W_k$  is not  $(k - 1)$ -context-free<sup>1</sup>. Since  $W_k$  is the intersection of  $\text{WP}(\text{FIM}_k, X_k)$  with the context-free language  $L_k$ , this implies that  $\text{WP}(\text{FIM}_k, X_k)$  is not context-free.  $\square$

Since for  $k \geq 2$  FIM<sub>k</sub> contains submonoids isomorphic to FIM<sub>n</sub> for all  $n$ , the following theorem is immediate from Lemma 2.

<sup>1</sup> The language  $L^{(2,k)}$  in the referenced result is not precisely  $W_k$ , but the associated set of integer tuples differs from that associated to  $W_k$  only by a constant (arising from the symbol  $\#$ ), which does not affect stratification properties and therefore does not affect the property of not being  $(k - 1)$ - $\mathcal{CF}$ .



**Theorem 5.** *For any  $k \geq 2$ ,  $\text{WP}(\text{FIM}_k)$  is not poly- $\mathcal{CF}$ .*

Note that the argument in Lemma 2 does not work for  $\iota(\text{FIM}_k)$ , since the set  $\{(x_1^{m_1} \bar{x}_1^{m_1} \dots x_k^{m_k} \bar{x}_k^{m_k}, x_1^{m_1} \bar{x}_1^{m_1} \dots x_k^{m_k} \bar{x}_k^{m_k}) \mid m_i \in \mathbb{N}_0\}$  is context-free. Writing the idempotent on the second tape in reverse (that is, starting with  $x_k^{m_k} \bar{x}_k^{m_k}$ ) does not help, as the resulting language is still 2- $\mathcal{CF}$ . It appears likely that the intersection of  $\iota(\text{FIM}_k)$  with the following  $\mathcal{CF}$  language would not be  $(k-1)$ - $\mathcal{CF}$ :

$$\{(x_1^{l_1} \bar{x}_1^{l_1} \dots x_k^{l_k} \bar{x}_k^{l_k} x_1^{m_1} \bar{x}_1^{m_1} \dots x_k^{m_k} \bar{x}_k^{m_k}, x_1^{n_1} \bar{x}_1^{n_1} \dots x_k^{n_k} \bar{x}_k^{n_k}) \mid l_i, m_i, n_i \in \mathbb{N}_0\},$$

but proving this would require delicate arguments about intersections of stratified semilinear sets beyond the scope of this paper.

The author conjectures that neither version of the word problem for  $\text{FIM}_k$ ,  $k \geq 2$  is indexed. While a nested stack automaton can easily be used to store a Munn tree, there appears to be no way to check while reading a word  $w \in X^*$  that the path traced out by  $w$  visits every leaf of the stored Munn tree.

## References

1. Aho, A.: Nested stack automata. *J. ACM* **16**(3), 383–406 (1969)
2. Boone, W.W.: The word problem. *Ann. Math.* **2**(70), 207–265 (1959)
3. Brough, T.: Groups with poly-context-free word problem. *Groups Complex. Cryptol.* **6**(1), 9–29 (2014)
4. Brough, T.: Inverse semigroups with rational word problem are finite, Unpublished note. [arxiv:1311.3955](https://arxiv.org/abs/1311.3955) (2013)
5. Ciobanu, L., Diekert, V., Elder, M.: Solution sets for equations over free groups are EDT0L languages. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) *ICALP 2015*. LNCS, vol. 9135, pp. 134–145. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-47666-6\\_11](https://doi.org/10.1007/978-3-662-47666-6_11)
6. Ciobanu, L., Elder, M., Ferov, M.: Applications of L systems to group theory. *Int. J. Algebra Comput.* **28**, 309–329 (2018)
7. Duncan, A., Gilman, R.H.: Word hyperbolic semigroups. *Math. Proc. Camb. Philos. Soc.* **136**, 513–524 (2004)
8. Gilbert, N.D., Noonan Heale, R.: The idempotent problem for an inverse monoid. *Int. J. Algebra Comput.* **21**, 1170–1194 (2011)
9. Gilman, R.H., Shapiro, M.: On groups whose word problem is solved by a nested stack automaton. [arxiv:math/9812028](https://arxiv.org/abs/math/9812028)
10. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. *J. ACM* **14**(2), 389–418 (1967)
11. Greibach, S.: Checking automata and one-way stack languages. *J. Comput. Syst. Sci.* **3**, 196–217 (1969)
12. Hoffman, M., Holt, D.F., Owens, M.D., Thomas, R.M.: Semigroups with a context-free word problem. In: *Proceedings of the 16th International Conference on Developments in Language Theory, DLT 2012*, pp. 97–108 (2012)
13. Holt, D.F., Owens, M.D., Thomas, R.M.: Groups and semigroups with a one-counter word problem. *J. Aust. Math. Soc.* **85**, 197–209 (2005)
14. Holt, D.F., Röver, C.E., Rees, S.E., Thomas, R.M.: Groups with a context-free co-word problem. *J. Lond. Math. Soc.* **71**, 643–657 (2005)

15. Hopcroft, J., Ullman, J.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston (1979)
16. Kambites, M.: Anisimov's Theorem for inverse semigroups. *Int. J. Algebra Comput.* **25**, 41–49 (2015)
17. Lawson, M.V.: *Inverse Semigroups: The Theory of Partial Symmetries*. World Scientific, Singapore (1998)
18. Lohrey, M., Ondrusch, N.: Inverse monoids: decidability and complexity of algebraic questions. *Inf. Comput.* **205**(8), 1212–1234 (2007)
19. Lisovik, L.P., Red'ko, V.N.: Regular events in semigroups. *Problemy Kibernetiki* **37**, 155–184 (1980)
20. Munn, W.D.: Free inverse semigroups. *Proc. Lond. Math. Soc.* **s3–29**(3), 385–404 (1974)
21. Novikov, P.S.: On the algorithmic unsolvability of the word problem in group theory. *Am. Math. Soc. Transl. Ser.* **2**(9), 1–122 (1958)
22. Pfeiffer, M.J.: *Adventures in applying iteration lemmas*, Ph.D. thesis, University of St Andrews (2013)
23. Post, E.: Recursive unsolvability of a problem of Thue. *J. Symb. Log.* **12**(1), 1–11 (1947)
24. Rozenberg, G., Salomaa, A.: *The Book of L*. Springer, Heidelberg (1986). <https://doi.org/10.1007/978-3-642-95486-3>
25. van Leeuwen, J.: Variations of a new machine model. In: *Conference Record 17th Annual IEEE Symposium on Foundations of Computer Science*, pp. 228–235 (1976)



# Most Complex Deterministic Union-Free Regular Languages

Janusz A. Brzozowski<sup>1</sup> and Sylvie Davies<sup>2</sup>(✉)

<sup>1</sup> David R. Cheriton School of Computer Science, University of Waterloo,  
Waterloo, ON N2L 3G1, Canada

brzozo@uwaterloo.ca

<sup>2</sup> Department of Pure Mathematics, University of Waterloo,  
Waterloo, ON N2L 3G1, Canada

sldavies@uwaterloo.ca

**Abstract.** A regular language  $L$  is *union-free* if it can be represented by a regular expression without the union operation. A union-free language is *deterministic* if it can be accepted by a deterministic *one-cycle-free-path* finite automaton; this is an automaton which has one final state and exactly one cycle-free path from any state to the final state. Jirásková and Masopust proved that the state complexities of the basic operations reversal, star, product, and boolean operations in deterministic union-free languages are exactly the same as those in the class of all regular languages. To prove that the bounds are met they used five types of automata, involving eight types of transformations of the set of states of the automata. We show that for each  $n \geq 3$  there exists one ternary witness of state complexity  $n$  that meets the bound for reversal and product. Moreover, the restrictions of this witness to binary alphabets meet the bounds for star and boolean operations. We also show that the tight upper bounds on the state complexity of binary operations that take arguments over different alphabets are the same as those for arbitrary regular languages. Furthermore, we prove that the maximal syntactic semigroup of a union-free language has  $n^n$  elements, as in the case of regular languages, and that the maximal state complexities of atoms of union-free languages are the same as those for regular languages. Finally, we prove that there exists a most complex union-free language that meets the bounds for all these complexity measures. Altogether this proves that the complexity measures above cannot distinguish union-free languages from regular languages.

**Keywords:** Atom · Boolean operation · Concatenation  
Different alphabets · Most complex · One-cycle-free-path · Regular  
Reversal · Star · State complexity · Syntactic semigroup  
Transition semigroup · Union-free

---

This work was supported by the Natural Sciences and Engineering Research Council of Canada grant No. OGP0000871.

# 1 Introduction

Formal definitions are postponed until Sect. 2.

The class of regular languages over a finite alphabet  $\Sigma$  is the smallest class of languages containing the empty language  $\emptyset$ , the language  $\{\varepsilon\}$ , where  $\varepsilon$  is the empty word, and the *letter languages*  $\{a\}$  for each  $a \in \Sigma$ , and closed under the operations of union, concatenation, and (Kleene) star. Hence each regular language can be written as a finite expression involving the above basic languages and operations. An expression defining a regular language in this way is called a *regular expression*. Because regular languages are also closed under complementation, we may also consider regular expressions that allow complementation, which are called *extended regular expressions*. In this paper we deal exclusively with regular languages.

A natural question is: what kind of languages are defined if one of the operations in the definitions given above is missing? If the star operation is removed from the extended regular expressions we get the well known *star-free* languages [10, 21, 26], which have been extensively studied. Less attention was given to classes defined by removing an operation from ordinary regular expressions, but recently language classes defined without union or concatenation have been studied.

If we remove some operations from regular expressions, we obtain the following classes of languages:

**Union only** subsets of  $\{\varepsilon\} \cup \Sigma$ .

**Concatenation only**  $\emptyset$  and  $\{w\}$  for each  $w \in \Sigma^*$ .

**Star only**  $\emptyset$ ,  $\{\varepsilon\}$ ,  $\{a\}$  for each  $a \in \Sigma$ , and  $\{a\}^*$  for each  $a \in \Sigma$ .

**Union and Concatenation.** Finite languages.

**Concatenation and Star.** These are the *union-free* languages that constitute the main topic of this paper.

**Union and Star.** These are the *concatenation-free* languages that were studied in [15, 19].

Union-free regular languages were first considered by Brzozowski [3] in 1962 under the name *star-dot* regular languages, where *dot* stands for concatenation. He proved that every regular language is a union of union-free languages [3, p. 216, Theorem 9.5]<sup>1</sup>. Much more recently, in 2001, Crvenković et al. [13] studied equations satisfied by union-free regular languages, and proved that the class of these languages cannot be axiomatized by a finite set of equations. This is also known to be true for the class of all regular languages. In 2006 Nagy studied union-free languages in detail and characterized them in terms of nondeterministic finite automata (NFAs) recognizing them [22], which he called *one-cycle-free-path* NFAs. In 2009 minimal union-free decompositions of regular languages were studied in [1] by Afonin and Golomazov. They also presented a new algorithm for deciding whether a given deterministic finite automaton (DFA) accepts

---

<sup>1</sup> Terminology changed to that of the present paper.

a union-free language. Decompositions of regular languages in terms of union-free languages were further studied by Nagy in 2010 [23]. The state complexities of operations on union-free languages were examined in 2011 by Jirásková and Masopust [17], who proved that the state complexities of basic operations on these languages are the same as those in the class of all regular languages. It was shown in [17] that the class of languages defined by DFAs with the one-cycle-free-path property is a proper subclass of that defined by one-cycle-free-path NFAs; the former class is called the class of *deterministic union-free* languages. In 2012 Jirásková and Nagy [18] proved that the class of finite unions of deterministic union-free languages is a proper subclass of the class of regular languages. They also showed that every deterministic union-free language is accepted by a special kind of a one-cycle-free-path DFA called a *balloon* DFA. A summary of the properties of union-free languages was presented in 2017 in [15].

## 2 Preliminaries

Let  $L$  be a regular language. We define the *alphabet* of  $L$  to be the set of letters which appear *at least once* in a word of  $L$ . For example, consider the language  $L = \{a, ab, ac\}$  and the subset  $K = \{a, ac\}$ ; we say  $L$  has alphabet  $\{a, b, c\}$  and  $K$  has alphabet  $\{a, c\}$ .

A *deterministic finite automaton* (DFA) is a 5-tuple  $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ , where  $Q$  is a finite non-empty set of *states*,  $\Sigma$  is a finite non-empty *alphabet*,  $\delta: Q \times \Sigma \rightarrow Q$  is the *transition function*,  $q_0 \in Q$  is the *initial* state, and  $F \subseteq Q$  is the set of *final* states. We extend  $\delta$  to functions  $\delta: Q \times \Sigma^* \rightarrow Q$  and  $\delta: 2^Q \times \Sigma^* \rightarrow 2^Q$  as usual (where  $2^Q$  denotes the set of all subsets of  $Q$ ). A DFA  $\mathcal{D}$  *accepts* a word  $w \in \Sigma^*$  if  $\delta(q_0, w) \in F$ . The *language accepted by*  $\mathcal{D}$  is the set of all words accepted by  $\mathcal{D}$ , and is denoted by  $L(\mathcal{D})$ . If  $q$  is a state of  $\mathcal{D}$ , then the language  $L_q(\mathcal{D})$  of  $q$  is the language accepted by the DFA  $(Q, \Sigma, \delta, q, F)$ . A state is *empty* (or *dead* or a *sink state*) if its language is empty. Two states  $p$  and  $q$  of  $\mathcal{D}$  are *equivalent* if  $L_p(\mathcal{D}) = L_q(\mathcal{D})$ . A state  $q$  is *reachable* if there exists  $w \in \Sigma^*$  such that  $\delta(q_0, w) = q$ . A DFA  $\mathcal{D}$  is *minimal* if it has the smallest number of states among all DFAs accepting  $L(\mathcal{D})$ . We say a DFA has a *minimal alphabet* if its alphabet is equal to the alphabet of  $L(\mathcal{D})$ . It is well known that a DFA with a minimal alphabet is minimal if and only if all of its states are reachable and no two states are equivalent.

A *nondeterministic finite automaton* (NFA) is a 5-tuple  $\mathcal{N} = (Q, \Sigma, \delta, I, F)$ , where  $Q$ ,  $\Sigma$  and  $F$  are as in a DFA,  $\delta: Q \times \Sigma \rightarrow 2^Q$ , and  $I \subseteq Q$  is the *set of initial states*. Each triple  $(p, a, q)$  with  $p, q \in Q$ ,  $a \in \Sigma$  is a *transition* if  $q \in \delta(p, a)$ . A sequence  $((p_0, a_0, q_0), (p_1, a_1, q_1), \dots, (p_{k-1}, a_{k-1}, q_{k-1}))$  of transitions, where  $p_{i+1} = q_i$  for  $i = 0, \dots, k-2$  is a *path* in  $\mathcal{N}$ . The word  $a_0 a_1 \dots a_{k-1}$  is the word *spelled* by the path. A word  $w$  is *accepted* by  $\mathcal{N}$  if there exists a path with  $p_0 \in I$  and  $q_{k-1} \in F$  that spells  $w$ . If  $q \in \delta(p, a)$  we also use the notation  $p \xrightarrow{a} q$ . We extend this notation also to words, and write  $p \xrightarrow{w} q$  for  $w \in \Sigma^*$ .

The *state complexity* [20, 27] of a regular language  $L$ , denoted by  $\kappa(L)$ , is the number of states in the minimal DFA accepting  $L$ . Henceforth we frequently

refer to state complexity simply as *complexity*, and we denote a language of complexity  $n$  by  $L_n$ , and a DFA with  $n$  states by  $\mathcal{D}_n$ .

The *state complexity* of a regularity-preserving *unary* operation  $\circ$  on regular languages is the maximal value of  $\kappa(L^\circ)$ , expressed as a function of one parameter  $n$ , where  $L$  varies over all regular languages with complexity at most  $n$ . For example, the state complexity of the reversal operation is  $2^n$ ; it is known that if  $L$  has complexity at most  $n$ , then  $\kappa(L^R) \leq 2^n$ , and furthermore this upper bound is tight in the sense that for each  $n \geq 1$  there exists a language  $L_n$  such that  $\kappa(L_n^R) = 2^n$ . In general, to show that an upper bound on  $\kappa(L^\circ)$  is tight, we need to exhibit a sequence  $(L_n \mid n \geq k) = (L_k, L_{k+1}, \dots)$ , called a *stream*, of languages of each complexity  $n \geq k$  (for some small constant  $k$ ) that meet this upper bound. Often we are not interested in the special-case behaviour of the operation that may occur at very small values of  $n$ ; the parameter  $k$  allows us to ignore these small values and simplify the statements of results.

The *state complexity* of a regularity-preserving *binary* operation  $\circ$  on regular languages is the maximal value of  $\kappa(L' \circ L)$ , expressed as a function of two parameters  $m$  and  $n$ , where  $L'$  varies over all regular languages of complexity at most  $m$  and  $L$  varies over all regular languages of complexity at most  $n$ . In this case, to show an upper bound on the state complexity is tight, we need to exhibit two classes  $(L'_{m,n} \mid m \geq h, n \geq k)$  and  $(L_{m,n} \mid m \geq h, n \geq k)$  of languages meeting the bound; the notation  $L'_{m,n}$  and  $L_{m,n}$  implies that  $L'_{m,n}$  and  $L_{m,n}$  depend on both  $m$  and  $n$ . However, in most cases studied in the literature, it is enough to use witness streams  $(L'_m \mid m \geq h)$  and  $(L_n \mid n \geq k)$ , where  $L'_m$  is independent of  $n$  and  $L_n$  is independent of  $m$ .

For binary operations we consider two types of state complexity: *restricted* and *unrestricted* state complexity. For restricted state complexity the operands of the binary operations are required to have the same alphabet. For unrestricted state complexity the alphabets of the operands may differ. See [9] for more details.

Sometimes the same stream can be used for both operands of a binary operation, but this is not always possible. For example, for boolean operations when  $m = n$ , the state complexity of  $L_n \cup L_n = L_n$  is  $n$ , whereas the upper bound is  $mn = n^2$ . However, in many cases the second language is a “dialect” of the first, that is, it “differs only slightly” from the first. The notion “differs only slightly” is defined as follows [4, 6, 8]: Let  $\Sigma = \{a_1, \dots, a_k\}$  be an alphabet ordered as shown; if  $L \subseteq \Sigma^*$ , we denote it by  $L(a_1, \dots, a_k)$  to stress its dependence on  $\Sigma$ . A *dialect* of  $L_n(\Sigma)$  is a language obtained from  $L_n(\Sigma)$  by deleting some letters of  $\Sigma$  in the words of  $L_n(\Sigma)$  – by this we mean that words containing these letters are deleted – or replacing them by letters of another alphabet  $\Sigma'$ . In this paper we consider only the cases where  $\Sigma = \Sigma'$ , and we encounter only two types of dialects:

1. A dialect in which some letters were deleted; for example,  $L_n(a, b)$  is a dialect of  $L_n(a, b, c)$  with  $c$  deleted, and  $L_n(a, -, c)$  is a dialect with  $b$  deleted. Note that deleted letters are replaced by dashes, and if the letters  $\{a_i, \dots, a_k\}$  are all deleted then the corresponding dashes are not shown.

2. A dialect in which the roles of two letters are exchanged; for example,  $L_n(b, a)$  is such a dialect of  $L_n(a, b)$ .

These two types of dialects can be combined, for example, in  $L_n(a, -, b)$  the letter  $c$  is deleted, and  $b$  plays the role that  $c$  played originally. The notion of dialects also extends to DFAs; for example, if  $\mathcal{D}_n(a, b, c)$  recognizes  $L_n(a, b, c)$  then  $\mathcal{D}_n(a, -, b)$  recognizes the dialect  $L_n(a, -, b)$ .

We use  $Q_n = \{0, \dots, n-1\}$  as our basic set with  $n$  elements. A *transformation* of  $Q_n$  is a mapping  $t: Q_n \rightarrow Q_n$ . The *image* of  $q \in Q_n$  under  $t$  is denoted by  $qt$ , and this notation is extended to subsets of  $Q_n$ . The *preimage* of  $q \in Q_n$  under  $t$  is the set  $qt^{-1} = \{p \in Q_n : pt = q\}$ , and this notation is extended to subsets of  $Q_n$  as follows:  $St^{-1} = \{p \in Q_n : pt \in S\}$ . The *rank* of a transformation  $t$  is the cardinality of  $Q_n t$ . If  $s$  and  $t$  are transformations of  $Q_n$ , their composition is denoted  $st$  and we have  $q(st) = (qs)t$  for  $q \in Q_n$ . The  $k$ -fold composition  $tt \cdots t$  (with  $k$  occurrences of  $t$ ) is denoted  $t^k$ , and for  $S \subseteq Q_n$  we define  $St^{-k} = S(t^k)^{-1}$ . Let  $\mathcal{T}_{Q_n}$  be the set of all  $n^n$  transformations of  $Q_n$ ; then  $\mathcal{T}_{Q_n}$  is a monoid under composition.

For  $k \geq 2$ , a transformation  $t$  of a set  $P = \{q_0, q_1, \dots, q_{k-1}\} \subseteq Q_n$  is a  $k$ -*cycle* if  $q_0 t = q_1, q_1 t = q_2, \dots, q_{k-2} t = q_{k-1}, q_{k-1} t = q_0$ . This  $k$ -cycle is denoted by  $(q_0, q_1, \dots, q_{k-1})$ , and leaves the states in  $Q_n \setminus P$  unchanged. A 2-cycle  $(q_0, q_1)$  is called a *transposition*. A transformation that sends state  $p$  to  $q$  and acts as the identity on the remaining states is denoted by  $(p \rightarrow q)$ . The identity transformation is denoted by  $\mathbf{1}$ .

Let  $\mathcal{D} = (Q_n, \Sigma, \delta, 0, F)$  be a DFA. For each word  $w \in \Sigma^*$ , the transition function induces a transformation  $\delta_w$  of  $Q_n$  by  $w$ : for all  $q \in Q_n$ ,  $q\delta_w = \delta(q, w)$ . The set  $\mathcal{T}_{\mathcal{D}}$  of all such transformations by non-empty words is the *transition semigroup* of  $\mathcal{D}$  under composition. Often we use the word  $w$  to denote the transformation  $t$  it induces; thus we write  $qw$  instead of  $q\delta_w$ . We also write  $w: t$  to mean that  $w$  induces the transformation  $t$ .

The size of the *syntactic semigroup* of a regular language is another measure of the complexity of the language [4]. Write  $\Sigma^+$  for  $\Sigma^* \setminus \{\varepsilon\}$ . The *syntactic congruence* of a language  $L \subseteq \Sigma^*$  is defined on  $\Sigma^+$  as follows: For  $x, y \in \Sigma^+$ ,  $x \approx_L y$  if and only if  $wxz \in L \Leftrightarrow wyz \in L$  for all  $w, z \in \Sigma^*$ . The quotient set  $\Sigma^+ / \approx_L$  of equivalence classes of  $\approx_L$  is a semigroup, the *syntactic semigroup*  $T_L$  of  $L$ . The syntactic semigroup is isomorphic to the *transition semigroup* of the minimal DFA of  $L$  [24].

The (left) *quotient* of  $L \subseteq \Sigma^*$  by a word  $w \in \Sigma^*$  is the language  $w^{-1}L = \{x : wx \in L\}$ . It is well known that the number of quotients of a regular language is finite and equal to the state complexity of the language.

The atoms of a regular language are defined by a left congruence, where two words  $x$  and  $y$  are congruent whenever  $ux \in L$  if and only if  $uy \in L$  for all  $u \in \Sigma^*$ . Thus  $x$  and  $y$  are congruent whenever  $x \in u^{-1}L$  if and only if  $y \in u^{-1}L$  for all  $u \in \Sigma^*$ . An equivalence class of this relation is an *atom* of  $L$  [12]. Atoms can be expressed as non-empty intersections of complemented and uncomplemented quotients of  $L$ . The number of atoms and their state complexities were suggested

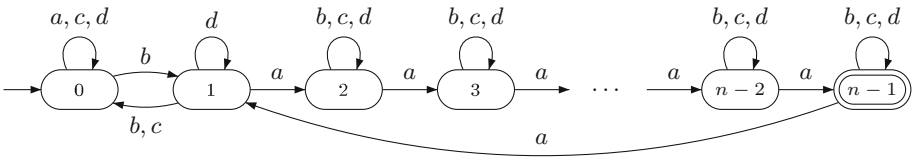
as measures of complexity of regular languages [4] because all quotients of a language and all quotients of its atoms are unions of atoms [11, 12, 16].

### 3 Main Results

The automata described in [22] that characterize union-free languages are called there *one-cycle-free-path* automata. They are defined by the property that there is only one final state and a unique cycle-free path from each state to the final state. We are now ready to define a most complex deterministic one-cycle-free-path DFA and its most complex deterministic union-free language.

The most complex stream below meets all of our complexity bounds. However, our witness uses three letters for restricted product whereas [17] uses binary witnesses. The same shortcoming of most complex streams occurs in the case of regular languages [4]; that seems to be the price of getting a witness for all operations rather than minimizing the alphabet for each operation.

**Definition 1.** For  $n \geq 3$ , let  $\mathcal{D}_n = \mathcal{D}_n(a, b, c, d) = (Q_n, \Sigma, \delta_n, 0, \{n - 1\})$ , where  $\Sigma = \{a, b, c, d\}$ , and  $\delta_n$  is defined by the transformations  $a: (1, \dots, n - 1)$ ,  $b: (0, 1)$ ,  $c: (1 \rightarrow 0)$ , and  $d: \mathbf{1}$ ; see Fig. 1. Let  $L_n = L_n(a, b, c, d)$  be the language accepted by  $\mathcal{D}_n(a, b, c, d)$ .



**Fig. 1.** Most complex minimal one-cycle-free-path DFA  $\mathcal{D}_n(a, b, c, d)$  of Definition 1.

The DFA of Definition 1 bears some similarities to the DFA for reversal in Fig. 6 in [17, p. 1650]. It is evident that it is a one-cycle-free-path DFA. Let  $E = (a(b \cup c \cup d)^*)^{n-2}a$ . One verifies that

$$L_n = [(a \cup c \cup d) \cup b(d \cup E(b \cup c \cup d)^*a)^*(b \cup c)]^* b(d \cup E(b \cup c \cup d)^*a)^*E(b \cup c \cup d)^*.$$

Noting that  $(E_1 \cup E_2 \cup \dots \cup E_k)^* = (E_1^*E_2^* \dots E_k^*)^*$  for all regular expressions  $E_i, i = 1, \dots, k$ , we obtain a union-free expression for  $L_n$ .

**Theorem 1 (Most Complex Deterministic Union-Free Languages).**  
 For each  $n \geq 3$ , the DFA of Definition 1 is minimal and recognizes a deterministic union-free language. The stream  $(L_n(a, b, c) \mid n \geq 3)$  with some dialect streams is most complex in the class of deterministic union-free languages in the following sense:



1. The syntactic semigroup of  $L_n(a, b, c)$  has cardinality  $n^n$ , and at least three letters are required to reach this bound.
2. Each quotient of  $L_n(a, b)$  has complexity  $n$ .
3. The reverse of  $L_n(a, b, c)$  has complexity  $2^n$ . Moreover,  $L_n(a, b, c)$  has  $2^n$  atoms.
4. Each atom  $A_S$  of  $L_n(a, b, c)$  has maximal complexity:

$$\kappa(A_S) = \begin{cases} 2^n - 1, & \text{if } S \in \{\emptyset, Q_n\}; \\ 1 + \sum_{x=1}^{|S|} \sum_{y=1}^{n-|S|} \binom{n}{x} \binom{n-x}{y}, & \text{if } \emptyset \subsetneq S \subsetneq Q_n. \end{cases}$$

5. The star of  $L_n(a, b)$  has complexity  $2^{n-1} + 2^{n-2}$ .
6. (a) Restricted product:  $\kappa(L_m(a, b, c)L_n(a, b, c)) = (m-1)2^n + 2^{n-1}$ .  
 (b) Unrestricted product:  $\kappa(L_m(a, b, c)L_n(a, b, c, d)) = m2^n + 2^{n-1}$ .
7. (a) Restricted boolean operations: For  $(m, n) \neq (3, 3)$ ,  $\kappa(L_m(a, b) \circ L_n(b, a)) = mn$  for all binary boolean operations  $\circ$  that depend on both arguments.  
 (b) Additionally, when  $m \neq n$ ,  $\kappa(L_m(a, b) \circ L_n(a, b)) = mn$ .  
 (c) Unrestricted boolean operations ( $\oplus$  denotes symmetric difference):

$$\begin{cases} \kappa(L_m(a, b, -, c) \circ L_n(b, a, -, d)) = (m+1)(n+1) \text{ if } \circ \in \{\cup, \oplus\}, \\ \kappa(L_m(a, b, -, c) \setminus L_n(b, a)) = mn + n, \\ L_m(a, b) \cap L_n(b, a) = mn. \end{cases}$$

All of these bounds are maximal for deterministic union-free languages.

*Proof.* Only state 0 accepts  $ba^{n-2}$ , and the shortest word accepted by state  $q$ ,  $1 \leq q \leq n-1$ , is  $a^{n-1-q}$ . Hence all the states are distinguishable, and  $\mathcal{D}_n$  is minimal. We noted above that it recognizes a deterministic union-free language.

1. It is well known that the three transformations  $a'$ :  $(0, \dots, n-1)$ ,  $b$ :  $(0, 1)$ , and  $c$ :  $(1 \rightarrow 0)$  generate all  $n^n$  transformations of  $Q_n$ . We have  $b$  and  $c$  in  $\mathcal{D}_n$ , and  $a'$  is generated by  $ab$ . Hence our semigroup is maximal.
2. This is easily verified.
3. By [12] the number of atoms is the same as the complexity of the reverse. By [25] the complexity of the reverse is  $2^n$ .
4. The proof in [5] applies here as well.
5. We construct an NFA for  $(L_n(a, b))^*$  by taking  $\mathcal{D}_n(a, b)$  and adding a new initial accepting state  $s$  with  $s \xrightarrow{a} 0$  and  $s \xrightarrow{b} 1$ , and adding new transitions  $n-2 \xrightarrow{a} 0$  and  $n-1 \xrightarrow{b} 0$ ; then we determinize to get a DFA. For  $S \subseteq Q_n$  and  $a \in \Sigma$ , the transition function of the DFA is given by

$$Sa = \begin{cases} Sa \cup \{0\}, & \text{if } n-1 \in Sa; \\ Sa, & \text{otherwise.} \end{cases}$$

We claim that the following states are reachable and pairwise distinguishable: the initial state  $\{s\}$ , states of the form  $\{0\} \cup S$  with  $S \subseteq Q_n \setminus \{0\}$ , and non-empty states  $S$  with  $S \subseteq Q_n \setminus \{0, n-1\}$ , for a total of  $2^{n-1} + 2^{n-2}$  states.

First consider states  $\{0\} \cup S$  with  $S \subseteq Q_n \setminus \{0\}$ . We prove by induction on  $|S|$  that all of these states are reachable. In the process, we will also show that  $S$  is reachable when  $\emptyset \neq S \subseteq Q_n \setminus \{0, n-1\}$ . For the base case  $|S| = 0$ , note that we can reach  $\{0\}$  from the initial state  $\{s\}$  by  $a$ .

To reach  $\{0\} \cup S$  with  $S \subseteq Q_n \setminus \{0\}$  and  $|S| > 0$ , assume we can reach all states  $\{0\} \cup T$  with  $T \subseteq Q_n \setminus \{0\}$  and  $|T| < |S|$ . Let  $q$  be the minimal element of  $S$ ; then  $1 \in Sa^{1-q}$ . More precisely, if  $S = \{q, q_1, q_2, \dots, q_k\}$  with  $1 \leq q < q_1 < \dots < q_k \leq n-1$ , then  $Sa^{1-q} = \{1, q_1 - q + 1, \dots, q_k - q + 1\}$ . Set  $T = Sa^{1-q} \setminus \{1\}$  and note that  $|T| < |S|$ . By the induction hypothesis, we can reach  $\{0\} \cup T$ . Apply  $b$  to reach either  $\{0, 1\} \cup T$  (if  $n-1 \in T$ ) or  $\{1\} \cup T$  (if  $n-1 \notin T$ ). Note that the only way we can have  $n-1 \in T$  is if  $n-1 \in S$  and  $q = 1$ . Now apply  $a^{q-1}$  to reach either  $\{0\} \cup S$  (if  $n-1 \in S$ ) or just  $S$  (if  $n-1 \notin S$ ). In the latter case, we can apply  $a^{n-1}$  to reach  $\{0\} \cup S$ . This shows that if  $S \subseteq Q_n \setminus \{0\}$ , then  $\{0\} \cup S$  is reachable. Furthermore, if  $S \subseteq Q_n \setminus \{0, n-1\}$  then  $S$  is reachable.

For distinguishability, if  $S, T \subseteq Q_n$  and  $S \neq T$ , let  $q$  be an element of the symmetric difference of  $S$  and  $T$ . If  $q \neq 0$  then  $a^{n-1-q}$  distinguishes  $S$  and  $T$ ; if  $q = 0$  use  $ba^{n-2}$ . To distinguish the accepting state  $\{s\}$  from accepting states  $S \subseteq Q_n$ , use  $b$ .

6. To avoid confusion between the states of  $\mathcal{D}_m$  and  $\mathcal{D}_n$ , we mark the states of  $\mathcal{D}_m$  with primes: instead of  $Q_m$  we use  $Q'_m = \{0', 1', 2', \dots, (m-1)'\}$ . In the restricted case, we construct an NFA for  $L_m(a, b, c)L_n(a, b, c)$  by taking the disjoint union of  $\mathcal{D}_m(a, b, c)$  and  $\mathcal{D}_n(a, b, c)$ , making state  $(m-1)'$  non-final, and adding transitions  $(m-2)' \xrightarrow{a} 0$  and  $(m-1)' \xrightarrow{\sigma} 0$  for  $\sigma \in \{b, c\}$ ; then we determinize to get a DFA. The states of this DFA are sets of the form  $\{q'\} \cup S$ , where  $q' \in Q'_m$  and  $S \subseteq Q_n$ . For  $a \in \Sigma$ , the transition function is given by

$$(\{q'\} \cup S)a = \begin{cases} \{q'a, 0\} \cup Sa, & \text{if } q'a = (m-1)'; \\ \{q'a\} \cup Sa, & \text{otherwise.} \end{cases}$$

In the unrestricted case, we use the same construction with  $\mathcal{D}_m(a, b, c)$  and  $\mathcal{D}_n(a, b, c, d)$ , but there are additional reachable states. In the NFA, if we are in subset  $\{q'\} \cup S$ , then by input  $d$  we reach  $S$ , since  $d$  is not in the alphabet of  $\mathcal{D}_m(a, b, c)$ . So the determinization also has states  $S$  where  $S \subseteq Q_n$ .

We claim the following states of our DFA for product are reachable and pairwise distinguishable:

- *Restricted case:* All states of the form  $\{q'\} \cup S$  with  $q' \neq (m-1)'$  and  $S \subseteq Q_n$ , and all states of the form  $\{(m-1)', 0\} \cup S$  with  $S \subseteq Q_n \setminus \{0\}$ .
- *Unrestricted case:* All states from the restricted case, and all states  $S$  where  $S \subseteq Q_n$ .

The initial state is  $\{0'\}$ , and we have

$$\{0'\} \xrightarrow{b} \{1'\} \xrightarrow{a^{m-2}} \{(m-1)', 0\} \xrightarrow{a} \{1', 0\} \xrightarrow{b} \{0', 1\}.$$

That is,  $\{0'\} \xrightarrow{ba^{m-1}b} \{0', 1\}$ . For  $0 \leq k \leq n-2$  we have  $\{0', 1\} \xrightarrow{a^k} \{0', 1+k\}$ , and  $\{0', 1\} \xrightarrow{c} \{0', 0\}$ . Thus all states of the form  $\{0', q\}$  for  $q \in Q_n$  are

reachable from  $\{0'\}$ , using the set of words  $\{x, xa, xa^2, \dots, xa^{n-2}, xc\}$  where  $x = ba^{m-1}b$ . Since all of these words are permutations of  $Q_n$  except for  $xc$ , by [14, Theorem 2] all states of the form  $\{0'\} \cup S$  with  $S \subseteq Q_n$  are reachable. To reach  $\{q'\} \cup S$  with  $1 \leq q \leq m-2$ , reach  $\{0'\} \cup Sa^{-q}$  and apply  $a^q$ . To reach  $\{(m-1)', 0\} \cup S$ , reach  $\{(m-2)'\} \cup Sa^{-1}$  and apply  $a$ . In the unrestricted case, we can also reach each state  $S$  from  $\{0'\} \cup S$  by  $d$ .

To see all of these states are distinguishable, consider two distinct states  $X \cup S$  and  $Y \cup T$ . In the restricted case,  $X$  and  $Y$  are singleton subsets of  $Q'_m$ ; in the unrestricted case they may be singletons or empty sets. In both cases  $S$  and  $T$  are arbitrary subsets of  $Q_n$ . If  $S \neq T$ , let  $q$  be an element of the symmetric difference of  $S$  and  $T$ . If  $q \neq 0$  then  $a^{n-1-q}$  distinguishes the states; if  $q = 0$  use  $ba^{n-2}$ . If  $S = T$ , then  $X \neq Y$  and at least one of  $X$  or  $Y$  is non-empty. Assume without loss of generality that  $Y$  is non-empty, say  $Y = \{q'\}$ , and assume  $X$  is either empty or equal to  $\{p'\}$  where  $p < q$ . We consider several cases:

- (i) If  $0 \notin S$ , then  $a^{m-1-q}$  reduces this case to the case where  $S \neq T$ .
- (ii) If  $0 \in S$  and  $1 \notin S$ , and  $\{p', q'\} \neq \{0', 1'\}$ , then  $b$  reduces this to case (i).
- (iii) If  $0, 1 \in S$ , and  $\{p', q'\} \neq \{0', 1'\}$ , then  $c$  reduces this to case (ii).
- (iv) If  $\{p', q'\} = \{0', 1'\}$ , then  $a$  reduces this to case (i), (ii) or (iii).

This shows that in both the restricted and unrestricted cases, all reachable states are pairwise distinguishable.

7. (a) A binary boolean operation is *proper* if it depends on both arguments. For example,  $\cup$ ,  $\cap$ ,  $\setminus$  and  $\oplus$  are proper, whereas the operation  $(L', L) \mapsto L$  is not proper since it depends only on the second argument. Since the transition semigroups of  $\mathcal{D}_m$  and  $\mathcal{D}_n$  are the symmetric groups  $S_m$  and  $S_n$ , for  $m, n \geq 5$ , Theorem 1 of [2] applies, and all proper binary boolean operations have complexity  $mn$ . For  $(m, n) \in \{(3, 4), (4, 3), (4, 4)\}$  we have verified our claim by computation.
- (b) This holds by [2, Theorem 1] as well.
- (c) The upper bounds for unrestricted boolean operations on regular languages were derived in [9]. The proof that the bounds are tight is very similar to the corresponding proof of Theorem 1 in [9]. For  $m, n \geq 3$ , let  $D'_m(a, b, -, c)$  be the dialect of  $\mathcal{D}'_m(a, b, c, d)$  where  $c$  plays the role of  $d$  and the alphabet is restricted to  $\{a, b, c\}$ , and let  $\mathcal{D}_n(b, a, -, d)$  be the dialect of  $\mathcal{D}_n(a, b, c, d)$  in which  $a$  and  $b$  are permuted, and the alphabet is restricted to  $\{a, b, d\}$ ; see Fig. 2.

Next we complete the two DFAs by adding empty states. Restricting both DFAs to the alphabet  $\{a, b\}$ , leads us to the problem of determining the complexity of two DFAs over the same alphabet. In the direct product of the two DFAs, by [2, Theorem 1] and computation for the cases  $(m, n) \in \{(3, 4), (4, 3), (4, 4)\}$ , all  $mn$  states of the form  $\{p', q\}$ ,  $p' \in Q'_m$ ,  $q \in Q_n$ , are reachable and pairwise distinguishable by words in  $\{a, b\}^*$  for all proper boolean operations. As shown in Fig. 3, the remaining states of the direct product are reachable; hence all  $(m+1)(n+1)$  states are reachable.

The proof of distinguishability of pairs of states in the direct product for the union, intersection and symmetric difference is the same as that in [9]. The proof for difference given in [9] is incorrect, but a corrected version is available in [7].  $\square$

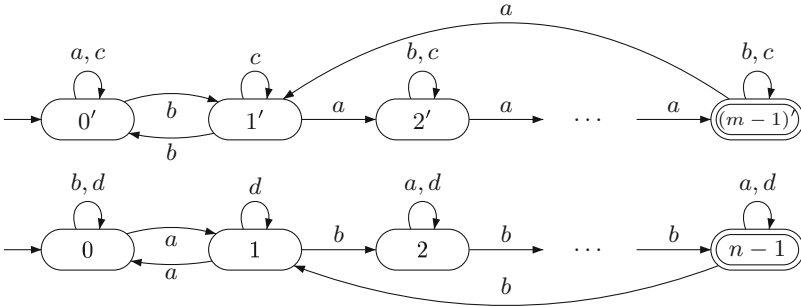


Fig. 2. Witnesses  $D'_m(a, b, -, c)$  and  $D_n(b, a, -, d)$  for boolean operations.

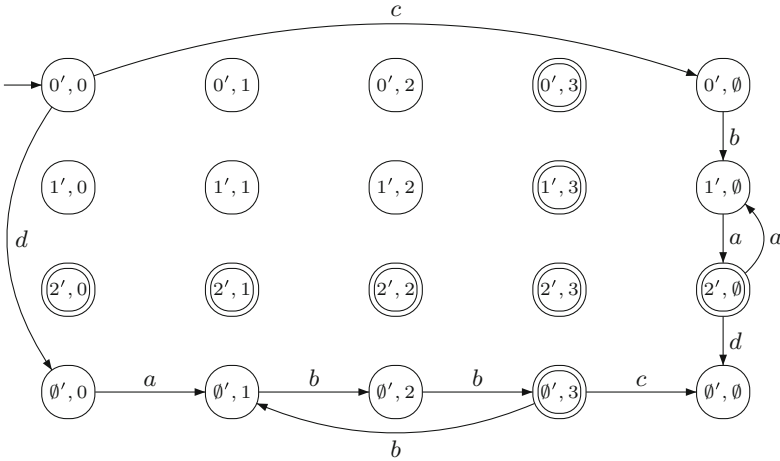


Fig. 3. Direct product for union shown partially.

## 4 Conclusions

We have exhibited a single ternary language stream that is a witness for the maximal state complexities of star and reversal of union-free languages. Together with some dialects it also constitutes a witness for union, intersection, difference, symmetric difference, and product in case the alphabets of the two operands are

the same. As was shown in [17] these bounds are the same as those for regular languages. We prove that our witness also has the largest syntactic semigroup and most complex atoms, and that these complexities are again the same as those for arbitrary regular languages. By adding a fourth input inducing the identity transformation to our witness we obtain witnesses for unrestricted binary operations, where the alphabets of the operands are not the same. The bounds here are again the same as those for regular languages. In summary, this shows that the complexity measures proposed in [4] do not distinguish union-free languages from regular languages.

## References

1. Afonin, S., Golomazov, D.: Minimal union-free decompositions of regular languages. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 83–92. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-00982-2\\_7](https://doi.org/10.1007/978-3-642-00982-2_7)
2. Bell, J., Brzozowski, J., Moreira, N., Reis, R.: Symmetric groups and quotient complexity of boolean operations. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 1–12. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-43951-7\\_1](https://doi.org/10.1007/978-3-662-43951-7_1)
3. Brzozowski, J.A.: Regular expression techniques for sequential circuits. Ph.D. thesis, Princeton University, Princeton, NJ (1962). <http://maveric.uwaterloo.ca/~brzozo/publication.html>
4. Brzozowski, J.A.: In search of the most complex regular languages. *Int. J. Found. Comput. Sci.* **24**(6), 691–708 (2013)
5. Brzozowski, J.A., Davies, S.: Quotient complexities of atoms of regular ideal languages. *Acta Cybern.* **22**, 293–311 (2015)
6. Brzozowski, J.A., Davies, S., Liu, B.Y.V.: Most complex regular ideal languages. *Discrete Math. Theor. Comput. Sci.* **18**(3) (2016). Paper #15
7. Brzozowski, J.A., Sinnamon, C.: Unrestricted state complexity of binary operations on regular and ideal languages (2016). <http://arxiv.org/abs/1609.04439>. Accessed 2017
8. Brzozowski, J.A., Sinnamon, C.: Complexity of right-ideal, prefix-closed, and prefix-free regular languages. *Acta Cybern.* **23**(1), 9–41 (2017)
9. Brzozowski, J.A., Sinnamon, C.: Unrestricted state complexity of binary operations on regular and ideal languages. *J. Autom. Lang. Comb.* **22**(1–3), 29–59 (2017)
10. Brzozowski, J.A., Szykuła, M.: Large aperiodic semigroups. *Int. J. Found. Comput. Sci.* **26**(7), 913–931 (2015)
11. Brzozowski, J.A., Tamm, H.: Complexity of atoms of regular languages. *Int. J. Found. Comput. Sci.* **24**(7), 1009–1027 (2013)
12. Brzozowski, J.A., Tamm, H.: Theory of atomata. *Theor. Comput. Sci.* **539**, 13–27 (2014)
13. Crvenković, S., Dolinka, I., Ésik, Z.: On equations for union-free regular languages. *Inf. Comput.* **164**, 152–172 (2001)
14. Davies, S.: A new technique for reachability of states in concatenation automata. In: Konstantinidis, S., Pighizzini, G. (eds.) DCFS 2018. LNCS, vol. 10316, pp. 75–87. Springer, Cham (2018). Earlier version at <https://arxiv.org/abs/1710.05061>

15. Holzer, M., Kutrib, M.: Structure and complexity of some subregular language families. In: Konstantinidis, S., Moreira, N., Reis, R., Shallit, J. (eds.) *The Role of Theory in Computer Science*, pp. 59–82. World Scientific, Singapore (2017)
16. Iván, S.: Complexity of atoms, combinatorially. *Inf. Process. Lett.* **116**(5), 356–360 (2016)
17. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. *Int. J. Found. Comput. Sci.* **22**(7), 1639–1653 (2011)
18. Jirásková, G., Nagy, B.: On union-free and deterministic union-free languages. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) *TCS 2012. LNCS*, vol. 7604, pp. 179–192. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-33475-7\\_13](https://doi.org/10.1007/978-3-642-33475-7_13)
19. Kutrib, M., Wendlandt, M.: Concatenation-free languages. *Theor. Comput. Sci.* **679**(Suppl. C), 83–94 (2017)
20. Maslov, A.N.: Estimates of the number of states of finite automata. *Dokl. Akad. Nauk SSSR* **194**, 1266–1268 (1970). (in Russian). English translation: *Soviet Math. Dokl.* **11**, 1373–1375 (1970)
21. McNaughton, R., Papert, S.: *Counter-Free Automata*. The MIT Press, Cambridge (1971)
22. Nagy, B.: Union-free regular languages and 1-cycle-free-path-automata. *Publ. Math. Debr.* **68**(1–2), 183–197 (2006)
23. Nagy, B.: On union complexity of regular languages. In: *CINTI 2010*, pp. 177–182. IEEE (2010)
24. Pin, J.E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages*, vol. 1: Word, Language, Grammar, pp. 679–746. Springer, New York (1997). [https://doi.org/10.1007/978-3-642-59136-5\\_10](https://doi.org/10.1007/978-3-642-59136-5_10)
25. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. *Theor. Comput. Sci.* **320**, 315–329 (2004)
26. Schützenberger, M.: On finite monoids having only trivial subgroups. *Inf. Control* **8**, 190–194 (1965)
27. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.* **125**, 315–328 (1994)



# Site-Directed Insertion: Decision Problems, Maximality and Minimality

Da-Jung Cho<sup>1</sup>, Yo-Sub Han<sup>1</sup>, Kai Salomaa<sup>2(✉)</sup>, and Taylor J. Smith<sup>2</sup>

<sup>1</sup> Department of Computer Science, Yonsei University,  
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea  
{dajungcho,emmous}@yonsei.ac.kr

<sup>2</sup> School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada  
{ksalomaa,tsmith}@cs.queensu.ca

**Abstract.** Site-directed insertion is an overlapping insertion operation that can be viewed as analogous to the overlap assembly or chop operations that concatenate strings by overlapping a suffix and a prefix of the argument strings. We consider decision problems and language equations involving site-directed insertion. By relying on the tools provided by *semantic shuffle on trajectories* we show that one variable equations involving site-directed insertion and regular constants can be solved. We consider also maximal and minimal variants of the site-directed insertion operation.

## 1 Introduction

Site-directed mutagenesis is one of the most important techniques for generating mutations on specific sites of DNA using polymerase chain reaction (PCR) based methods [18]. The algorithmic applications of mutagenesis have been considered e.g. by Franco and Manca [10]. Contextual insertion/deletion systems in the study of molecular computing have been used, e.g. by Kari and Thierrin [16], Daley et al. [4] and Enaganti et al. [8].

Site-directed insertion (SDI) of a string  $y$  into a string  $x$  involves matching an outfix of  $y$  with a substring of  $x$  and inserting the “middle part” of  $y$  not belonging to the outfix into  $x$ . Site-directed insertion has earlier been considered under the name *outfix-guided insertion* [2]. The operation is an overlapping variant of the insertion operation in the same sense as the overlap assembly, a.k.a. chop operation, is a variant of string concatenation [3, 9, 12, 13].

The maximal (respectively, minimal) SDI of a string  $y$  into a string  $x$  requires that, at the chosen location of  $x$ , the operation matches a maximal

---

Cho and Han were supported by the Basic Science Research Program through NRF (2015R1D1A1A01060097) and the International Research & Development Program of NRF (2017K1A3A1A12024971). Salomaa and Smith were supported by Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

(respectively, minimal) outfix of  $y$  with a substring of  $x$ . This is analogous to the maximal and minimal chop operations studied by Holzer et al. [13].

Site-directed insertion can be represented as a *semantic shuffle on trajectories* (SST). Shuffle on trajectories was introduced by Mateescu et al. [17] and the extension to SST is due to Domaratzki [5]. Further extensions of the shuffle-on-trajectories operation have been studied by Domaratzki et al. [7].

Here we study decision problems and language equations involving site-directed insertion and its maximal and minimal variants. The representation of SDI as a semantic shuffle on a regular set of trajectories gives regularity preserving left- and right-inverses of the operation. By the general results of Kari [15] on the decidability of equations, translated for SST by Domaratzki [5], this makes it possible to decide linear equations involving SDI where the constants are regular languages.

The maximal and minimal SDI operations do not, in general, preserve regularity. This means that the operations cannot be represented by SST [5] (on a regular set of trajectories) and the above tools are not available to deal with language equations. We show that for maximal and minimal SDI certain independence properties related to coding applications [14] can be decided in a polynomial time. The decidability of whether a regular language is closed under max/min SDI remains open.

In the last section we give a tight bound for the nondeterministic state complexity of alphabetic SDI, where the matching outfix must consist of a prefix and suffix of length exactly one. An upper bound for the state complexity of the general site-directed insertion is known but it remains open whether the bound is optimal.

## 2 Preliminaries

We assume the reader to be familiar with the basics of finite automata, regular languages and context-free languages [19]. Here we briefly recall some notation.

Let  $\Sigma$  be an alphabet and  $w \in \Sigma^*$ . If we can write  $w = xyz$  we say that the pair  $(x, z)$  is an *outfix* of  $w$ . The outfix  $(x, z)$  is a *nontrivial outfix* of  $w$  if  $x \neq \varepsilon$  and  $z \neq \varepsilon$ . For  $L \subseteq \Sigma^*$ ,  $\bar{L} = \Sigma^* - L$  is the complement of  $L$ .

A *nondeterministic finite automaton* (NFA) is a tuple  $A = (\Sigma, Q, \delta, q_0, F)$  where  $\Sigma$  is the input alphabet,  $Q$  is the finite set of states,  $\delta: Q \times \Sigma \rightarrow 2^Q$  is the transition function,  $q_0 \in Q$  is the initial state and  $F \subseteq Q$  is the set of final states. In the usual way  $\delta$  is extended as a function  $Q \times \Sigma^* \rightarrow 2^Q$  and the *language accepted by  $A$*  is  $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$ . The automaton  $A$  is a *deterministic finite automaton* (DFA) if  $|\delta(q, a)| \leq 1$  for all  $q \in Q$  and  $a \in \Sigma$ . It is well known that the deterministic and nondeterministic finite automata recognize the class of *regular languages*.

The so called fooling set lemma gives a technique for establishing lower bounds for the size of NFAs:



**Lemma 1 (Birget 1992 [1]).** *Let  $L \subseteq \Sigma^*$  be a regular language. Suppose that there exists a set  $P = \{(x_i, w_i) \mid 1 \leq i \leq n\}$  of pairs of strings such that: (i)  $x_i w_i \in L$  for  $1 \leq i \leq n$ , and, (ii) if  $i \neq j$ , then  $x_i w_j \notin L$  or  $x_j w_i \notin L$  for  $1 \leq i, j \leq n$ . Then, any minimal NFA for  $L$  has at least  $n$  states.*

Finally we recall some notions concerning operations on languages and language equations. Let  $\odot$  be a binary operation on languages, and  $L, R$  are languages over an alphabet  $\Sigma$ .

- (i) The language  $L$  is *closed under*  $\odot$  if  $L \odot L \subseteq L$ .
- (ii) The language  $L$  is  $\odot$ -*free* with respect to  $R$  if  $L \odot R = \emptyset$ .
- (iii) The language  $L$  is  $\odot$ -*independent* with respect to  $R$  if  $(L \odot \Sigma^+) \cap R = \emptyset$ .
- (iv) A *solution* for an equation  $X \odot L = R$  (respectively,  $L \odot X = R$ ) is a language  $S \subseteq \Sigma^*$  such that  $S \odot L = R$  (respectively,  $L \odot S = R$ ).

The  $\odot$ -freeness and independence properties can be related to coding applications, where it might be desirable that we cannot produce new strings by applying an operation, such as site-directed insertion, to strings of the language. Domaratzki [6] defines trajectory-based codes analogously with (iii). As we will see, languages that are site-directed insertion independent with respect to themselves have a definition closely resembling outfix-codes of index one [14].

### 3 Site-Directed Insertion

The site-directed insertion is a partially overlapping insertion operation analogously as the overlap-assembly (or self-assembly) [3, 9] models an overlapping concatenation of strings. The overlapping concatenation operation is also called the chop operation [13].

The *site-directed insertion* (SDI) of a string  $y$  into a string  $x$  is defined as

$$x \stackrel{\text{sdi}}{\leftarrow} y = \{x_1 u z v x_2 \mid x = x_1 u v x_2, y = u z v, u \neq \epsilon, v \neq \epsilon\}.$$

The above definition requires that the pair  $(u, v)$  is a nontrivial outfix of the string  $y$  and  $uv$  is a substring of  $x$ . If  $y = u z v$  is inserted into  $x$  by matching the outfix with a substring  $uv$  of  $x$ , we say that  $(u, v)$  is an *insertion guide* for the operation. Note that a previous paper [2] uses the name “outfix-guided insertion” for the same operation.

The site-directed insertion operation is extended in the usual way for languages by setting

$$L_1 \stackrel{\text{sdi}}{\leftarrow} L_2 = \bigcup_{w_i \in L_i, i=1,2} w_1 \stackrel{\text{sdi}}{\leftarrow} w_2.$$

We recall that regular languages are closed under site-directed insertion.

**Proposition 1** ([2]). *If  $A$  and  $B$  are NFAs with  $m$  and  $n$  states, respectively, the language  $L(A) \stackrel{\text{sdi}}{\leftarrow} L(B)$  has an NFA with  $3mn + 2m$  states.*

A simpler form of the overlap-assembly operation requires the overlapping part of the strings to consist of a single letter. This operation is called “chop” by Holzer and Jacobi [12] but the later definition of the chop-operation [13] coincides with general overlap-assembly [9]. Analogously we define *alphabetic site-directed insertion* by requiring that the overlapping prefix and suffix of the inserted string each consist of a single letter.

The *alphabetic site-directed insertion* of a string  $y$  into a string  $x$  is

$$x \stackrel{\text{a-sdi}}{\leftarrow} y = \{x_1azbx_2 \mid x = x_1abx_2, y = azb, a, b \in \Sigma, x_1, x_2, z \in \Sigma^*\}.$$

Note that the alphabetic site-directed insertion will have different closure properties than the standard site-directed insertion. For example, it is not difficult to see that the context-free languages are closed under alphabetic site-directed insertion, while the context-free languages are not closed under general site-directed insertion [2].

### 3.1 Decision Problems

For a regular language  $L$ , it is decidable whether  $L$  is closed under site-directed insertion. The algorithm relies on the construction of Proposition 1 and operates in polynomial time when  $L$  is specified by a DFA [2]. Deciding whether a context-free language is closed under site-directed insertion is undecidable [2].

A language  $L$  is  $\stackrel{\text{sdi}}{\leftarrow}$ -free, or SDI-free, with respect to  $R$  if no string of  $R$  can be site-directed inserted into a string of  $L$ , that is, if  $L \stackrel{\text{sdi}}{\leftarrow} R = \emptyset$ . The language  $L$  is SDI-independent with respect to  $R$  if site-directed inserting a non-empty string into  $L$  cannot produce a string of  $R$ . Note that  $L$  being SDI-independent with respect to itself resembles the notion of  $L$  being an outfix-code of index one [14] with the difference that we require the outfix to be nontrivial. For example,  $\{ab, b\}$  is SDI-independent but it is not an outfix-code of index one.

**Theorem 1.** *For NFAs  $A$  and  $B$  we can decide in polynomial time whether*

- (i)  $L(A)$  is SDI-free (or SDI-independent) with respect to  $L(B)$ .
- (ii)  $L(A)$  is alphabetic SDI-free (or alphabetic SDI-independent) with respect to  $L(B)$ .

For context-free languages deciding SDI-freeness and SDI-independence is undecidable.

**Proposition 2.** *For context-free languages  $L_1$  and  $L_2$  it is undecidable whether*

- (i)  $L_1$  is SDI-free with respect to  $L_2$ ,
- (ii)  $L_1$  is SDI-independent with respect to  $L_2$ .

For dealing with language equations we express the site-directed insertion operation as a *semantic shuffle on a set of trajectories* (SST) due to Domaratzki [5]. The semantic shuffle extends the (syntactic) *shuffle on trajectories* originally defined by Mateescu et al. [17]. We use a simplified definition of SST that does not allow *content restriction* [5].

The *trajectory alphabet* is  $\Gamma = \{0, 1, \sigma\}$  and a trajectory is a string over  $\Gamma$ . The semantic shuffle of  $x, y \in \Sigma^*$  on a trajectory  $t \in \Gamma^*$ , denoted by  $x \uparrow_t y$ , is defined as follows.

If  $x = y = \varepsilon$ , then  $x \uparrow_t y = \varepsilon$  if  $t = \varepsilon$  and is undefined otherwise. If  $x = ax'$ ,  $a \in \Sigma$ ,  $y = \varepsilon$  and  $t = ct'$ ,  $c \in \Gamma$ , then

$$x \uparrow_t \varepsilon = \begin{cases} a(x' \uparrow_{t'} \varepsilon) & \text{if } c = 0, \\ \emptyset, & \text{otherwise.} \end{cases}$$

If  $x = \varepsilon$ ,  $y = by'$ ,  $b \in \Sigma$ , and  $t = ct'$ ,  $c \in \Gamma$ , then

$$\varepsilon \uparrow_t y = \begin{cases} b(\varepsilon \uparrow_{t'} y') & \text{if } c = 1, \\ \emptyset, & \text{otherwise.} \end{cases}$$

In the case where all the strings are nonempty, for  $x = ax'$ ,  $y = by'$ ,  $a, b \in \Sigma$ , and  $t = ct'$ ,  $c \in \Gamma$ , we define

$$x \uparrow_t y = \begin{cases} a(x' \uparrow_{t'} y) & \text{if } c = 0, \\ b(x \uparrow_{t'} y') & \text{if } c = 1, \\ a(x' \uparrow_{t'} y') & \text{if } a = b \text{ and } c = \sigma, \\ \emptyset, & \text{otherwise.} \end{cases}$$

Intuitively, the trajectory  $t$  is a sequence of instructions that guide the shuffle of strings  $x$  and  $y$ : 0 selects the next symbol of  $x$ , 1 the next symbol of  $y$  (these are as in the original definition of syntactic shuffle [17]) and  $\sigma$  represents synchronized insertion where the next symbols of the argument strings must coincide.

For  $x, y \in \Sigma^*$  and  $t \in \Gamma^*$ ,  $x \uparrow_t y$  either consists of a unique string or is undefined. For  $T \subseteq \Gamma^*$ ,  $x \uparrow_T y = \bigcup_{t \in T} x \uparrow_t y$  and the operation is extended in the natural way for languages over  $\Sigma$ .

Directly from the definition it follows that the SDI and alphabetic SDI operations can be represented as semantic shuffle on a regular set of trajectories.

**Proposition 3.** *Let  $T_{\text{sdi}} = 0^* \sigma^+ 1^* \sigma^+ 0^*$  and  $T_{\text{a-sdi}} = 0^* \sigma 1^* \sigma 0^*$ . Then, for any languages  $L_1$  and  $L_2$ ,*

$$L_1 \stackrel{\text{sdi}}{\leftarrow} L_2 = L_1 \uparrow_{T_{\text{sdi}}} L_2, \text{ and, } L_1 \stackrel{\text{a-sdi}}{\leftarrow} L_2 = L_1 \uparrow_{T_{\text{a-sdi}}} L_2.$$

Now using the strong decidability results of Domaratzki [5] we can effectively decide linear language equations involving site-directed insertion where the constants are regular languages. The representation of SDI using SST guarantees the existence of regularity preserving left- and right-inverse of the operation. This

makes it possible to use the results of Kari [15] to decide existence of solutions to linear equations where the constants are regular languages. The maximal solutions to the equations are represented using *semantic deletion along trajectories* [5]. For the deletion operation we consider a trajectory alphabet  $\Delta = \{i, d, \sigma\}$ . Intuitively, a trajectory  $t \in \Delta^*$  guides the deletion of a string  $y$  from  $x$  as follows: a symbol  $i$  (insertion) indicates that we output the next symbol of  $x$ , a symbol  $d$  (deletion) indicates that the next symbol of  $y$  must match the next symbol of  $x$  and nothing is produced in the output and a symbol  $\sigma$  (synchronization) indicates that the next symbols of  $x$  and  $y$  must match and this symbol is placed in the output. The result of deleting  $y$  from  $x$  along trajectory  $t$  is denoted  $x \rightsquigarrow_t y$  and the operation is extended in the natural way for sets of trajectories and for languages.

We can express the left- and right-inverse (as defined in [5, 15]) of SDI using semantic deletion along trajectories, and these relations are used to express solutions for linear language equations. Given a binary operation  $\diamond$  on strings, let  $\diamond^{\text{rev}}$  be the operation defined by  $x \diamond^{\text{rev}} y = y \diamond x$  for all  $x, y \in \Sigma^*$ . Using Theorems 6.4 and 6.5 of [5] we obtain:

**Theorem 2.** *Let  $L, R \subseteq \Sigma^*$  be regular languages. Then for each of the following equations it is decidable whether a solution exists: (a)  $X \stackrel{\text{sdi}}{\leftarrow} L = R$ , (b)  $L \stackrel{\text{sdi}}{\leftarrow} X = R$ , (c)  $X \stackrel{\text{a-sdi}}{\leftarrow} L = R$ , (d)  $L \stackrel{\text{a-sdi}}{\leftarrow} X = R$ .*

*Define  $T_1 = i^* \sigma^+ d^* \sigma^+ i^*$ ,  $T_1^a = i^* \sigma d^* \sigma i^*$ ,  $T_2 = d^* \sigma^+ i^* \sigma^+ d^*$ , and  $T_2^a = d^* \sigma i^* \sigma d^*$ . If a solution exists, a superset of all solutions is, respectively, for the different cases: (a)  $S_a = \overline{R \rightsquigarrow_{T_1} L}$ , (b)  $S_b = \overline{L(\rightsquigarrow_{T_2})^{\text{rev}} R}$ , (c)  $S_c = \overline{R \rightsquigarrow_{T_1^a} L}$ , (d)  $S_d = \overline{L(\rightsquigarrow_{T_2^a})^{\text{rev}} R}$ .*

The above result does not give a polynomial time decision algorithm, even in the case where the languages  $L$  and  $R$  are given by DFA's. Semantic shuffle on and deletion along regular sets of trajectories preserve regularity but the operations are inherently nondeterministic and complementation blows up the size of an NFA. Note that deleting an individual string  $y$  from a string  $x$  along trajectory  $t$  is deterministic, but the automaton construction for the result of the operation on two DFA languages is nondeterministic. An explicit construction of an NFA for the syntactic shuffle of two regular languages is given in [17].

The known trajectory based methods for two variable equations [5] do not allow the trajectories to use the synchronizing symbol  $\sigma$  that is needed to represent the overlap of SDI. However, if we are just interested to know whether a solution exists (as opposed to finding maximal solutions), it is easy to verify that an equation  $X_1 \stackrel{\text{sdi}}{\leftarrow} X_2 = R$  has a solution if and only if all strings of  $R$  have length at least two.

## 4 Maximal and Minimal Site-Directed Insertion

Holzer et al. [13] define two deterministic variants of the chop operation. The max-chop (respectively, min-chop) of strings  $x$  and  $y$  chooses the non-empty suffix of  $x$  overlapping with  $y$  to be as long (respectively, as short) as possible.

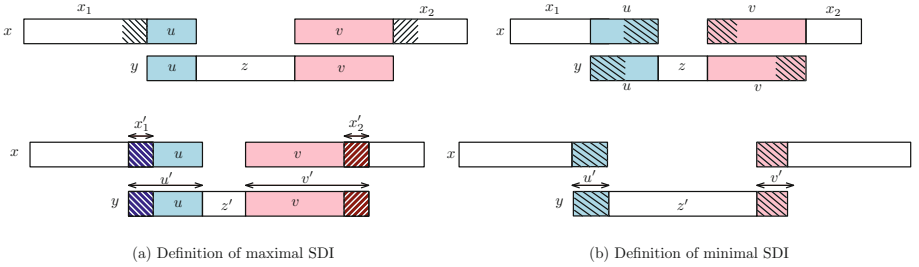
By a *maximal site-directed insertion* of string  $y$  into a string  $x$  we mean, roughly speaking, an insertion where neither the overlapping prefix nor the overlapping suffix can be properly extended. The operation is not deterministic because  $y$  could be inserted in different positions in  $x$ . At a specific position in  $x$ , a string  $y$  can be maximally (respectively, minimally) inserted in at most one way.

Formally, the *maximal site-directed insertion* (max-SDI) of a string  $y$  into string  $x$  is defined as follows:

$$x \xrightarrow{\max\text{-sdi}} y = \{x_1uzvx_2 \mid x = x_1uvx_2, y = uzv, u \neq \epsilon, v \neq \epsilon, \text{ and} \\ \text{there exist no suffix } x'_1 \text{ of } x_1 \text{ and prefix } x'_2 \text{ of } x_2 \text{ such that} \\ x'_1x'_2 \neq \epsilon \text{ and } y = x'_1uz'vx'_2, z' \in \Sigma^*\}$$

Equivalently the maximal SDI of  $x$  and  $y$  is

$$x \xrightarrow{\max\text{-sdi}} y = \{x_1uzvx_2 \mid x = x_1uvx_2, y = uzv, u \neq \epsilon \neq v, \text{ no suffix of } x_1u \\ \text{of length greater than } |u| \text{ is a prefix of } uz \text{ and no prefix} \\ \text{of } vx_2 \text{ of length greater than } |v| \text{ is a suffix of } zv\}.$$



**Fig. 1.** Insertion of  $y$  into  $x$  depicted at top left is not maximal when  $x$  and  $y$  have decompositions as depicted at bottom left.

In particular, if  $x$  and  $y$  are unary strings with  $|x| \geq |y| \geq 2$ , then  $x \xrightarrow{\max\text{-sdi}} y = x$  because the maximal overlapping outfix consists always of the entire string  $y$ . If  $|x| \geq 2$  and  $|y| > |x|$ , then  $x \xrightarrow{\max\text{-sdi}} y = y$ . If  $|x| < 2$  or  $|y| < 2$ , the operation is undefined.

*Example 1.* Consider alphabet  $\Sigma = \{a, b, c\}$ . Now

$$ababab \xrightarrow{\max\text{-sdi}} acbab = \{acbabab, abacbab, ababacbab\}$$

For example also the string  $abacbabab$  is obtained by site-directed inserting  $y = acbab$  into  $x = ababab$ . In this operation the prefix  $a$  of  $y$  is matched with the 3rd symbol of  $x$  and the suffix  $b$  of  $y$  is matched with the 4th symbol of  $x$ . However, this operation does not satisfy the maximality condition because after the 3rd symbol of  $x$  we can match a longer suffix of  $y$ .

The *minimal site-directed insertion* (min-SDI) operation is defined as follows:

$$x \stackrel{\text{min-sdi}}{\leftarrow} y = \{x_1uzvx_2 \mid x = x_1uvx_2, y = uzv, u \neq \epsilon, v \neq \epsilon, \\ \text{no proper nonempty suffix of } u \text{ is a prefix of } u, \text{ and} \\ \text{no proper nonempty prefix of } v \text{ is a suffix of } v\}.$$

Note that in the definition of min-SDI,  $u$  and  $v$  are *unbordered words*. Figure 1(b) illustrates the definition of minimal SDI. The alphabetic SDI can be viewed as an “extreme” case of minimal SDI: if the first and last letter of  $y$  coincide with a substring of  $x$  of length two, then the alphabetic and minimal site-directed insertion of  $y$  in that position coincide.

If  $x$  and  $y$  are unary strings with  $|x|, |y| \geq 2$ , then  $x \stackrel{\text{min-sdi}}{\leftarrow} y$  is the unary string of length  $|y| + |x| - 2$  and the operation is undefined for  $|x| < 2$  or  $|y| < 2$ .

Note that while the maximal or minimal SDI is considerably more restricted than the unrestricted SDI operation, if a string  $y$  can be site-directed inserted to a string  $x$ , it can be also maximally or minimally inserted at the same position. The result of an alphabetic insertion is always a minimal insertion. These observations are formalized in the next lemma.

**Lemma 2.** *Let  $x, y \in \Sigma^*$ .*

- (i)  $x \stackrel{\text{max-sdi}}{\leftarrow} y \subseteq x \stackrel{\text{sdi}}{\leftarrow} y$  and  $x \stackrel{\text{a-sdi}}{\leftarrow} y \subseteq x \stackrel{\text{min-sdi}}{\leftarrow} y \subseteq x \stackrel{\text{sdi}}{\leftarrow} y$ .
- (ii)  $x \stackrel{\text{sdi}}{\leftarrow} y \neq \emptyset$  iff  $x \stackrel{\text{max-sdi}}{\leftarrow} y \neq \emptyset$  iff  $x \stackrel{\text{min-sdi}}{\leftarrow} y \neq \emptyset$ .
- (iii) It is possible that  $x \stackrel{\text{min-sdi}}{\leftarrow} y \neq \emptyset$  and  $x \stackrel{\text{a-sdi}}{\leftarrow} y = \emptyset$ .

Since the max-chop and min-chop operations do not preserve regularity [13], it can be expected that the same holds for maximal and minimal SDI. The proof of the following proposition is inspired by Theorem 3 of [13].

**Proposition 4.** *The maximal and minimal site-directed insertion do not preserve regularity.*

**Proof.** Let  $\Sigma = \{a, b, \$, \%\}$  and choose

$$L_1 = ba^+ba^+\$, \quad L_2 = ba^+ba^+\%\$$$

We claim that

$$(L_1 \stackrel{\text{max-sdi}}{\leftarrow} L_2) \cap (ba^+)^3\%\$ = \{ba^m ba^n ba^k\%\$ \mid m \neq n \text{ or } k < n, m, n, k \geq 1\}$$

We denote the right side of the equation by  $L_{\text{result}}$  which is clearly nonregular. Since the strings of  $L_2$  contain the marker  $\%$  that does not occur in strings of  $L_1$ , when inserting a string  $y \in L_2$  into a string of  $L_1$  the overlapping suffix of  $y$  must consist exactly of the last symbol  $\$$ . Consider  $x = ba^i ba^j \$ \in L_1$  and  $y = ba^r ba^s \% \$ \in L_2$ . In order for the resulting string to have three symbols  $b$ , a prefix of  $ba^r$  must overlap with  $ba^j$ , that is,  $j \leq r$ . In order for the overlap to be

maximal we must have  $r \neq i$  or  $s < j$ . These relations guarantee that the unique string in  $x \stackrel{\max\text{-sdi}}{\leftarrow} y$  is in  $L_{\text{result}}$ .

For the converse inclusion we note that, for  $m \neq n$  or  $k < n$ ,

$$ba^m ba^n ba^k \% \$ \in ba^m ba^n \$ \stackrel{\max\text{-sdi}}{\leftarrow} ba^n ba^k \% \$.$$

For non-closure under min-SDI we claim that

$$(L_1 \stackrel{\min\text{-sdi}}{\leftarrow} L_2) \cap (ba^+)^2 \% \$ = \{ba^m ba^n \% \$ \mid n > m \geq 1\} =^{\text{def}} L'_{\text{result}}.$$

Consider  $x = ba^i ba^j \% \$ \in L_1$  and  $y = ba^r ba^s \% \$ \in L_2$ . In order for the result of site-directed insertion of  $y$  into  $x$  to have two  $b$ 's,  $ba^i ba^j$  must be a prefix of  $ba^r ba^s$ , that is,  $i = r$  and  $j \leq s$ . For the site-directed insertion to be minimal, no proper non-empty prefix of  $ba^i ba^j$  can be its suffix, that is  $i < j$ . These relations guarantee that the minimal SDI of  $x$  and  $y$  is in  $L'_{\text{result}}$ :

Conversely, for  $n > m$ ,  $ba^m ba^n \% \$ \in ba^m ba^n \$ \stackrel{\min\text{-sdi}}{\leftarrow} ba^m ba^n \% \$$ .  $\square$

In fact, extending the max-chop and min-chop constructions from Theorem 3 of [13] it would be possible to show that there exist regular languages  $L_1$  and  $L_2$  such that  $L_1 \stackrel{\max\text{-sdi}}{\leftarrow} L_2$  (or  $L_1 \stackrel{\min\text{-sdi}}{\leftarrow} L_2$ ) is not context-free. The maximal or minimal site-directed insertion of a finite language into a regular language (and vice versa) is regular.

**Proposition 5.** *Let  $R$  be a regular language and  $L$  a finite language. Then the languages  $R \stackrel{\max\text{-sdi}}{\leftarrow} L$ ,  $R \stackrel{\min\text{-sdi}}{\leftarrow} L$ ,  $L \stackrel{\max\text{-sdi}}{\leftarrow} R$ , and  $L \stackrel{\min\text{-sdi}}{\leftarrow} R$  are regular.*

**Proof.** We show that  $R \stackrel{\max\text{-sdi}}{\leftarrow} L$  is regular. The other cases are very similar.

Since

$$R \stackrel{\max\text{-sdi}}{\leftarrow} L = \bigcup_{y \in L} R \stackrel{\max\text{-sdi}}{\leftarrow} y$$

and regular languages are closed under finite union, it is sufficient to consider the case where  $L$  consists of one string  $y$ .

Let  $A$  be an NFA for  $R$  and  $y \in \Sigma^*$ . We outline how an NFA  $B$  can recognize  $L(A) \stackrel{\max\text{-sdi}}{\leftarrow} y$ . On an input  $w$ ,  $B$  nondeterministically guesses a decomposition  $w = x_1 y_1 y_2 y_3 x_2$  where  $x_1 y_1 y_3 x_2 \in L(A)$ ,  $y_1 y_2 y_3 = y$  and  $y_1, y_3 \neq \varepsilon$ . When reading the prefix  $x_1 y_1$ ,  $B$  simulates a computation of  $A$  ending in a state  $q$ , then skips the substring  $y_2$ , and continues simulation of  $A$  from state  $q$  on the suffix  $y_3 x_2$ .

The above checks that the input is in  $L(A) \stackrel{\text{sdi}}{\leftarrow} y$  and, additionally,  $B$  needs to verify that the insertion is maximal. This is possible because  $B$  is looking for maximal insertions of the one fixed string  $y$ .

(i) When processing the prefix  $x_1$ , the state of  $B$  remembers the last  $|y| - 1$  symbols scanned. When the computation nondeterministically guesses the

substrings  $y_1, y_2, y_3$ , it can then check that for no nonempty suffix  $x'_1$  of  $x_1$ ,  $x'_1 y_1$  is a prefix of  $y_1 y_2$ . If this condition does not hold, the corresponding transition is undefined.

(ii) Similarly, when processing the (nondeterministically selected) suffix  $x_2$  of the input,  $B$  remembers the first  $|y| - 1$  symbols and is able to check that for no nonempty prefix  $x'_2$  of  $x_2$ ,  $y_3 x'_2$  is a suffix of  $y_2 y_3$ .

If the checks in both (i) and (ii) are successful and at the end the simulation of  $A$  ends with a final state, this means that the decomposition  $x_1 y_1 y_2 y_3 x_2$  gives a maximal site-directed insertion of  $y$  into a string of  $L(A)$ .  $\square$

#### 4.1 Decision Problems for Maximal/Minimal SDI

From Proposition 4 we know that the maximal or minimal SDI of regular languages need not be regular. However, for regular languages  $L_1$  and  $L_2$  we can decide membership in  $L_1 \stackrel{\text{max-sdi}}{\leftarrow} L_2$  (or  $L_1 \stackrel{\text{min-sdi}}{\leftarrow} L_2$ ) in polynomial time.

**Lemma 3.** *For DFAs  $A$  and  $B$  and  $w \in \Sigma^*$  we can decide in time  $O(n^6)$  whether  $w \in L(A) \stackrel{\text{max-sdi}}{\leftarrow} L(B)$ , or whether  $w \in L(A) \stackrel{\text{min-sdi}}{\leftarrow} L(B)$ .*

As we have seen, the maximal and minimal SDI operations are often more difficult to handle than the unrestricted SDI. Using Lemma 2 (ii) we note that deciding maximal (or minimal) SDI-freeness is the same as deciding SDI-freeness and by Theorem 1 we have:

**Corollary 1.** *For NFAs  $A$  and  $B$  we can decide in polynomial time whether or not  $L(A)$  is maximal SDI-free (respectively, minimal SDI-free) with respect to  $L(B)$ .*

Also, deciding whether regular languages are max-SDI (or min-SDI) independent can be done in polynomial time.

**Theorem 3.** *For NFAs  $A$  and  $B$ , we can decide in polynomial time whether or not  $L(A)$  is maximal SDI-independent (respectively, minimal SDI-independent) with respect to  $L(B)$ .*

**Proof.** Let  $\Sigma$  be the underlying alphabet of  $A$  and  $B$ . We verify that  $L(A) \stackrel{\text{max-sdi}}{\leftarrow} \Sigma^+ = L(A) \stackrel{\text{sdi}}{\leftarrow} \Sigma^+$ . The inclusion from left to right holds by Lemma 2 (i). Conversely, suppose  $w \in L(A) \stackrel{\text{sdi}}{\leftarrow} y_1 y_2 y_3$ , where  $w = x_1 y_1 y_2 y_3 x_2$ ,  $y_1, y_3 \neq \varepsilon$ ,  $x_1 y_1 y_3 x_2 \in L(A)$ . Then  $w \in L(A) \stackrel{\text{max-sdi}}{\leftarrow} x_1 y_1 y_2 y_3 x_2$ , where the latter insertion uses the outfix  $(x_1 y_1, y_3 x_2)$  as insertion guide. The insertion is maximal because the outfix cannot be expanded. In the same way we see that  $L(A) \stackrel{\text{min-sdi}}{\leftarrow} \Sigma^+ = L(A) \stackrel{\text{sdi}}{\leftarrow} \Sigma^+$ . Now the claim follows by Theorem 1.  $\square$



Since the max-SDI and min-SDI operations do not preserve regularity there is no straightforward algorithm to decide whether a regular language is closed under maximal SDI or under minimal SDI. We conjecture that the problem is decidable.

*Problem 1.* Is there an algorithm that for a given regular language  $L$  decides whether or not  $L \stackrel{\text{max-sdi}}{\leftarrow} L \subseteq L$  (respectively,  $L \stackrel{\text{min-sdi}}{\leftarrow} L \subseteq L$ )?

Using Proposition 5 we can decide closure of a regular language under max/min-SDI with a finite language.

**Corollary 2.** *Given a regular language  $R$  and a finite language  $F$  we can decide whether or not (i)  $R \stackrel{\text{max-sdi}}{\leftarrow} F \subseteq R$ , (ii)  $R \stackrel{\text{min-sdi}}{\leftarrow} F \subseteq R$ . If  $R$  is specified by a DFA and the length of the longest string in  $F$  is bounded by a constant, the algorithm works in polynomial time.*

**Proof.** By Proposition 5 the languages  $R_{\text{max}} = R \stackrel{\text{max-sdi}}{\leftarrow} F$  and  $R_{\text{min}} = R \stackrel{\text{min-sdi}}{\leftarrow} F$  are effectively regular.

Suppose  $R = L(A)$  where  $A$  is a DFA with  $m$  states and underlying alphabet  $\Sigma$  and the length of the longest string in  $F$  is  $c_F$ . The NFA  $B$  constructed in the proof of Proposition 5 for  $R_{\text{max}}$  (or  $R_{\text{min}}$ ) has  $O(m \cdot |\Sigma|^{c_F})$  states. Recall that the NFA stores in the state a sequence of symbols having length of the inserted string. Strictly speaking, the proof of Proposition 5 assumes that  $F$  consists of a single string, but a similar construction works for a finite language. When  $c_F$  is a constant, the size of  $B$  is polynomial in  $m$  and we can decide in polynomial time whether or not  $L(B) \cap \overline{L(A)} = \emptyset$ .  $\square$

The max-SDI and min-SDI operations do not preserve regularity and, consequently, they cannot be represented using semantic shuffle on trajectories. Thus, the tools developed in Sect.6 of [5] to deal with language equations are not available and it remains open whether we can solve language equations involving max-SDI or min-SDI.

*Problem 2.* Let  $L$  and  $R$  be regular languages. Is it decidable whether the equation  $X \stackrel{\text{max-sdi}}{\leftarrow} L = R$  (respectively,  $L \stackrel{\text{max-sdi}}{\leftarrow} X = R$ ,  $X \stackrel{\text{min-sdi}}{\leftarrow} L = R$ ,  $L \stackrel{\text{min-sdi}}{\leftarrow} X = R$ ) has a solution?

## 5 Nondeterministic State Complexity

The site-directed insertion (SDI) operation preserves regularity [2] (Proposition 1 above) and the construction can be modified to show that also alphabetic SDI preserves regularity. To conclude, we consider the nondeterministic state complexity of these operations.

**Lemma 4.** *For NFAs  $M$  and  $N$  having, respectively,  $m$  and  $n$  states, the language  $L(M) \stackrel{\text{a-sdi}}{\leftarrow} L(N)$  can be recognized by an NFA with  $mn + 2m$  states.*

The upper bound is the same as the bound for the nondeterministic state complexity of ordinary insertion [11], however, the construction used for Lemma 4 is not the same. Using Lemma 1 (the fooling set lemma [1]) we can establish a matching lower bound.

**Lemma 5.** *For  $m, n \in \mathbb{N}$ , there exist regular languages  $L_1$  and  $L_2$  over a binary alphabet having NFAs with  $m$  and  $n$  states, respectively, such that any NFA for  $L_1 \stackrel{\text{a-sdi}}{\leftarrow} L_2$  needs at least  $mn + 2m$  states.*

The above lemmas establish the precise nondeterministic state complexity of alphabetic SDI.

**Corollary 3.** *The worst case nondeterministic state complexity of the alphabetic site-directed insertion of an  $n$ -state NFA language into an  $m$ -state NFA language is  $mn + 2m$ . The lower bound can be reached by languages over a binary alphabet.*

It is less obvious what is the precise nondeterministic state complexity of the general SDI. If  $A$  has  $m$  states and  $B$  has  $n$  states, Proposition 1 gives an upper bound  $3mn + 2m$  for the nondeterministic state complexity of  $L(A) \stackrel{\text{sdi}}{\leftarrow} L(B)$ . Likely the bound cannot be improved but we do not have a proof for the lower bound.

*Problem 3.* What is the nondeterministic state complexity of site-directed insertion?

## References

1. Birget, J.C.: Intersection and union of regular languages and state complexity. *Inf. Process. Lett.* **43**, 185–190 (1992)
2. Cho, D.-J., Han, Y.-S., Ng, T., Salomaa, K.: Outfix-guided insertion. *Theor. Comput. Sci.* **701**, 70–84 (2017)
3. Csuhaj-Varju, E., Petre, I., Vaszil, G.: Self-assembly of string and languages. *Theor. Comput. Sci.* **374**, 74–81 (2007)
4. Daley, M., Kari, L., Gloor, G., Siromoney, R.: Circular contextual insertions/deletions with applications to biomolecular computation. In: *String Processing and Information Retrieval Symposium*, pp. 47–54 (1999)
5. Domaratzki, M.: Semantic shuffle on and deletion along trajectories. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) *DLT 2004*. LNCS, vol. 3340, pp. 163–174. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-30550-7\\_14](https://doi.org/10.1007/978-3-540-30550-7_14)
6. Domaratzki, M.: Trajectory-based codes. *Acta Inf.* **40**, 491–527 (2004)
7. Domaratzki, M., Rozenberg, G., Salomaa, K.: Interpreted trajectories. *Fundamenta Informaticae* **73**, 81–97 (2006)
8. Enaganti, S., Kari, L., Kopecki, S.: A formal language model of DNA polymerase enzymatic activity. *Fundamenta Informaticae* **138**, 179–192 (2015)

9. Enaganti, S., Ibarra, O., Kari, L., Kopecki, S.: On the overlap assembly of strings and languages. *Nat. Comput.* **16**, 175–185 (2017)
10. Franco, G., Manca, V.: Algorithmic applications of XPCR. *Nat. Comput.* **10**, 805–811 (2011)
11. Han, Y.-S., Ko, S.-K., Ng, T., Salomaa, K.: State complexity of insertion. *Int. J. Found. Comput. Sci.* **27**, 863–878 (2016)
12. Holzer, M., Jakobi, S.: Descriptive complexity of chop operations on unary and finite languages. *J. Autom. Lang. Comb.* **17**(2–4), 165–183 (2012)
13. Holzer, M., Jakobi, S., Kutrib, M.: The chop of languages. *Theor. Comput. Sci.* **682**, 122–137 (2017)
14. Jürgensen, H., Konstantinidis, S.: Codes. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages*, Vol. 1, pp. 511–607. Springer, Heidelberg (1997). [https://doi.org/10.1007/978-3-642-59136-5\\_8](https://doi.org/10.1007/978-3-642-59136-5_8)
15. Kari, L.: On language equations with invertible operations. *Theor. Comput. Sci.* **132**, 129–150 (1994)
16. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. *Inf. Comput.* **131**, 47–61 (1996)
17. Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: syntactic constraints. *Theor. Comput. Sci.* **197**, 1–56 (1998)
18. Reikofski, J., Yao, B.Y.: Polymerase chain reaction (PCR) techniques for site-directed mutagenesis. *Biotechnol. Adv.* **10**, 535–547 (1992)
19. Shallit, J.: *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, Cambridge (2009)



# Two-Way Automata over Locally Finite Semirings

Louis-Marie Dando and Sylvain Lombardy<sup>(✉)</sup>

LaBRI UMR 5800, Université de Bordeaux, INP Bordeaux, CNRS, Bordeaux, France  
{louis-marie.dando,sylvain.lombardy}@labri.fr

**Abstract.** Two-way transducers or weighted automata are in general more powerful than one-way ones. We show that two-way automata over locally finite semirings may have undefined behaviour. We prove that it is decidable whether this behaviour is defined, and, if it is, we show that two-way automata over locally finite semirings are equivalent to one-way automata.

## 1 Introduction

Weighted two-way automata and transducers have been recently intensively studied for their interest in verification [3]. Their expressiveness is in general larger than the expressiveness of one-way models. We consider in this paper two-way automata over locally finite semirings. Finite or locally finite semirings occur in many models, like distributive lattices or fuzzy logic. One-way automata over these semirings have been studied for many decades. For instance, the first proof of the limitedness problem [4] relies on automata on the idempotent semiring  $\{0, 1, \omega, \infty\}$  (where 1 means “something” and  $\omega$  means “a lot”).

It is folklore that every one-way automaton over a locally finite semiring is equivalent to a deterministic finite automaton where the weight of the run only depends on the state where the run stops. For two-way automata over locally finite semirings, the situation is not as simple. For instance, if the weights belong to  $\mathbb{Z}/2\mathbb{Z}$ , then the weight of an input depends on the parity of the number of accepting runs; since in a two-way automaton, there may exist an infinite number of runs accepting some input, this weight may be not defined.

For every two-way automaton over a locally finite semiring, we build an automaton that describes the potentially infinite family of weights of runs on every input. From this object, knowing which infinite sums are defined in the locally finite semiring is sufficient to decide whether the behaviour of the two-way automaton is defined. We also prove that every two-way automaton over a locally finite semiring with a defined behaviour is equivalent to a one-way automaton.

In Sect. 2, we consider locally finite semirings. In particular, we study how the *additive order* allows to encode infinite sums. In Sect. 3, we introduce weighted two-way automata over locally finite semirings and we show that they can be

normalized in such a way that the weight of every run only depends on the final state, and the move of the input head is fully characterized by the current state. In Sect. 4, we use an extension of crossing sequences [6, 7] to convert a two-way automaton into a one-way automaton. It leads to a deterministic one-way automaton where each final state describes the (potentially infinite) family of weights of runs of the two-way automaton on the input. It is then decidable whether the weight of every input is defined, and, if it is, then the deterministic automaton can be turned into a (deterministic) one-way automaton equivalent to the two-way automaton.

## 2 Locally Finite Semirings

A semiring  $(\mathbb{K}, +, \cdot)$  is a set  $\mathbb{K}$  endowed with two associative operations: a commutative *addition* and a *multiplication* that is distributive over the addition. Moreover, the semiring contains at least two distinct elements which are respectively neutral for each of these operations:  $0_{\mathbb{K}}$  for the addition and  $1_{\mathbb{K}}$  for the multiplication; it is also required for  $0_{\mathbb{K}}$  to be an annihilator for the multiplication.

A semiring  $\mathbb{K}$  is locally finite if, for every finite subset  $F$  of  $\mathbb{K}$  containing both  $0_{\mathbb{K}}$  and  $1_{\mathbb{K}}$ , the semiring generated by  $F$  is finite.

Moreover, we assume that  $\mathbb{K}$  is endowed with a partially defined operator for countable sums:  $\sum_I x$ , where  $I$  is a countable set and  $x = (x_i)_{i \in I}$  a family of elements of  $x$ . If  $I$  is finite, then  $\sum_I x$  is always defined and equal to the sum of elements of  $x$ . We also assume that two families equal up to a permutation have the same sum.

In this paper, we deal with weighted automata where a finite number of elements of some locally finite semiring occur. These elements generate a finite semiring. In the sequel, we assume that the semiring  $\mathbb{K}$  is finite.

*Example 1.* Let  $(L, \vee, \wedge)$  be an infinite distributive lattice;  $L$  is a semiring with  $\vee$  as addition and  $\wedge$  as multiplication.  $0_L$  is the minimum element of  $L$  and  $1_L$  is the maximum element. The infinite sum of a family of elements of  $L$  is the supremum of this family; if it exists. In lattices, the supremum of families with a finite number of distinct elements is always defined.  $\square$

*Example 2.* Let  $\mathbb{K}_1 = \{o, i, x\}$  be the finite semiring where  $o$  is the zero,  $i$  is the unit,  $i + i = o$ , and  $r + x = r \cdot x = x$  for every  $r$  in  $\{i, x\}$ . Intuitively, this semiring allows to count values modulo 2, and  $x$  stands for values where the parity information is lost. In  $\mathbb{K}_1$ , the sum of any family which contains at least one  $x$  is defined and equal to  $x$ , and a family that does not contain any  $x$  is summable if and only if the number of occurrences of  $i$  is finite.  $\square$

Since  $\mathbb{K}$  is finite and families are unordered, a family is totally characterized by the number of occurrences of each element of  $\mathbb{K}$ . Hence, we represent such a family  $s$  by a vector  $v$  in  $(\mathbb{N} \cup \{\infty\})^{\mathbb{K}}$  such that, for every  $x$  in  $\mathbb{K}$ ,  $v_x$  is the number of occurrences of  $x$  in  $s$ . The *evaluation* of  $v$  is the sum of  $s$  if it is defined. Notice that if  $s$  and  $s'$  are two families respectively represented by vectors  $v$  and  $v'$ , then the union of  $s$  and  $s'$  is represented by  $v + v'$ .

In Sect. 4.3, a one-way deterministic  $\mathbb{K}$ -automaton equivalent to a two-way  $\mathbb{K}$ -automaton is built. The states of this automaton store vectors representing families. In order to build a finite number of states, we need to define a finite set of representatives. Two vectors  $v$  and  $v'$  in  $(\mathbb{N} \cup \{\infty\})^{\mathbb{K}}$  are equivalent if for every vector  $u$ ,  $v + u$  and  $v' + u$  have the same evaluation. This property will be required during a determinization step.

We show that every vector in  $(\mathbb{N} \cup \{\infty\})^{\mathbb{K}}$  is equivalent to a vector where finite entries are bounded. We consider the natural external product: for every  $(k, x)$  in  $\mathbb{N} \times \mathbb{K}$ ,  $k.x$  is the sum of  $k$  elements  $x$ . Likewise, if it is defined, the infinite sum of  $x$  is denoted  $\infty.x$ .

**Definition 1.** *Let  $x$  be an element of a locally finite semiring  $\mathbb{K}$ . The additive order of  $x$  is the minimal couple  $(n_x, p_x)$  of integers (with  $p_x > 0$ ) verifying*

$$\forall k \geq n_x, (k + p_x).x = k.x. \quad (1)$$

The additive order of  $x$  is always defined in a locally finite semiring. Actually, the set  $\{k.x \mid k \in \mathbb{N}\}$  is finite and the sequence  $s = (k.x)_{k \in \mathbb{N}}$  is ultimately periodic: if  $s_i = s_j$ , then for every  $k$   $s_{i+k} = i.x + k.x = j.x + k.x = s_{j+k}$ . Thus,  $p_x$  is the minimal distance between two occurrences of the same value in  $s$  and  $n_x$  is the smallest  $i$  such that  $s_i$  appears infinitely often in  $s$ .

If  $(n_x, p_x)$  is the order of  $x$ , then for every  $(m, q)$  with  $m$  larger than or equal to  $k$  and  $q$  multiple of  $p_x$ , it also holds  $\forall k \geq m, (k + q).x = k.x$ . The additive order of a finite semiring is then the minimal pair which is admissible for every element of the semiring.

**Definition 2.** *Let  $\mathbb{K}$  be a finite semiring. The additive order of  $\mathbb{K}$  is the couple*

$$(\max\{n_x \mid x \in \mathbb{K}\}, \text{lcm}\{p_x \mid x \in \mathbb{K}\}), \quad (2)$$

where  $\text{lcm} X$  is the least common multiple of elements of  $X$ . For every  $k$  in  $\mathbb{N} \cup \{\infty\}$ , the value of  $k$  modulo  $(n, p)$  is defined as

$$k \bmod(n, p) = \begin{cases} k & \text{if } k < n \text{ or } k = \infty \\ n + ((k - n) \bmod p) & \text{otherwise.} \end{cases} \quad (3)$$

Hence,  $k \bmod(n, p)$  is in  $\llbracket 0; n + p - 1 \rrbracket \cup \{\infty\}$  and if the additive order of a finite semiring  $\mathbb{K}$  is  $(n, p)$ , then for every  $x$  in  $\mathbb{K}$ ,  $k.x = (k \bmod(n, p)).x$ .

If  $(n, p)$  is the additive order of  $\mathbb{K}$ , then every vector  $v$  in  $(\mathbb{N} \cup \{\infty\})^{\mathbb{K}}$  is equivalent to the vector where each entry is considered modulo  $(n, p)$ . The number of distinct vectors modulo  $(n, p)$  is equal to  $(n + p + 1)^{|\mathbb{K}|}$  (some entries may be equal to  $\infty$ ).

In the sequel,  $\mathcal{N}_{n,p}$  is the semiring  $\llbracket 0; n + p - 1 \rrbracket \cup \{\infty\}$  where the operations between finite integers are modulo  $(n, p)$ .

*Example 3.* The additive order of elements of  $\mathbb{K}_1$  (Example 2) is:  $\text{order}(o) = (0, 1)$ ,  $\text{order}(i) = (0, 2)$ , and  $\text{order}(x) = (1, 1)$ . Hence, the additive order of  $\mathbb{K}_1$  is  $(1, 2)$ : for every element  $k$  in  $\mathbb{K}_1$ ,  $k = 3.k$ .  $\square$

### 3 Two-Way Automata

#### 3.1 Definition and Behaviour

We follow in this paper the definition of two-way weighted automata given in [5]. Contrary to the classical definition of two-way finite automata [6, 7], the move of the reading head does not depend on the transitions but on the states. This model was first used in [1] and was there proved to be equivalent to the classical one.

**Definition 3.** Let  $\mathbb{K}$  be a locally finite semiring,  $A$  an alphabet and  $\square$  a special symbol, called endmarker, which does not belong to  $A$ . A two-way  $\mathbb{K}$ -automaton over  $A$  is a tuple  $(F, B, E, I, T)$ , where

- $F$  is the set of forward states,  $B$  the set of backward states;  $Q = F \cup B$  is the set of states;
- $E : Q \times (A \cup \{\square\}) \times Q \longrightarrow \mathbb{K}$  is the transition weight function;
- $I : F \longrightarrow \mathbb{K}$  and  $T : F \longrightarrow \mathbb{K}$  are the initial and the final weight function.

The set of transitions is the support of  $E$ , the set of initial states is the support of  $I$ , and the set of final states is the support of  $T$ . For every transition  $e = (p, a, q)$ ,  $\lambda(e) = a$  is the label of  $e$ ,  $\sigma(e) = p$  its source, and  $\tau(e) = q$  its target.

For every state  $p$ , we note  $\delta(p) = 1$  if  $p$  is in  $F$  and  $\delta(p) = -1$  if  $p$  is in  $B$ . The head of a two-way automaton can not go beyond endmarkers, hence for every pair  $(p, q)$  of states, if  $\delta(p)\delta(q) = 1$ , then  $E(p, \square, q) = 0_{\mathbb{K}}$ .

A run of length  $k$  in the automaton is a triple  $(p, (e_i)_{i \in \llbracket 1; k \rrbracket}, q)$ , where  $p$  and  $q$  are in  $F$  and  $(e_i)_{i \in \llbracket 1; k \rrbracket}$  is a sequence of transitions such that, for every  $i$  in  $\llbracket 1; k - 1 \rrbracket$ ,  $\tau(e_i) = \sigma(e_{i+1})$ ; if  $k > 0$ , then  $p = \sigma(e_1)$  and  $q = \tau(e_k)$ ; otherwise  $p = q$ .

This run admits a label  $w = w_1 \dots w_n$  in  $A^*$  if there exists a mapping  $\pi : \llbracket 1; k \rrbracket \rightarrow \llbracket 0; n + 1 \rrbracket$  such that  $\pi(1) = 1$ ,  $\pi(k) = n$ , and

- for every  $i$  in  $\llbracket 2; k \rrbracket$ ,  $\pi(i) = \pi(i - 1) + \delta(\sigma(e_i))$ ,
- for every  $i$  in  $\llbracket 1; k \rrbracket$ ,  $\lambda(e_i) = w_{\pi(i)}$  if  $\pi(i)$  is in  $\llbracket 1; n \rrbracket$ , and  $\lambda(e_i) = \square$  otherwise.

The function  $\pi$  is the *position mapping*, giving the position of the reading head during the computation.

The run  $(p, (e_i)_{i \in \llbracket 1; k \rrbracket}, q)$  meets  $k + 1$  states. The sequence of these states is defined by  $p_0 = p$  and  $p_j = \tau(e_j)$  for every  $j$  in  $\llbracket 1; k \rrbracket$ . The *position* of the  $j$ -th state of the run is defined by  $\text{pos}(0) = 0$ , and

$$\forall j \in \llbracket 1; k \rrbracket, \text{pos}(j) = \begin{cases} \text{pos}(j - 1) & \text{if } \delta(p_j) \neq \delta(p_{j-1}) \\ \text{pos}(j - 1) + \delta(p_j) & \text{otherwise} \end{cases} \quad (4)$$

The positions of states and transitions are related; for every  $j \in \llbracket 1; k \rrbracket$ ,  $\text{pos}(j) = \pi(j)$  if  $\delta(p_j) = 1$ , and  $\text{pos}(j) = \pi(j) - 1$  otherwise.

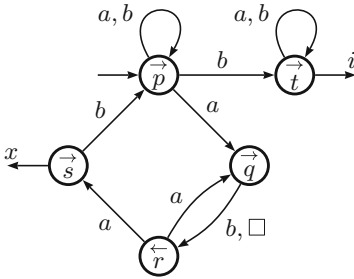
The weight of a run  $(p, (e_i)_{i \in \llbracket 1; k \rrbracket}, q)$  is  $I_p \cdot (\prod_{i=1}^k E(e_i)) \cdot T_q$ . The weight of a word  $w$  in a two-way weighted automaton is defined if the family of the weights of runs with label  $w$  is summable.

**Definition 4.** An automaton is valid if for every word  $w$  the weight of  $w$  in the automaton is defined, and two automata are equivalent if, for every word  $w$ , either the weight of  $w$  is undefined in both automata, or the weight of  $w$  is defined in both automata and is equal.

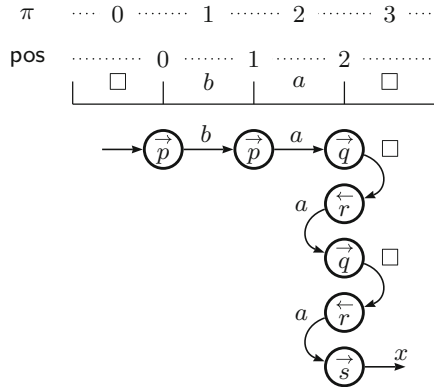
*Remark 1.* If  $B = \emptyset$ , then the two-way  $\mathbb{K}$ -automaton is actually one-way, and every one-way  $\mathbb{K}$ -automaton can be described as a two-way  $\mathbb{K}$ -automaton where  $B = \emptyset$ .

*Example 4.* Let  $\mathcal{A}_1$  be the two-way  $\mathbb{K}_1$ -automaton of Fig. 1. The final weight of the states is represented by an arrow outgoing from states. States  $p, q, s$  and  $t$  are forward,  $r$  is backward. States  $p, q$  and  $r$  are not final, which means that their final weight is equal to  $0_{\mathbb{K}_1} = o$ . The weight of every transition is  $1_{\mathbb{K}_1} = i$ : it is a *characteristic* automaton.

Figure 1(b) shows a run of  $\mathcal{A}_1$ . State  $p$  appears at position 0 and 1. State  $r$  appears twice at the same position; this is an *unmoving circuit*: the part of the run between the two occurrences of  $r$  can be repeated in order to get longer runs over the same word. □



(a) The two-way  $\mathbb{K}_1$ -automaton  $\mathcal{A}_1$ .



(b) A run of  $\mathcal{A}_1$  over the word  $ba$

**Fig. 1.** A two-way  $\mathbb{K}_1$ -automaton and one of its runs.

*Remark 2.* In the sequel, we state results for two-way automata over finite semirings. If  $\mathcal{A}$  is a two-way automaton over a locally finite semiring  $\mathbb{K}$ , then it can be considered as an automaton over the finite semiring  $\mathbb{K}'$  where  $\mathbb{K}'$  is the semiring generated by the weights occurring in  $\mathcal{A}$ .

### 3.2 Characteristic and $\delta$ -Normalized Two-Way Automaton

Like for one-way automata, we show here that two-way automata with weights in a finite semiring are equivalent to *characteristic* two-way automata where



the weight of the run only depends on the final state. In our model, in such an automaton, the weights of all transitions as well as initial weights are all equal to  $1_{\mathbb{K}}$ .

**Proposition 1.** *Let  $\mathcal{A}$  be a two-way automaton over a locally finite semiring. There exists a characteristic two-way automaton  $\mathcal{B}$  equivalent to  $\mathcal{A}$ .*

Actually, through a run, the product of weights spans over a finite set and this can be stored in states. Hence, if  $\mathbb{K}$  is the finite semiring generated by the weights of  $\mathcal{A}$ , then states of  $\mathcal{B}$  belong to  $Q \times \mathbb{K}$ .

Figure 1(b) shows that transitions during a run can be classified into four types: forward transitions, backward transitions, and forward and backward half-turns. We consider now  $\delta$ -normalized two-way automata, where the type of a transition depends on its source.

**Definition 5.** ([2]). *A two-way automaton  $\mathcal{A} = (F, B, E, I, T)$  is  $\delta$ -normalized if  $F$  and  $B$  are respectively partitioned into  $F = F_+ \cup F_-$  and  $B = B_+ \cup B_-$ , such that every final state is in  $F_+$  and, for every state  $p$  and every transition  $(p, a, q)$ ,  $q$  is in  $F$  if and only if  $p$  is in  $F_+ \cup B_+$ .*

*Example 5.* The automaton  $\mathcal{A}_1$  of Fig. 1 is  $\delta$ -normalized. Actually,  $F_+ = \{p, s, t\}$ ,  $F_- = \{q\}$ ,  $B_+ = \{r\}$  and  $B_- = \emptyset$ .  $\square$

**Proposition 2.** ([2]). *For every two-way  $\mathbb{K}$ -automaton  $\mathcal{A}$ , there exists an equivalent  $\delta$ -normalized  $\mathbb{K}$ -automaton  $\mathcal{B}$ .*

To convert  $\mathcal{A}$  into  $\mathcal{B}$ , every state of  $\mathcal{A}$  is split into two copies, the outgoing transitions that go to a forward state are assigned to the first copy, those that go to a backward state are assigned to the second one. This construction applied to a characteristic automaton gives a characteristic automaton.

From now on, we assume that two-way automata are characteristic and  $\delta$ -normalized.

## 4 Counting Paths

To convert two-way  $\mathbb{K}$ -automata into one-way automata, we use a variant of the method of crossing sequences initiated in [6]. Nevertheless, the method must be improved in order to keep records of the number of paths.

More precisely, let  $(n, p)$  be the order of  $\mathbb{K}$ . The algorithm builds a (one-way) deterministic automaton  $\mathcal{B}$ , where each state is a crossing sequence (that may have states repeating at most once).  $\mathcal{B}$  is then used to build a one-way automaton. The idea is to build from  $\mathcal{B}$  an automaton that “counts” (modulo  $(n, p)$ ) the number of runs with a given weight ending in the last state of a crossing sequence.

## 4.1 Crossing Sequences

**Definition 6.** Let  $\rho$  be a run with label  $w$  in a two-way automata, and let  $(p_j)_{j \in \llbracket 0; k \rrbracket}$  be the sequence of states met by  $\rho$ . For every  $i$  in  $\llbracket 0; |w| \rrbracket$ , the crossing sequence of  $\rho$  at position  $i$  is the subsequence of states  $p_j$  such that  $\text{pos}(j) = i$ .

*Example 6.* On Fig. 1(b), crossing sequences are  $(p)$ ,  $(p)$  and  $(q, r, s, r, s)$ .

*Remark 3.* Crossing sequences are sequences of states with odd length. More precisely, the first state is in  $F$ , and there is an alternation of states in  $F$  and  $B$ . Using a regular expression, we can say that crossing sequences are in  $(FB)^*F$ .

If a crossing sequence of  $\rho$  contains (at least) two occurrences of the same state, it means that during the run, the automaton comes back to the same state with the input head at the same position; we call this an *unmoving circuit*. Such a circuit can be removed from  $\rho$  in order to obtain a valid run for the same input. On the other hand, it can be iterated to produce an infinite number of valid runs. In a characteristic automaton, all these runs have the same weight.

In the classical crossing sequence construction [6], *reduced crossing sequences*, that are crossing sequences without repetitions, are considered; we have to consider also *crossing sequences with 1 repetition* in which no state appears more than twice, in order to detect unmoving circuits.

**Proposition 3.** ([2]). Let  $\rho$  be a run of a  $\delta$ -normalized two-way  $\mathbb{K}$ -automaton over a word  $w$ , and let  $(c_i)_{i \in \llbracket 0; |w| \rrbracket}$  be the list of the crossing sequences of  $\rho$ . Then  $\rho$  is characterized by  $w$  and  $(c_i)_{i \in \llbracket 0; |w| \rrbracket}$ .

Notice that this proposition is independent from the weights on transitions and that it is true whether or not the two-way automaton is characteristic.

We describe briefly how, starting with two consecutive crossing sequences  $c_{i-1}$  and  $c_i$ , it is possible to build the unique list of transitions in position  $i$  which is consistent with these crossing sequences, and the  $i$ -th letter of  $w$  denoted  $a$ .

The algorithm scans sequences  $c_{i-1}$  and  $c_i$ . If the current state  $p$  of  $c_{i-1}$  is in  $F_+$ , there is a transition from  $p$  to the current state  $q$  of  $c_i$ ; if it is in  $F_-$ , there is a transition from  $p$  to the next state  $p'$  of  $c_{i-1}$ . The algorithm deals then with the next states. If the current states are in  $B$ , the analysis is based on the nature of the current state in  $c_i$ . This process produces the list of transitions in position  $i$  in the run. It is formally described in [2].

If  $c_{i-1}$  and  $c_i$  are not successive crossing sequences of a run, the algorithm may fail or build a list of triples which are not transitions of the automaton.

**Definition 7.** Let  $\mathcal{A} = (F, B, E, I, T)$  be a  $\delta$ -normalized two-way  $\mathbb{K}$ -automaton, and let  $c_1$  and  $c_2$  be two potential crossing sequences in  $(FB)^*F$ .

- The sequence  $c_2$  is a successor of  $c_1$  by the letter  $a$  if the analysis applied to  $c_1$  and  $c_2$  succeeds and returns a list of transitions of  $\mathcal{A}$  with label  $a$ .
- $c_1 = (p_i)_{i \in \llbracket 1; 2h+1 \rrbracket}$  is an initial crossing sequence if  $p_1$  is initial, and, for every  $i$  in  $\llbracket 1; h \rrbracket$ ,  $(p_{2i}, \square, p_{2i+1})$  is a transition of  $\mathcal{A}$ .

- $c_1 = (p_i)_{i \in [1; 2h+1]}$  is an final crossing sequence if  $p_{2h+1}$  is final, and, for every  $i$  in  $[1; h]$ ,  $(p_{2i-1}, \square, p_{2i})$  is a transition of  $\mathcal{A}$ .

Let  $\mathcal{S}$  be the infinite one-way automaton of crossing sequences of the  $\delta$ -normalized two-way  $\mathbb{K}$ -automaton  $\mathcal{A}$ . Its states are the elements of  $(FB)^*F$ . A state  $c$  is initial if  $c$  is an initial crossing sequence, it is final if  $c$  is a final crossing sequence, and there is a transition from  $c$  to  $c'$  with label  $a$  if  $c'$  is a successor of  $c$  by  $a$ . This automaton accepts the same words as  $\mathcal{A}$ , and there is a bijection between runs of  $\mathcal{S}$  and runs of  $\mathcal{A}$ .  $\mathcal{S}$  can be turned into a  $\mathbb{K}$ -automaton: for every final crossing sequence  $c = (p_i)_{i \in [1; 2h+1]}$ , the final weight of  $c$  can be set as the final weight of  $p_{2h+1}$  in  $\mathcal{A}$ . This gives an infinite one-way  $\mathbb{K}$ -automaton equivalent to  $\mathcal{A}$ . Notice that the  $\mathbb{K}$ -automaton of crossing sequences of a two-way  $\mathbb{K}$ -automaton  $\mathcal{A}$  is not always equivalent to  $\mathcal{A}$ , in particular if  $\mathbb{K}$  is not commutative. Here, the fact that  $\mathcal{A}$  is characteristic is crucial.

To convert two-way automata (without weights) to NFA, it is sufficient to restrict  $\mathcal{S}$  to states without any repetition; this is a classic construction of one-way NFA from two-way NFA.

We want to check whether there is an infinite number of runs that end in a given state. As we have seen, as soon as a run has an unmoving circuit, this circuit can be iterated to get an infinite number of runs. Therefore, it is unnecessary to keep track of crossing sequences with more than two occurrences of the same state. If a crossing sequence  $c$  of a run  $\rho$  contains three times the same state, then the unmoving circuit between the first and the second one (or between the second one and the third one) can be removed; the resulting path has still an unmoving circuit.

**Lemma 1.** *Let  $\mathcal{A}$  be a  $\delta$ -normalized characteristic two-way  $\mathbb{K}$ -automaton, and let  $\rho$  be a run on  $\mathcal{A}$ . If  $\rho$  admits a crossing sequence at position  $i$  in which a state  $p$  appears more than two times, then there exists a run  $\rho'$  in which  $p$  appears only twice in the crossing sequence at position  $i$ , and  $\rho'$  and  $\rho$  end in the same state.*

Each crossing sequence with 1 repetition is the witness of an infinite number of runs with the same weight. Lemma 1 tells that every run of  $\mathcal{A}$  is either a run without any unmoving circuit, or a run which admits such a witness.

## 4.2 Automaton of Crossing Sequences with One Repetition

Let  $\mathcal{A} = (F, B, E, I, T)$  be a two-way  $\mathbb{K}$ -automaton. Let  $C_1$  (resp.  $C_2$ ) be the sequences of  $(FB)^*F$  where each state appears at most once (resp. twice). Let  $(n, p)$  be the order of the finite semiring generated by the coefficients of  $\mathcal{A}$ . Let  $\mathcal{B}$  be the one-way automaton over  $\mathcal{N}_{n,p}$  with states  $R = C_1 \cup C_2$  defined by:

- if  $c$  is an initial crossing sequence, then  $c$  is an initial state of  $\mathcal{B}$ , with weight 1 if  $c$  is in  $C_1$  and  $\infty$  if  $c$  is in  $C_2$ ;
- if  $c'$  is a successor of  $c$  by  $a$ , then  $(c, a, c')$  is a transition of  $\mathcal{B}$ , with weight  $\infty$  if  $c$  is in  $C_1$  and  $c'$  in  $C_2$ , and weight 1 otherwise;

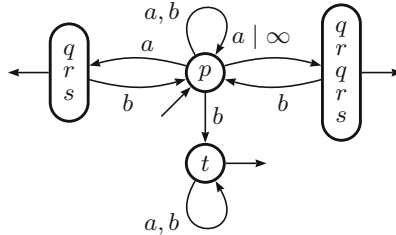
- if  $c$  is a final crossing sequence, then  $c$  is final in  $\mathcal{B}$ , with weight 1.

The only interesting part of  $\mathcal{B}$  is its trim part formed with states that can actually occur in some runs. Thus, our construction only requires to build the accessible part of  $\mathcal{B}$ .

Automaton  $\mathcal{B}$  is a variant of the crossing sequence automaton.

On the one hand, there is a bijection between runs with weight 1 in  $\mathcal{B}$  and runs without unmoving circuits in  $\mathcal{A}$ . On the other hand, every run in  $\mathcal{B}$  with weight  $\infty$  is the witness of an infinite number of runs in  $\mathcal{A}$  ending in the same state. Notice that some runs of  $\mathcal{A}$  may admit several witnesses, but all these witnesses show that there are infinitely many runs for the same final state. A run of  $\mathcal{B}$  ends in a final crossing sequence; the weight of the corresponding run in  $\mathcal{A}$  is given by the last state of this crossing sequence.

*Example 7.* Four different crossing sequences may appear in the runs of  $\mathcal{A}_1$ ; this leads to the automaton  $\mathcal{B}_1$  on Fig. 2. The weight of every transition in this automaton is equal to 1, except the transition that goes to the crossing sequence with repetitions where the weight is equal to  $\infty$ . For instance, this automaton shows that there is an infinite number of runs over  $a$  in  $\mathcal{A}_1$  that start in state  $p$  and stop in state  $s$ . □



**Fig. 2.** The  $\mathcal{N}_{n,p}$ -automaton  $\mathcal{B}_1$  of crossing sequences of  $\mathcal{A}_1$ .

### 4.3 Gathering Runs with the Same Label

On some locally finite semirings, the automaton  $\mathcal{B}$  is sufficient to decide whether the two-way automaton is valid and to build an equivalent one-way automaton. In these cases (Case 1 and 2), the last step of the algorithm can be avoided; it leads to a simpler construction.

*Case 1: Every Infinite Sum Is Defined.* A non deterministic characteristic one-way  $\mathbb{K}$ -automaton  $\mathcal{C}$  equivalent to the two-way  $\mathbb{K}$ -automaton can be built from the automaton  $\mathcal{B}$  of crossing sequences with 1 repetition. The weight of a run is  $\infty$  as soon as the weight  $\infty$  is met (otherwise it is 1); this information can be stored in the states of  $\mathcal{C}$  that belong to  $R \times \{1, \infty\}$ , where  $R$  is the set of states of  $\mathcal{B}$ . The accessible part of  $\mathcal{C}$  is inductively defined as:

- if  $p$  is initial in  $\mathcal{B}$  with weight  $k$ , then  $(p, k)$  is initial in  $\mathcal{C}$ ;
- if  $(p, a, q)$  is a transition in  $\mathcal{B}$  with weight  $k$  and  $(p, r)$  is a state of  $\mathcal{C}$ , then  $((p, k), a, (q, k.r))$  is a transition of  $\mathcal{C}$ ;
- if  $p$  is final in  $\mathcal{B}$ , then  $p$  is a final crossing sequence whose last state is final with weight  $x$  in  $\mathbb{K}$ ; every state  $(p, k)$  of  $\mathcal{C}$  is final with weight  $k.x$ .

*Case 2: No Infinite Sum Is Defined.* The two-way  $\mathbb{K}$ -automaton is valid if and only if no transition with weight  $\infty$  appears in any run of  $\mathcal{B}$ . If there is no such transition, then the previous construction applies.

*Case 3: Some Infinite Sums Are Defined, Some Are Not.* To decide whether the family of weights of all runs labelled by a given word is defined, all these runs must be gathered. We use a determinization-like algorithm to build an automaton that computes, for each word  $w$ , the number (modulo  $(n, p)$ ) of runs that end in each final crossing sequence. The determinization  $\mathcal{D}$  of  $\mathcal{B}$  gathers the vectors corresponding to all runs on every input. Each state of  $\mathcal{D}$  is a vector in  $\mathcal{N}_{n,p}^R$ , where  $R$  is the set of states of  $\mathcal{B}$ .

- the initial vector of  $\mathcal{D}$  is  $I$ , where  $I_c$  is the initial weight of  $c$  in  $\mathcal{B}$ ;
- if  $X$  is a state of  $\mathcal{D}$ , then the successor of  $X$  by letter  $a$  is the state  $Y$  defined by:

$$Y_{c'} = \sum_{c \in R} X_c \cdot E(c, a, c'). \quad (5)$$

- A state  $X$  is final if there exists a final crossing sequence  $c$  such that  $X_c$  is different from 0.

We consider the projection  $\pi$  from the final crossing sequences of  $R$  onto  $F$  which maps  $c$  onto the last state of  $c$ ; this projection allows to map every final state  $X$  of  $\mathcal{D}$  onto a vector  $v$  in  $\mathcal{N}_{n,p}^F$ :

$$\forall f \in F, v_f = \sum_{c \in R, c \text{ final}, \pi(c)=p} X_c. \quad (6)$$

For every word  $w$ , if the run on  $w$  in  $\mathcal{D}$  ends in state  $X$ , then for every  $c$ ,  $X_c$  is the number (modulo  $(n, p)$ ) of runs on  $w$  in  $\mathcal{A}$  whose last crossing sequence is  $c$ . Hence,  $v_f$  is the number (modulo  $(n, p)$ ) of runs on  $w$  in  $\mathcal{A}$  that end in state  $f$ .

Therefore, the weight of  $w$  in  $\mathcal{A}$  is equal to the evaluation of  $v$ ; if this evaluation is not defined, then the weight of  $w$  in  $\mathcal{A}$  is not defined. If, for every final state  $X$  of  $\mathcal{D}$ , the evaluation of the corresponding vector is defined, this weight can be assigned to  $X$  as a final weight. This turns  $\mathcal{D}$  to a characteristic and deterministic one-way  $\mathbb{K}$ -automaton equivalent to  $\mathcal{A}$ .

**Theorem 1.** *Over locally finite semirings, the validity of two-way automata is decidable, and every valid two-way automaton is equivalent to a one-way automaton.*

The construction follows the algorithm outlined in this paper. Given a two-way automaton  $\mathcal{A}$  over a locally finite semiring, we can first consider it as an automaton over  $\mathbb{K}$ , the finite semiring generated by the weights of the automaton. A  $\delta$ -normalized characteristic two-way  $\mathbb{K}$ -automaton  $\mathcal{A}'$  equivalent to  $\mathcal{A}$  is then constructed. The automaton of crossing sequences with 1 repetition of  $\mathcal{A}'$ ,  $\mathcal{B}$ , allows to compute the number of runs that stop in each state. The end of the algorithm depends on the cases described in Sect. 4.3. In Cases 1 and 2 the validity is directly decidable on  $\mathcal{B}$  and a one-way  $\mathbb{K}$ -automaton equivalent to  $\mathcal{A}$  can also be derived from  $\mathcal{B}$ . In Case 3, a determinization step is required. As explained above, this deterministic automaton allows both to decide the validity and to build a one-way  $\mathbb{K}$ -automaton equivalent to  $\mathcal{A}$ , if there exists one.

*Remark 4.* The fact that the weight of a word  $w$  is defined in  $\mathcal{A}$  only depends on the state ending the run over  $w$  in  $\mathcal{D}$ . Hence the language of words with an undefined weight is regular.

*Example 8.* We denote  $X = (q, r, q, r, s)$  and  $Y = (q, r, s)$ ; hence the states of automaton  $\mathcal{B}_1$  are  $p, t, X$  and  $Y$ . The states of the determinization of  $\mathcal{B}_1$  are vectors in  $\mathcal{N}_{n,p}^{\{p,t,X,Y\}}$ ; the determinization is drawn on Fig. 3. Only non zero entries are written in each state. The initial state is not final since it does not contain any final crossing sequence.

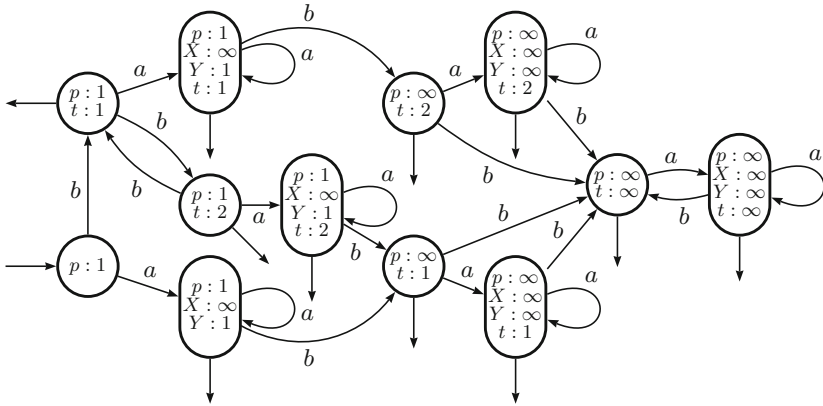


Fig. 3. The determinization of  $\mathcal{B}_1$ .

We replace now each vector in  $\mathcal{N}_{n,p}^{\{p,t,X,Y\}}$  by the corresponding vector in  $\mathcal{N}_{n,p}^{\mathbb{K}_1}$  (the entry corresponding to  $o$  is not shown since it does not influence the evaluation). Consider for instance the vector  $[p:1, X:\infty, Y:1, t:1]$ .  $p$  is not a final crossing sequence and does not contribute; the last state of  $X$  and  $Y$  is  $s$  with final weight  $x$ , hence,  $x$  appears with multiplicity  $\infty + 1 = \infty$  and the final weight of  $t$  is  $i$ ; finally, the family of weights for this state is  $[x:\infty, i:1]$  and the final weight is  $\infty \cdot x + 1 \cdot i = x$ . We obtain the automaton of Fig. 4. Notice that

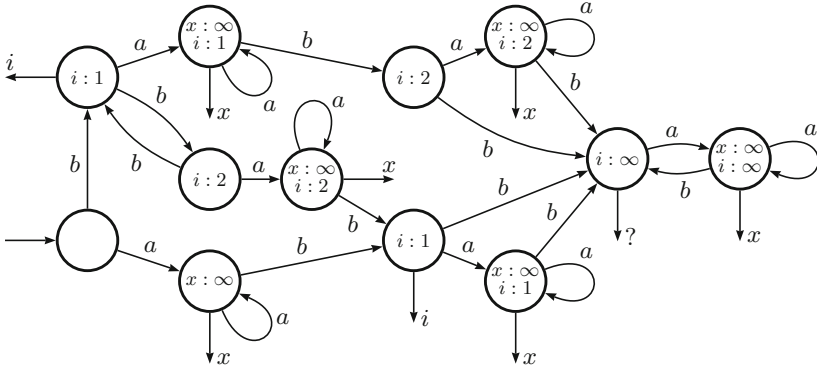


Fig. 4. The automaton  $\mathcal{A}_1$  is not valid.

states with vector  $2.i$  are not final since  $i + i = o = 0_{\mathbb{K}}$ . The construction fails because one state corresponds to the vector  $\infty.i$  and the infinite sum of  $i$  is not defined, thus no final weight can be defined for this state. For every word leading to this state (for instance  $abb$ ), there is in  $\mathcal{A}_1$  an infinite number of runs and the weight of each of these runs is equal to  $i$ . Therefore, the two-way automaton  $\mathcal{A}_1$  is not valid.  $\square$

### 5 Conclusion

This paper describes an algorithm to decide whether a two-way  $\mathbb{K}$ -automaton is valid, and, if it is, to build an equivalent one-way  $\mathbb{K}$ -automaton. The construction involves several steps. The first two – getting a characteristic  $\mathbb{K}$ -automaton and then a  $\delta$ -normalized  $\mathbb{K}$ -automaton – have low state complexity: respectively  $|\mathbb{K}||Q|$  and  $2|Q|$ . In contrast, the last two have huge complexity. The state complexity of the automaton of crossing sequences with 1 repetition is in  $2^{O(|Q| \log |Q|)}$  and the determinization is in  $(n+p)^{|Q|}$ , where  $Q$  is the set of states of the automaton on which each construction is applied and  $(n, p)$  is the order of the finite semiring of weights. Finally the state complexity of the construction is a double exponential.

This work encompasses all locally finite semirings. It is an open question to know whether there exist in general a more efficient construction. For particular classes of semirings, this construction can certainly be improved.

### References

1. Birget, J.C.: Concatenation of inputs in a two-way automaton. *Theor. Comput. Sci.* **63**(2), 141–156 (1989). [https://doi.org/10.1016/0304-3975\(89\)90075-3](https://doi.org/10.1016/0304-3975(89)90075-3)
2. Carnino, V., Lombardy, S.: On determinism and unambiguity of weighted two-way automata. In: *AFL 2014. EPTCS*, vol. 151, pp. 188–200 (2014)

3. Engelfriet, J., Hoogeboom, H.J.: MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.* **2**(2), 216–254 (2001)
4. Leung, H.: Limitedness theorem on finite automata with distance functions: an algebraic proof. *Theor. Comput. Sci.* **81**(1, (Part A)), 137–145 (1991)
5. Lombardy, S.: Weighted two-way automata. In: NCMA 2015. books@ocg.at, vol. 318, pp. 37–47. Österreichische Computer Gesellschaft (2015)
6. Rabin, M.O., Scott, D.: Finite automata and their decision problems. *IBM J. Res. Dev.* **3**(2), 114–125 (1959). <https://doi.org/10.1147/rd.32.0114>
7. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.* **3**(2), 198–200 (1959). <https://doi.org/10.1147/rd.32.0198>





# A New Technique for Reachability of States in Concatenation Automata

Sylvie Davies<sup>(✉)</sup>

Department of Pure Mathematics,  
University of Waterloo, Waterloo, Canada  
sldavies@uwaterloo.ca

**Abstract.** We present a new technique for demonstrating the reachability of states in deterministic finite automata representing the concatenation of two languages. Such demonstrations are a necessary step in establishing the state complexity of the concatenation of two languages, and thus in establishing the state complexity of concatenation as an operation. Typically, ad-hoc induction arguments are used to show particular states are reachable in concatenation automata. We prove some results that seem to capture the essence of many of these induction arguments. Using these results, reachability proofs in concatenation automata can often be done more simply and without using induction directly.

## 1 Introduction

The *state complexity* of a regular language  $L$ , denoted  $\text{sc}(L)$ , is the least number of states needed to recognize the language with a deterministic finite automaton. The *state complexity of an operation* on regular languages is the worst-case state complexity of the result of the operation, expressed as a function of the maximal allowed state complexity of the input languages.

To establish the state complexity of an operation, there are two steps. First, one derives an upper bound on the state complexity. Next, one searches for witnesses, that is, families of languages which attain the upper bound. One must not only find these witnesses but also *prove* that the desired state complexity bound is reached. Such proofs are the subject of this paper.

We are interested in the case where the operation is *concatenation* of languages. We assume that one is working within some subclass of the regular languages, and has derived an upper bound  $f(m, n)$  for the worst-case state complexity of concatenation within this subclasses. We also assume one has found candidate witnesses for this upper bound, in the form of two sequences of languages  $(L_m : m \geq 1)$  and  $(K_n : n \geq 1)$  such that  $\text{sc}(L_m) \leq m$  and  $\text{sc}(K_n) \leq n$ . The goal is to prove that for each pair  $(m, n)$ , the concatenation  $L_m K_n$  has state complexity  $f(m, n)$ . We may divide such a proof into three steps:

1. Construct a deterministic automaton  $\mathcal{A}$  for  $L_m K_n$  in the standard way.
2. Show that  $\mathcal{A}$  contains at least  $f(m, n)$  reachable states.
3. Show exactly  $f(m, n)$  reachable states in  $\mathcal{A}$  are pairwise distinguishable.

We present a new technique for dealing with step (2) of this process. The standard way to construct a deterministic finite automaton  $\mathcal{A}$  for the concatenation of two languages yields an automaton in which the states are *sets*; to show a particular set is reachable, one typically proceeds by induction on the size of the set. We prove a result that seems to generalize many of these ad-hoc induction arguments, and can be used to establish reachability of sets without directly using induction. Additionally, we prove some helpful lemmas that make our main result easier to apply.

We have tested our technique by applying it to a variety of concatenation witnesses taken from the literature. The state complexity of concatenation has been studied in the class of all regular languages, as well as many subclasses. Table 1 lists some examples of subclasses that have been studied, and the state complexity of concatenation in each subclass. See the cited papers for definitions of each subclass and derivations/proofs of each complexity.

**Table 1.** Subclasses of regular languages and the state complexity of the concatenation operation within each subclass. **Bold** type indicates that the complexity grows exponentially in terms of  $n$ .

Subclass	Complexity	Subclass	Complexity
Regular [3, 9, 16, 20]	$(\mathbf{m} - 1)2^n + 2^{n-1}$	Prefix-closed [2, 9]	$(\mathbf{m} + 1)2^{n-2}$
Unary [17, 18, 20]	$\sim mn$ (asymptotically)	Prefix-free [9, 13, 15]	$m + n - 2$
Finite unary [10, 19]	$m + n - 2$	Suffix-closed [2, 8]	$mn - n + 1$
Finite binary [10]	$(\mathbf{m} - n + 3)2^{n-2} - 1$	Suffix-free [8, 14]	$(\mathbf{m} - 1)2^{n-2} + 1$
Star-free [7]	$(\mathbf{m} - 1)2^n + 2^{n-1}$	Right ideal [1, 6, 9]	$\mathbf{m} + 2^{n-2}$
Non-returning [5, 12]	$(\mathbf{m} - 1)2^{n-1} + 1$	Left ideal [1, 6, 8]	$m + n - 1$

If the state complexity of concatenation grows exponentially with  $n$  (indicated in Table 1 by **bold** type), it is typical to use an induction argument to prove the desired number of states is reachable. It is cases like this in which our technique is most likely to be useful. We selected 16 concatenation witnesses, all from subclasses in which the state complexity of concatenation is exponential in  $n$ , and tried to apply our technique to these witnesses. (In the interest of space, we present just two of these applications; the other 14 can be found in the arXiv version of this paper [11].) In many cases we were able to produce shorter and simpler proofs than the original authors, and we only found two cases in which our technique did not work or was not useful. This suggests that our technique is widely applicable and should be considered as a viable alternative to the traditional induction argument when attempting reachability proofs in concatenation automata.

The rest of the paper is structured as follows. Section 2 contains background material and definitions needed to understand the paper. Section 3 describes our new technique and proves the relevant results. Section 4 concludes the paper.

## 2 Preliminaries

### 2.1 Relations and Functions

A *binary relation*  $\rho$  between  $X$  and  $Y$  is a subset of  $X \times Y$ . If  $\rho \subseteq X \times Y$  and  $\tau \subseteq Y \times Z$ , the *composition* of  $\rho$  and  $\tau$  is the relation

$$\rho\tau = \{(x, z) \in X \times Z : \text{there exists } y \in Y \text{ such that } (x, y) \in \rho \text{ and } (y, z) \in \tau\}.$$

For  $x \in X$  and  $\rho \subseteq X \times Y$ , the *image* of  $x$  under  $\rho$  is the set  $x\rho = \{y \in Y : (x, y) \in \rho\}$ . For  $x \notin X$  we define  $x\rho = \emptyset$ . The *converse* of a binary relation  $\rho \subseteq X \times Y$  is the relation  $\rho^{-1} = \{(y, x) : (x, y) \in \rho\} \subseteq Y \times X$ . The set  $y\rho^{-1} = \{x \in X : (x, y) \in \rho\}$  is called the *preimage* of  $y$  under  $\rho$ . Elements of this set are called *preimages* of  $y$ ; for example, if  $x \in y\rho^{-1}$  we say that  $x$  is a preimage of  $y$ .

If we write  $\mathcal{P}(S)$  for the power set of a set  $S$  (that is, the set of all subsets of  $S$ ), then we can view  $\rho$  as a map  $\rho: X \rightarrow \mathcal{P}(Y)$ . We may also *extend*  $\rho$  *by union* to a map  $\rho: \mathcal{P}(X) \rightarrow \mathcal{P}(Y)$  as follows: for  $S \subseteq X$ , we define

$$S\rho = \bigcup_{s \in S} s\rho.$$

We thus have two ways to make sense of an expression like  $x\rho\tau$ : it is the image of  $x$  under the composite relation  $\rho\tau \subseteq X \times Z$ , and it is also the image of the set  $x\rho \subseteq Y$  under the map  $\tau: \mathcal{P}(Y) \rightarrow \mathcal{P}(Z)$ . Additionally, we have a way to make sense of a composition  $\rho\tau: X \rightarrow \mathcal{P}(Z)$  of maps  $\rho: X \rightarrow \mathcal{P}(Y)$  and  $\tau: Y \rightarrow \mathcal{P}(Z)$ : take the composition of the corresponding relations.

A *function*  $f: X \rightarrow Y$  is a binary relation  $f \subseteq X \times Y$  such that  $|xf| = 1$  for all  $x \in X$ . Following our notation for binary relations, we write functions to the *right* of their arguments. Composition of functions is defined by composing the corresponding relations. Thus the order of composition is *left-to-right*; in a composition  $fg$ , first  $f$  is applied and then  $g$ .

A *transformation* of a set  $X$  is a function  $t: X \rightarrow X$ , that is, a function from  $X$  into itself. We say  $t$  is a *permutation* of  $X$  if  $Xt = X$ . We say  $t$  *acts as a permutation* on  $S \subseteq X$  if  $St = S$ . If  $t$  acts as a permutation on  $S$ , then every element of  $S$  has at least one preimage under  $t$ , that is, for all  $s \in S$ , the set  $st^{-1} = \{x \in X : xt = s\}$  is non-empty.

A *cyclic permutation* of a set  $\{x_1, \dots, x_k\} \subseteq X$  is a permutation  $p$  such that  $x_i p = x_{i+1}$  for  $1 \leq i < k$ ,  $x_k p = x_1$ , and  $xp = x$  for all  $x \in X \setminus \{x_1, \dots, x_k\}$ . We denote such a permutation as  $(x_1, \dots, x_k)$ . A *transposition* is a cyclic permutation of a two-element set. We denote the identity transformation by *id*.

The notation  $(S \rightarrow x)$  for  $S \subseteq X$  and  $x \in X$  denotes a transformation that sends every element of  $S$  to  $x$  and fixes every element of  $S \setminus X$ . For example,  $(\{i\} \rightarrow j)$  maps  $i$  to  $j$  and fixes everything else, and  $(X \rightarrow x)$  is a constant transformation that maps every element of  $X$  to  $x$ . In the case where  $X = \{1, 2, \dots, n\}$ , the notation  $(\binom{j}{i} x \rightarrow x + 1)$  denotes a transformation such that for each  $x$  with  $i \leq x \leq j$ , the transformation sends  $x$  to  $x + 1$ , and every other  $x$  is fixed. For example, the transformation  $(\binom{n-1}{2} x \rightarrow x + 1)$  fixes 1, sends  $x$  to  $x + 1$  for  $2 \leq x \leq n - 1$ , and fixes  $n$ . The notation  $(\binom{j}{i} x \rightarrow x - 1)$  is defined similarly.

## 2.2 Automata

A *finite automaton* (FA) is a tuple  $\mathcal{A} = (Q, \Sigma, T, I, F)$  where  $Q$  is a finite set of *states*,  $\Sigma$  is a finite set of *letters* called an *alphabet*,  $T \subseteq Q \times \Sigma \times Q$  is a set of *transitions*,  $I \subseteq Q$  is a set of *initial states*, and  $F \subseteq Q$  is a set of *final states*.

We now define a binary relation  $T_w \subseteq Q \times Q$  for each  $w \in \Sigma^*$ . Define  $T_\varepsilon = \{(q, q) : q \in Q\}$ ; in terms of maps, this is the identity map on  $Q$ . For  $a \in \Sigma$ , define  $T_a = \{(p, q) \in Q \times Q : (p, a, q) \in T\}$ . For  $w = a_1 \cdots a_k$  with  $a_1, \dots, a_k \in \Sigma$ , define  $T_w = T_{a_1} \cdots T_{a_k}$ . The relation  $T_w$  is called the *relation induced by  $w$* . The set  $\{T_w : w \in \Sigma^*\}$  is a monoid under composition, called the *transition monoid* of  $\mathcal{A}$ . If  $w$  is a word but is *not* a word over  $\Sigma$ , we define  $T_w$  to be the empty relation. We sometimes write  $p \xrightarrow{w} q$  to mean  $q \in pT_w$ .

If  $\mathcal{A} = (Q, \Sigma, T, I, F)$  is a finite automaton such that  $|I| = 1$  and  $T_a$  is a *function* for each  $a \in \Sigma$ , we say  $\mathcal{A}$  is *deterministic*. We abbreviate “deterministic finite automaton” to DFA.

Let  $\mathcal{A} = (Q, \Sigma, T, I, F)$  be an FA. A word  $w \in \Sigma^*$  is *accepted* by  $\mathcal{A}$  if we have  $IT_w \cap F \neq \emptyset$ . If  $\mathcal{A}$  is a DFA with  $I = \{i\}$ , this condition becomes  $iT_w \in F$ . The *language* of  $\mathcal{A}$ , denoted  $L(\mathcal{A})$ , is the set of all words it accepts. A language recognized by an FA is called a *regular language*.

Given two regular languages  $L$  and  $K$  with DFAs  $\mathcal{A} = (Q^A, \Sigma^A, T^A, i^A, F^A)$  and  $\mathcal{B} = (Q^B, \Sigma^B, T^B, i^B, F^B)$ , we may construct an FA  $\mathcal{AB} = (Q, \Sigma, T, I, F)$  that accepts the concatenation  $LK$  as follows:

- $Q = Q^A \cup Q^B$ . We assume without loss of generality that  $Q^A \cap Q^B = \emptyset$ .
- $\Sigma = \Sigma^A \cup \Sigma^B$ .
- $T = T^A \cup T^B \cup \{(q, a, i^B) : qT_a^A \in F^A, a \in \Sigma^A\}$ .
- $I = \{i^A\}$  if  $i^A \notin F^A$ , and otherwise  $I = \{i^A, i^B\}$ .
- $F = F^B$ .

Next, we convert the FA  $\mathcal{AB} = (Q, \Sigma, T, I, F)$  to a DFA recognizing the same language. We apply the usual subset construction to obtain the DFA  $\mathcal{C} = (\mathcal{P}(Q), \Sigma, T^C, I, F^C)$ , where  $T^C = \{(S, a, ST_a) : S \subseteq Q, a \in \Sigma\}$ , and  $S \subseteq Q$  is in  $F^C$  if  $S \cap F \neq \emptyset$ . We call  $\mathcal{C}$  the *concatenation DFA* for  $\mathcal{A}$  and  $\mathcal{B}$ .

We make some observations and introduce some conventions to make it easier to work with the concatenation DFA.

- Since we are assuming  $\mathcal{A}$  and  $\mathcal{B}$  are DFAs, the only reachable states in  $\mathcal{C}$  have the form  $S^{\mathcal{A}} \cup S^{\mathcal{B}}$ , where  $S^{\mathcal{A}} \subseteq Q^{\mathcal{A}}$ ,  $S^{\mathcal{B}} \subseteq Q^{\mathcal{B}}$ , and  $|S^{\mathcal{A}}| \leq 1$ . Without loss of generality, we can assume the state set of  $\mathcal{C}$  consists of states of this form, rather than all of  $\mathcal{P}(Q)$ .
- We mark the states of  $\mathcal{A}$  with primes so they can be distinguished from the states of  $\mathcal{B}$ . So a variable named  $p$  or  $q$  generally means an element of  $Q^{\mathcal{B}}$ , while  $p'$  or  $q'$  means an element of  $Q^{\mathcal{A}}$ .
- We identify the set  $S^{\mathcal{A}} \cup S^{\mathcal{B}}$  with the ordered pair  $(S^{\mathcal{A}}, S^{\mathcal{B}})$ . Hence we can view the states of  $\mathcal{C}$  as these ordered pairs. Reachable states are either of the form  $(\emptyset, S)$  or  $(\{q'\}, S)$  with  $q' \in Q^{\mathcal{A}}$ ,  $S \subseteq Q^{\mathcal{B}}$ .
- For convenience, we frequently make no distinction between singleton sets and the elements they contain, and so write  $(q', S)$  rather than  $(\{q'\}, S)$ .
- Rather than  $T_w, T_w^{\mathcal{A}}$  and  $T_w^{\mathcal{B}}$ , we simply write  $w$  when it is clear from context which relation is meant. For example,  $(q', S)w$  means  $(q', S)T_w$  since  $(q', S)$  is a state of  $\mathcal{C}$ , and thus  $T_w$  is the natural relation to apply. From our convention for marking the states of  $\mathcal{A}$  and  $\mathcal{B}$  with primes, one can infer that  $q'w$  means  $q'T_w^{\mathcal{A}}$  and  $qw$  means  $qT_w^{\mathcal{B}}$ .
- Rather than  $i^{\mathcal{A}}$  and  $i^{\mathcal{B}}$ , let  $1'$  denote the initial state of  $\mathcal{A}$  and let  $1$  denote the initial state of  $\mathcal{B}$ . We also assume without loss of generality that  $Q^{\mathcal{A}} = \{1', 2', \dots, m'\}$  and  $Q^{\mathcal{B}} = \{1, 2, \dots, n\}$  for some  $m$  and  $n$ .

Under these conventions, the transitions of  $\mathcal{C}$  can be described as follows:

$$(q', S)a = \begin{cases} (\emptyset, Sa), & \text{if } a \in \Sigma^{\mathcal{B}} \setminus \Sigma^{\mathcal{A}}; \\ (q'a, \emptyset), & \text{if } a \in \Sigma^{\mathcal{A}} \setminus \Sigma^{\mathcal{B}} \text{ and } q'a \notin F^{\mathcal{A}}; \\ (q'a, 1), & \text{if } a \in \Sigma^{\mathcal{A}} \setminus \Sigma^{\mathcal{B}} \text{ and } q'a \in F^{\mathcal{A}}; \\ (q'a, Sa), & \text{if } a \in \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}} \text{ and } q'a \notin F^{\mathcal{A}}; \\ (q'a, Sa \cup 1), & \text{if } a \in \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}} \text{ and } q'a \in F^{\mathcal{A}}. \end{cases}$$

Recall that  $T_w^{\mathcal{A}}$  is the empty relation if  $w$  is not a word over  $\Sigma^{\mathcal{A}}$ , and a similar statement can be made for  $\mathcal{B}$ . Thus the transitions admit a simpler description:

$$(q', S)a = \begin{cases} (q'a, Sa \cup 1), & \text{if } a \in \Sigma^{\mathcal{A}} \text{ and } q'a \in F^{\mathcal{A}}; \\ (q'a, Sa), & \text{otherwise.} \end{cases}$$

### 2.3 State Complexity

We say a DFA  $\mathcal{A}$  is *minimal* if it has the least number of states among all DFAs that recognize  $L(\mathcal{A})$ . It is well known that each regular language has a unique minimal DFA (up to renamings of the states). The *state complexity* of a regular language  $L$ , denoted  $sc(L)$ , is the number of states in its minimal DFA.

The following characterization of minimality is useful. Let  $\mathcal{D} = (Q, \Sigma, T, i, F)$  be a DFA. A state  $q \in Q$  is *reachable* if  $iw = q$ . For  $p, q \in Q$ , we say  $q$  is *reachable from  $p$*  if  $pw = q$ . Two states  $p, q \in Q$  are *indistinguishable* if they are equivalent under the following equivalence relation:  $p \sim q$  if for all  $w \in \Sigma^*$ , we have

$pw \in F \iff qw \in F$ . Otherwise they are *distinguishable* by some word  $w$  such that  $pw \in F \iff qw \notin F$ . A DFA is minimal if and only if all of its states are reachable and pairwise distinguishable.

Let  $\circ$  be a binary operation on regular languages. The *state complexity of the operation*  $\circ$  is the following function, where  $m$  and  $n$  are positive integers:

$$(m, n) \mapsto \max\{\text{sc}(L \circ K) : \text{sc}(L) \leq m, \text{sc}(K) \leq n\}.$$

This is the worst-case state complexity of the result of the operation, expressed as a function of the maximal allowed state complexities of the input languages. When computing state complexity of operations we assume the input languages  $L$  and  $K$  are languages over the same alphabet. The case where the inputs are allowed to have different alphabets has been studied by Brzozowski [4]. An example of how to apply our results in this case is given in [11, Theorem 4].

### 3 Results

Let  $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma^{\mathcal{A}}, T^{\mathcal{A}}, 1', F^{\mathcal{A}})$  and  $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma^{\mathcal{B}}, T^{\mathcal{B}}, 1, F^{\mathcal{B}})$  be DFAs, with  $Q^{\mathcal{A}} = \{1', 2', \dots, m'\}$  and  $Q^{\mathcal{B}} = \{1, 2, \dots, n\}$  for positive integers  $m$  and  $n$ . Let  $\mathcal{C} = (Q, \Sigma, T, I, F)$  denote the concatenation DFA of  $\mathcal{A}$  and  $\mathcal{B}$  as defined in Sect. 2.2.

*Remark 1.* Let  $p', q' \in Q^{\mathcal{A}}$ , let  $X, Y, Z \subseteq Q^{\mathcal{B}}$ , and let  $w \in \Sigma^*$ . Then in  $\mathcal{C}$ , if  $(p', X)w = (q', Y)$ , then  $(p', X \cup Z)w = (q', Y \cup Zw)$ .

Indeed, recall that the pair  $(p', X)$  stands for the set  $\{p'\} \cup X$ . Thus  $(\{p'\} \cup X)w = \{p'w\} \cup Xw = \{q'\} \cup Y$ . It follows that  $p'w = q'$  and  $Xw = Y$ . Hence  $(\{p'\} \cup X \cup Z)w = \{p'w\} \cup Xw \cup Zw = \{q'\} \cup Y \cup Zw$ , which in our pair notation is  $(q', Y \cup Zw)$ . We will readily use this basic fact in proofs.

Before stating our main result formally, we give some motivating exposition. Fix a state  $s' \in Q^{\mathcal{A}}$  and a subset  $B$  of  $Q^{\mathcal{B}}$ . The state  $s'$  is called the *focus state* or simply *focus*; it is often taken to be the initial state  $1'$  but in general can be any state. The subset  $B$  is called the *base*. Fix a set  $T$  with  $B \subseteq T \subseteq Q^{\mathcal{B}}$ , called the *target*. Our goal is to give sufficient conditions under which starting from  $(s', B)$ , we can reach  $(s', S)$  for all sets  $S$  with  $B \subseteq S \subseteq T$ . That is, we can reach any state of the concatenation DFA  $\mathcal{C}$  in which the first component is the focus and the second component lies between the base and the target.

The idea is to first assume we can reach  $(s', B)$ , the state consisting of the focus and the base. Now, for  $q \in Q$ , define a *q-word* to be a word  $w$  such that  $(s', B)w = (s', B \cup q)$ . We can think of this as a word that “adds” the state  $q$  to the base  $B$ . Our next assumption is that we have a *q-word* for each state  $q$  in the target  $T$ . To reach a set  $S$  with  $B \subseteq S \subseteq T$ , we will repeatedly use *q-words* to add each missing element of  $S$  to the base  $B$ .

There is a problem with this idea, which we illustrate with an example. Suppose  $w_p$  is a *p-word* and  $w_q$  is a *q-word*, and we want to reach  $(s', B \cup \{p, q\})$ . Starting from  $(s', B)$  we may apply  $w_p$  to reach  $(s', B \cup p)$ . But now if we apply

$w_q$ , we reach  $(s', B \cup \{pw_q, q\})$ . There is no guarantee that we have  $pw_q = p$ , and in many cases we will not. What we should really do is find a state  $r$  such that  $rw_q = p$ , use an  $r$ -word to reach  $(s', B \cup r)$ , and then apply  $w_q$  to reach  $(s', B \cup \{p, q\})$ . But this idea only works if  $p$  has a preimage under  $w_q$ , which may not be the case.

We resolve this by making a technical assumption, which ensures that preimages will always exist when we attempt constructions like the above. First, define a *construction set* for the target  $T$  to be a set of words consisting of exactly one  $q$ -word for each  $q \in T$ . If  $W$  is a construction set for  $T$ , we write  $W[q]$  for the unique  $q$ -word in  $W$ .

We say a construction set is *complete* if there is a total order  $\prec$  on the target  $T$  such that for all  $p, q \in T$  with  $p \prec q$ , the state  $q$  has at least one preimage under the unique  $p$ -word  $W[p]$ , and at least one of these preimages lies in  $T$ . More formally, whenever  $p \prec q$ , the set  $qW[p]^{-1} = \{s \in Q^{\mathcal{B}} : sW[p] = q\}$  intersects  $T$  non-trivially. Our final assumption is that we have a complete construction set for  $T$ .

Note that the definition of a  $q$ -word depends not only on  $q$ , but also on  $s'$  and  $B$ . Since a construction set for  $T$  is a set of  $q$ -words, the definition of construction set also depends on  $s'$  and  $B$ . For simplicity, we omit this dependence on  $s'$  and  $B$  from the notation for  $q$ -words and construction sets.

We summarize the definitions that have just been introduced:

- Fix a state  $s' \in Q^{\mathcal{A}}$ , called the *focus*, and a set  $B \subseteq Q^{\mathcal{B}}$  called the *base*.
- For  $q \in Q^{\mathcal{B}}$ , a  *$q$ -word* is a word  $w$  such that  $(s', B)w = (s', B \cup q)$ .
- Given a *target* set  $T$  with  $B \subseteq T \subseteq Q^{\mathcal{B}}$ , a *construction set* for  $T$  is a set of words that contains exactly one  $q$ -word for each  $q \in T$ .
- The unique  $q$ -word in a construction set  $W$  is denoted by  $W[q]$ .
- A construction set for  $T$  is *complete* if there exists a total order  $\prec$  on  $T$  such that for all  $p, q \in T$  with  $p \prec q$ , we have

$$qW[p]^{-1} \cap T = \{s \in Q^{\mathcal{B}} : sW[p] = q\} \cap T \neq \emptyset.$$

Now, we state our main theorem, which formalizes the above construction.

**Theorem 1.** *Fix a state  $s' \in Q^{\mathcal{A}}$  and sets  $B$  and  $T$  such that  $B \subseteq T \subseteq Q^{\mathcal{B}}$ . If there is a complete construction set for  $T$ , then all states of the form  $(s', S)$  with  $B \subseteq S \subseteq T$  are reachable from  $(s', B)$  in  $\mathcal{C}$ . In particular, if  $(s', B)$  itself is reachable, then all states  $(s', S)$  with  $B \subseteq S \subseteq T$  are reachable.*

*Proof.* Note that if  $B \subseteq S \subseteq T$ , we can write  $S = R \cup B$  with  $R \cap B = \emptyset$  and  $R \subseteq T$ . Thus it suffices to show that all states of the form  $(s', R \cup B)$  with  $R \cap B = \emptyset$  and  $R \subseteq T$  are reachable from  $(s', B)$ . We proceed by induction on  $|R|$ . When  $|R| = 0$ , the only state of this form is  $(s', B)$  itself.

Now suppose every state  $(s', R \cup B)$  with  $R \cap B = \emptyset$ ,  $R \subseteq T$  and  $0 \leq |R| < k$  is reachable from  $(s', B)$ . We want to show this also holds for  $|R| = k$ . Let  $W$  be a complete construction set for  $T$  and let  $\prec$  be the corresponding total order on  $T$ . Let  $p$  be the minimal element of  $R$  under  $\prec$ . Let  $w$  be  $W[p]$ , the unique

$p$ -word in  $W$ . For all  $q \in R \setminus p$ , we have  $p \prec q$  and thus  $qw^{-1}$  contains an element of  $T$  (since  $W$  is complete).

Construct sets  $X$  and  $Y$  as follows: starting with  $X = \emptyset$ , for each  $q \in R \setminus p$ , choose an element of  $qw^{-1} \cap T$  and add it to  $X$ . Then set  $Y = X \setminus B$ . Observe that  $X$  is a subset of  $T$  of size  $|R \setminus p| = k - 1$ . Hence  $Y$  is a subset of  $T$  of size at most  $k - 1$  such that  $Y \cap B = \emptyset$ . It follows by the induction hypothesis that  $(s', Y \cup B)$  is reachable from  $(s', B)$ . But  $Y \cup B = X \cup B$ , so  $(s', X \cup B)$  is reachable from  $(s', B)$ . By the definition of  $X$ , we have  $Xw = R \setminus p$ . Since  $w$  is a  $p$ -word, we have  $(s', B)w = (s', B \cup p)$ , and thus

$$(s', X \cup B)w = (s', Xw \cup B \cup p) = (s', (R \setminus p) \cup B \cup p) = (s', R \cup B).$$

Hence  $(s', R \cup B)$  is reachable from  $(s', B)$ , as required.  $\square$

The definition of completeness is somewhat complicated, which makes it difficult to use Theorem 1. Thus, we prove some results giving simpler sufficient conditions for a construction set to be complete.

Before stating our first such result, we introduce some notation. Define  $\Sigma_0 = \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}}$ . We call  $\Sigma_0$  the *shared alphabet* of  $\mathcal{A}$  and  $\mathcal{B}$ . The following remark shows that when  $\Sigma^{\mathcal{A}} \neq \Sigma^{\mathcal{B}}$ , it is important to work exclusively with the shared alphabet when looking for complete construction sets. Of course, in the case where  $\Sigma^{\mathcal{A}} = \Sigma^{\mathcal{B}}$  there is nothing to worry about.

*Remark 2.* A construction set for a non-empty target cannot be complete unless it is a subset of  $\Sigma_0^*$ . To see this, suppose  $W$  is a construction set and let  $w \in W$ . If  $w$  contains a letter from  $\Sigma^{\mathcal{A}} \setminus \Sigma^{\mathcal{B}}$ , then  $w$  is not a word over  $\Sigma^{\mathcal{B}}$ . Recall that if  $w$  is not a word over  $\Sigma^{\mathcal{B}}$ , then  $T_w^{\mathcal{B}}$  is defined to be the empty relation. Thus the converse relation  $(T_w^{\mathcal{B}})^{-1}$  is also empty, which means  $qw^{-1}$  is empty for all  $q$ . It follows  $W$  cannot be complete. On the other hand, suppose  $w$  contains a letter from  $\Sigma^{\mathcal{B}} \setminus \Sigma^{\mathcal{A}}$ . Then  $(s', B)w = (\emptyset, Bw)$ . Hence  $w$  is not a  $q$ -word for any  $q$ , and so  $w$  cannot be an element of a construction set, which is a contradiction. Thus all words in a complete construction set are words over the shared alphabet  $\Sigma_0$ .

**Lemma 1.** *Fix  $s' \in Q^{\mathcal{A}}$  and sets  $B \subseteq T \subseteq Q^{\mathcal{B}}$ . Let  $x_1, \dots, x_j$  be words over  $\Sigma_0$  that act as permutations on  $T$ , and let  $y$  be an arbitrary word over  $\Sigma_0$ . Choose  $x_0 \in \{\varepsilon, x_1, \dots, x_j\}$ . Define  $W = \{x_1, x_2, \dots, x_j\} \cup \{x_0y, x_0y^2, \dots, x_0y^k\}$ . If  $W$  is a construction set for  $T$ , then it is complete.*

*Proof.* For  $1 \leq i \leq j$ , let  $w_i = x_i$ . For  $1 \leq i \leq k$ , let  $w_{j+i} = x_0y^i$ . Let  $\ell = j + k$ . Then we have  $W = \{w_1, \dots, w_\ell\}$ . Let  $q_i$  be the state in  $T$  such that  $(s', B)w_i = (s', B \cup q_i)$ . Define an order  $\prec$  on  $T$  so that  $q_1 \prec q_2 \prec \dots \prec q_\ell$ . We claim this order makes  $W$  complete. Notice that  $w_r = W[q_r]$ , the unique  $q_r$ -word in  $W$ . Thus we must show that whenever  $q_r \prec q_s$ , we have  $q_s w_r^{-1} \cap T \neq \emptyset$ .

Suppose  $r < s$  and  $r \leq j$ . Then  $w_r = x_r$  acts as a permutation on  $T$ . Thus  $q_s w_r^{-1} \cap T$  is non-empty, since  $q_s \in T$ .

Suppose  $r < s$  and  $r > j$ . Since  $s - r > 0$ , we can write  $w_s = x_0y^{s-j} = x_0y^{s-r}y^{r-j} = w_{j+s-r}y^{r-j}$ . Thus  $(s', B)w_s = (s', B \cup q_{j+s-r})y^{r-j} = (s', B \cup q_s)$ . There are two possibilities:  $q_{j+s-r}y^{r-j} = q_s$ , or  $qy^{r-j} = q_s$  for some  $q \in B$ .



In either case,  $q_s(y^{r-j})^{-1} \cap T$  is non-empty. That is, there exists  $q \in T$  such that  $qy^{r-j} = q_s$ . Since  $x_0$  acts as a permutation on  $T$ , there exists  $p \in T$  such that  $px_0 = q$ . Thus  $px_0y^{r-j} = pw_r = q_s$ . It follows that  $q_sw_r^{-1} \cap T$  is non-empty, as required.  $\square$

Usually, we will use one of the following corollaries instead of Lemma 1 itself.

**Corollary 1.** *Fix  $s' \in Q^A$  and sets  $B \subseteq T \subseteq Q^B$ . Let  $x$  and  $y$  be words over  $\Sigma_0$  such that  $x$  acts as a permutation on  $T$ . Suppose  $W$  is one of the following sets:*

1.  $\{y, y^2, \dots, y^k\}$ .
2.  $\{\varepsilon, y, y^2, \dots, y^k\}$ .
3.  $\{x, xy, xy^2, \dots, xy^k\}$ .
4.  $\{\varepsilon, x, xy, xy^2, \dots, xy^k\}$ .

*If  $W$  is a construction set for  $T$ , then it is complete.*

*Proof.* All statements follow easily from Lemma 1:

1. Set  $j = 0$ .
2. Set  $j = 1$  and  $x_0 = x_1 = \varepsilon$ .
3. Set  $j = 1$  and  $x_0 = x_1 = x$ .
4. Set  $j = 2$ ,  $x_1 = \varepsilon$  and  $x_0 = x_2 = x$ .  $\square$

**Corollary 2.** *Fix  $s' \in Q^A$  and sets  $B \subseteq T \subseteq Q^B$ . Let  $W \subseteq \Sigma_0^*$  be a construction set for  $T$ .*

1. *If every word in  $W$  acts as a permutation on  $T$ , then  $W$  is complete.*
2. *If there is a word  $w \in W$  such that every word in  $W \setminus w$  acts as a permutation on  $T$ , then  $W$  is complete.*

*Proof.* Both statements follow easily from Lemma 1:

1. Set  $k = 0$  in Lemma 1.
2. Set  $k = 1$ ,  $x_0 = \varepsilon$  and  $y = w$  in Lemma 1.  $\square$

In the special case where  $W$  contains  $\varepsilon$ , Corollary 2 admits the following generalization, which we found occasionally useful.

**Lemma 2.** *Fix  $s' \in Q^A$  and sets  $B \subseteq T \subseteq Q^B$ . Let  $W = \{\varepsilon, w_1, \dots, w_k\}$  be a construction set for  $T$ , where  $w_1, \dots, w_k$  are non-empty words over  $\Sigma_0$ . Suppose that for every word  $w \in W$ , there exists a set  $S$  with  $T \setminus B \subseteq S \subseteq T$  such that  $w$  acts as a permutation on  $S$ . Then  $W$  is complete.*

*Proof.* Write  $B = \{q_1, \dots, q_j\}$ . Note that  $\varepsilon$  is a  $q_i$ -word for  $1 \leq i \leq j$ . Thus by the definition of a construction set,  $\varepsilon$  is the unique  $q_i$ -word in  $W$  for each  $q_i \in B$ , that is,  $W[q_i] = \varepsilon$  for  $1 \leq i \leq j$ . In particular, each non-empty word

in  $W$  is a  $q$ -word for some  $q \in T \setminus B$ . For  $1 \leq i \leq k$ , let  $q_{j+i}$  be the state such that  $(s', B)w_i = (s', B \cup q_{j+i})$ . Then  $T = \{q_1, \dots, q_{j+k}\}$ . Note that  $W[q_i] = \varepsilon$  if  $1 \leq i \leq j$ , and  $W[q_i] = w_{i-j}$  if  $j+1 \leq i \leq j+k$ .

Define an order  $\prec$  on  $T$  by  $q_1 \prec q_2 \prec \dots \prec q_{j+k}$ . We claim this order makes  $W$  complete. Choose  $q_r, q_s \in T$  with  $q_r \prec q_s$ ; we want to show that  $q_s W[q_r]^{-1} \cap T \neq \emptyset$ . Suppose  $q_r \in B$ . Then  $W[q_r] = \varepsilon$ , and we have  $q_s \varepsilon^{-1} \cap T$  non-empty as required. Now if  $q_r \notin B$ , then since  $q_r \prec q_s$  we also have  $q_s \notin B$ . In this case,  $W[q_r] = w_{r-j}$ , which acts as a permutation on some superset  $S$  of  $T \setminus B$ . Since  $q_s \in T \setminus B$ , it follows that  $q_s$  has a preimage under  $w_{r-j}$ , and furthermore this preimage lies in  $T$ , since  $S$  is a subset of  $T$ . Thus  $q_s w_{r-j}^{-1} \cap T \neq \emptyset$  as required. This proves that  $W$  is complete.  $\square$

Note that *all words* referred to in the above lemmas and corollaries are words over  $\Sigma_0$ , the shared alphabet of  $\mathcal{A}$  and  $\mathcal{B}$ . When working with automata that have different alphabets, it is important to use only words over the shared alphabet when trying to find a complete construction set.

The following “master theorem” summarizes the results of this section. We have attempted to state this theorem so that it can be cited without having to first define all the notions introduced in this section, such as  $q$ -words and construction sets and completeness.

**Theorem 2.** *Let  $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma^{\mathcal{A}}, T^{\mathcal{A}}, i^{\mathcal{A}}, F^{\mathcal{A}})$  and  $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma^{\mathcal{B}}, T^{\mathcal{B}}, i^{\mathcal{B}}, F^{\mathcal{B}})$  be DFAs. Let  $\mathcal{C} = (Q, \Sigma, T, I, F)$  denote the concatenation DFA of  $\mathcal{A}$  and  $\mathcal{B}$ , as defined in Sect. 2.2. Let  $\Sigma_0 = \Sigma^{\mathcal{A}} \cap \Sigma^{\mathcal{B}}$ .*

*Fix a state  $s' \in Q^{\mathcal{A}}$  and sets  $B \subseteq T \subseteq Q^{\mathcal{B}}$ . Suppose that for each  $q \in T$ , there exists a word  $w_q \in \Sigma_0^*$  such that  $(s', B) \xrightarrow{w_q} (s', B \cup q)$  in  $\mathcal{C}$ . Let  $W = \{w_q : q \in T\}$ . Suppose that one of the following conditions holds:*

1. *There exist words  $x, y \in \Sigma_0^*$ , where  $x$  acts as a permutation on  $T$ , such that  $W$  can be written in one of the following forms:
 
  - $W = \{y, y^2, \dots, y^k\}$ .
  - $W = \{\varepsilon, y, y^2, \dots, y^k\}$ .
  - $W = \{x, xy, xy^2, \dots, xy^k\}$ .
  - $W = \{\varepsilon, x, xy, xy^2, \dots, xy^k\}$ .*
2. *Every word in  $W$  acts as a permutation on  $T$ .*
3. *There exists  $w \in W$  such that every word in  $W \setminus w$  acts as a permutation on  $T$ .*
4.  *$W$  contains  $\varepsilon$ , and for every non-empty word  $w \in W$ , there exists a set  $S$  such that  $T \setminus B \subseteq S \subseteq T$  and  $w$  acts as a permutation on  $S$ .*
5. *There exists a total order  $\prec$  on  $T$  such that for all  $p, q \in T$  with  $p \prec q$ , the set  $q w_p^{-1} = \{s \in Q^{\mathcal{B}} : s \xrightarrow{w_p} q\}$  contains an element of  $T$ .*

*If one of the above conditions holds, then every state of the form  $(s', X)$  with  $B \subseteq X \subseteq T$  is reachable from  $(s', B)$  in  $\mathcal{C}$ .*

To close this section, we give two examples of how our results can be applied. Many more examples can be found in [11].

**Theorem 3. (Yu, Zhuang and Salomaa, 1994 [20]).** Define  $\mathcal{A}$  and  $\mathcal{B}$  as follows:

	$a$	$b$	$c$	<i>Final States</i>
$\mathcal{A}$ :	$(1', \dots, m')$	$(Q^{\mathcal{A}} \rightarrow 1')$	id	$\{m'\}$
$\mathcal{B}$ :	id	$(1, \dots, n)$	$(Q^{\mathcal{B}} \rightarrow 2)$	$\{n\}$

Then  $\mathcal{C}$  has  $(m - 1)2^n + 2^{n-1}$  reachable and pairwise distinguishable states.

*Proof.* The initial state of  $\mathcal{C}$  is  $(1', \emptyset)$ . For  $k \leq n - 2$  we have

$$(1', \emptyset) \xrightarrow{a^m} (1', 1) \xrightarrow{b^k} (1', 1 + k).$$

It follows that  $\{a^m, a^m b, \dots, a^m b^{n-1}\}$  is a construction set for  $Q^{\mathcal{B}}$  (with  $s' = 1'$  and  $B = \emptyset$ ). By Corollary 1, it is complete (taking  $x = a^m$  and  $y = b$ ). Hence all states  $(1', S)$  with  $S \subseteq Q^{\mathcal{B}}$  are reachable. We can reach  $(q', S)$  for  $q' \neq m'$  and  $(m', S \cup 1)$  by words in  $a^*$ .

Let  $(p', S)$  and  $(q', T)$  be distinct states of  $\mathcal{C}$ . If  $S \neq T$ , let  $r$  be a state in the symmetric difference of  $S$  and  $T$ . Then  $b^{n-r}$  distinguishes the states. If  $S = T$  and  $p' < q'$ , then  $ca^{m-q}b^{n-2}$  distinguishes the states.  $\square$

**Theorem 4. (Maslov, 1970 [16]).** Define  $\mathcal{A}$  and  $\mathcal{B}$  as follows:

	$a$	$b$	<i>Final States</i>
$\mathcal{A}$ :	$(1', \dots, m')$	id	$\{m'\}$
$\mathcal{B}$ :	$(n - 1, n)$	$(\overset{n-1}{1} q \rightarrow q + 1)$	$\{n\}$

Then  $\mathcal{C}$  has  $(m - 1)2^n + 2^{n-1}$  reachable and pairwise distinguishable states.

*Proof.* The initial state is  $(1', \emptyset)$ . We have

$$(1', \emptyset) \xrightarrow{a^m} (1', 1) \xrightarrow{b^k} (1', 1 + k).$$

Thus  $\{a^m, a^m b, a^m b^2, \dots, a^m b^{n-1}\}$  is a construction set for  $Q^{\mathcal{B}}$  (with  $s' = 1'$  and  $B = \emptyset$ ). By Corollary 1, it is complete. Hence  $(1', S)$  is reachable for all  $S \subseteq Q^{\mathcal{B}}$ . We can reach  $(q', S)$  for  $q' \neq m'$  and  $(m', S \cup 1)$  by words in  $a^*$ .

Let  $(p', S)$  and  $(q', T)$  be distinct states of  $\mathcal{C}$ . If  $S \neq T$ , let  $r$  be a state in the symmetric difference of  $S$  and  $T$ . Then  $b^{n-r}$  distinguishes the states. If  $S = T$  and  $p' < q'$ , by  $b^n$  we reach  $(p', n)$  and  $(q', n)$ . Then by  $a^{m-q}$  we reach  $((p + m - q)', na^{m-q})$  and  $(m', na^{m-q} \cup 1)$ . These states differ in their second component, so they are distinguishable.  $\square$

## 4 Conclusions

We have introduced a new technique for demonstrating the reachability of states in DFAs for the concatenation of two regular languages, and tested this technique in a wide variety of cases. In addition to the examples of Theorems 3 and 4,

we demonstrate in [11] that our results are applicable to three other regular language concatenation witnesses [3, 4, 9], a star-free witness [7], two non-returning witnesses [5, 12], a prefix-closed witness [2], two suffix-free witnesses [8, 14], and three right ideal witnesses [1, 6, 9]. This suggests our technique is worth considering as an alternative to traditional induction proofs when working with concatenation automata.

**Acknowledgements.** I thank Jason Bell, Janusz Brzozowski and the anonymous referees for proofreading and helpful comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871.

## References

1. Brzozowski, J.A., Davies, S., Liu, B.Y.V.: Most complex regular ideal languages. *Discrete Math. Theoret. Comput. Sci.* **18**(3) (2016), paper #15
2. Brzozowski, J.A., Jirásková, G., Zou, C.: Quotient complexity of closed languages. *Theory Comput. Syst.* **54**, 277–292 (2014)
3. Brzozowski, J.A.: In search of most complex regular languages. *Int. J. Found. Comput. Sci.* **24**(06), 691–708 (2013)
4. Brzozowski, J.: Unrestricted state complexity of binary operations on regular languages. In: Câmpeanu, C., Manea, F., Shallit, J. (eds.) *DCFS 2016. LNCS*, vol. 9777, pp. 60–72. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-41114-9\\_5](https://doi.org/10.1007/978-3-319-41114-9_5)
5. Brzozowski, J.A., Davies, S.: Most complex non-returning regular languages. In: Pighizzini, G., Câmpeanu, C. (eds.) *DCFS 2017. LNCS*, vol. 10316, pp. 89–101. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-60252-3\\_7](https://doi.org/10.1007/978-3-319-60252-3_7)
6. Brzozowski, J.A., Jirásková, G., Li, B.: Quotient complexity of ideal languages. *Theoret. Comput. Sci.* **470**, 36–52 (2013)
7. Brzozowski, J.A., Liu, B.: Quotient complexity of star-free languages. *Int. J. Found. Comput. Sci.* **23**(06), 1261–1276 (2012)
8. Brzozowski, J.A., Sinnamon, C.: Complexity of left-ideal, suffix-closed and suffix-free regular languages. In: Drewes, F., Martín-Vide, C., Truthe, B. (eds.) *LATA 2017. LNCS*, vol. 10168, pp. 171–182. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-53733-7\\_12](https://doi.org/10.1007/978-3-319-53733-7_12)
9. Brzozowski, J.A., Sinnamon, C.: Complexity of right-ideal, prefix-closed, and prefix-free regular languages. *Acta Cybernetica* **23**(1), 9–41 (2017)
10. Câmpeanu, C., Culik, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) *WIA 1999. LNCS*, vol. 2214, pp. 60–70. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-45526-4\\_6](https://doi.org/10.1007/3-540-45526-4_6)
11. Davies, S.: A new technique for reachability of states in concatenation automata (2017). <https://arxiv.org/abs/1710.05061>
12. Eom, H.S., Han, Y.S., Jirásková, G.: State complexity of basic operations on non-returning regular languages. *Fundam. Inform.* **144**, 161–182 (2016)
13. Han, Y.S., Salomaa, K., Wood, D.: Operational state complexity of prefix-free regular languages. In: Ésik, Z., Fülöp, Z. (eds.) *AFL 2009*, pp. 99–115. University of Szeged, Hungary, Institute of Informatics (2009)
14. Han, Y.S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. *Theoret. Comput. Sci.* **410**(27–29), 2537–2548 (2009)

15. Jirásková, G., Krausová, M.: Complexity in prefix-free regular languages. In: McQuillan, I., Pighizzini, G., Trost, B. (eds.) DCFS 2010, pp. 236–244. University of Saskatchewan (2010)
16. Maslov, A.N.: Estimates of the number of states of finite automata. Dokl. Akad. Nauk SSSR 194, 1266–1268 (Russian). English translation: Soviet Math. Dokl. **11**(1970), 1373–1375 (1970)
17. Nicaud, C.: Average state complexity of operations on unary automata. In: Kutylowski, M., Pacholski, L., Wierzbicki, T. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 231–240. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48340-3\\_21](https://doi.org/10.1007/3-540-48340-3_21)
18. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal’s function. Int. J. Found. Comput. Sci. **13**(01), 145–159 (2002)
19. Yu, S.: State complexity of regular languages. J. Autom. Lang. Comb. **6**, 221–234 (2001)
20. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoret. Comput. Sci. **125**(2), 315–328 (1994)



# Forward Injective Finite Automata: Exact and Random Generation of Nonisomorphic NFAs

Miguel Ferreira, Nelma Moreira<sup>(✉)</sup>, and Rogério Reis

CMUP and DCC, Faculdade de Ciências da Universidade do Porto,  
Rua do Campo Alegre, 1021, 4169-007 Porto, Portugal  
miguelferreira108@gmail.com, {nam,rvr}@dcc.fc.up.pt

**Abstract.** We define the class of forward injective finite automata (FIFA) and study some of their properties. Each FIFA has a unique canonical representation up to isomorphism. Using this representation an enumeration is given and an efficient uniform random generator is presented. We provide a conversion algorithm from a nondeterministic finite automaton or regular expression into an equivalent FIFA. Finally, we present some experimental results comparing the size of FIFA with other automata.

## 1 Introduction

The study of the average-case complexity of determinisation of nondeterministic finite automata (NFAs) is an important research topic. In most algorithms that use NFAs, it is needed, in a way or another, to convert them to equivalent deterministic finite automata (DFAs), and that leads to an exponential blow up, in the worst-case. In practice, the feasibility of algorithms and representations, dealing with regular languages and related systems, depends primarily on their complexity on the average case, rather than the worst-case scenario which may rarely occur. The average-case analysis of operations in automata is, in general, a difficult task. One approach to this problem is to consider uniformly distributed random representations and to perform statistically significant experiments requiring most of the times nonisomorphic sampled automata. There are several uniform random generators available for nonisomorphic DFAs [1, 3, 4, 10]. For all these generators, DFAs are considered initially connected and it is possible to order their states in a canonical way to ensure that no two different isomorphic automata are generated. However for NFAs, the problem seems unfeasible in general as for  $n$ -state NFAs the size of the automorphism group can be  $n!$ , and this is polynomially equivalent to testing if two NFAs are isomorphic.

---

Authors partially funded by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MCTES) and European structural funds through the programs FEDER, under the partnership agreement PT2020.

Recently, Héam and Joly [6] presented uniform random generators for some classes of NFAs up to isomorphism, using Monte Carlo Markov-Chains modified by the Metropolis-Hastings algorithm, to deal with isomorphic objects [8]. This involves the computation of the sizes of automorphism groups of NFAs. However, the performance of these algorithms does not seem to allow the generation of large sets of objects of acceptable size. Considering classes of NFAs for which this latter problem is polynomial, the authors obtain (randomised/heuristic) uniform random generators in polynomial time. These classes include trim NFAs with a single initial state and with states with a fixed maximal output degree.

In this paper, we follow a different path and study a class of NFAs for which it is possible the exact and uniform random generation of nonisomorphic elements. More precisely, we consider a class of initially-connected NFAs (with a single initial state) for which it is possible to define a canonical order over the set of states. Starting with the initial state, whenever new states are reached from an already visited state, we ensure that the set of transition labels are pairwise distinct. Based on a natural order of the power set of the alphabet, an ordering for the set of states is obtained. This induces a unique canonical representation for the nonisomorphic NFAs for which it is possible to obtain such an order, and which are named *forward injective finite automata* (FIFA). We also developed an algorithm that converts an NFA into an equivalent FIFA and performed some experimental tests in order to compare their relative sizes. The class of FIFA represents all regular languages, FIFA sampling is proven to be efficient to implement, and experimental results suggest that, on average, for each NFA one can find a FIFA which is not much larger. These properties convinced the authors that FIFA is a class which deserves to be studied. In particular, FIFA based sampling is a possible alternative for average complexity analysis of algorithms that have NFAs as input. The paper is organised as follows. In the next section, some notation is given and we review the canonical string representation for initially-connected DFAs. In Sect. 3, we present the new class of automata (FIFA), a canonical representation that is unique for nonisomorphic automata of this class and give an enumeration of FIFAs by states and alphabetic size. A uniform random generator for FIFA is given in Sect. 4. In Sect. 5 we adapted the determinisation to the conversion from NFAs to FIFAs, together with some experimental results in Sect. 6. Some future work is discussed in Sect. 7.

## 2 Preliminaries

Given two integers,  $m$  and  $n$ , let  $[m, n]$  be the set  $\{i \in \mathbb{Z} \mid m \leq i \wedge i \leq n\}$ , and let one use the additional interval notation with its standard meaning.

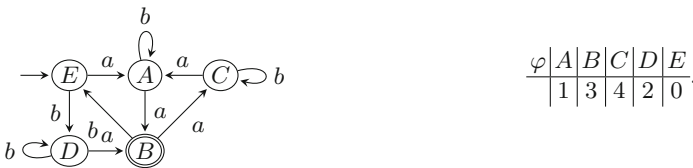
An NFA  $A$  is a tuple  $\langle Q, \Sigma, \delta, q_0, F \rangle$  where  $Q$  is a finite set of states,  $\Sigma$  the alphabet, *i.e.*, a nonempty finite set of symbols,  $\delta : Q \times \Sigma \rightarrow 2^Q$  is the transition function,  $q_0$  the initial state and  $F \subseteq Q$  the set of final states. As usual, let  $\delta(q) = \{p \mid \exists \sigma \in \Sigma, p \in \delta(q, \sigma)\}$ . We also consider  $\ell : Q \times Q \rightarrow 2^\Sigma$ , such that  $\ell(p, q) = \{\sigma \mid q \in \delta(p, \sigma) \wedge \sigma \in \Sigma\}$ , for any  $p, q \in Q$ , and for  $\emptyset \neq S \subseteq \Sigma$ , let  $\ell_p^{-1}(S) = \{q \mid \ell(p, q) = S\}$ . The function  $\ell$  allows us

to consider the amalgamation of all transitions with the same start and end states, and the value of  $\ell$  as label. In the rest of the paper, we will consider either  $\delta$  or  $\ell$  to refer to the transition function of a given NFA. The *size* of an NFA is its number of states,  $|Q| = n$ . An NFA is *initially connected* (or *accessible*) if for each state  $q \in Q$  there exists a sequence  $(q'_i)_{i \in [0, j]}$  of states and a sequence  $(\sigma_i)_{i \in [0, j-1]}$  of symbols, for some  $j < |Q|$ , such that  $q'_{m+1} \in \delta(q'_m, \sigma_m)$  for  $m \in [0, j[$ ,  $q'_0 = q_0$  and  $q'_j = q$ . The transition function  $\delta : Q \times \Sigma \rightarrow 2^Q$  extends naturally to  $\Sigma^*$  and to sets of states. The language accepted by  $A$  is  $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$ . Two automata are *equivalent* if they accept the same language. Two NFAs  $\langle Q, \Sigma, \delta, q_0, F \rangle$  and  $\langle Q', \Sigma, \delta', q'_0, F' \rangle$  are *isomorphic* if there is a bijection  $\iota : Q \rightarrow Q'$  such that  $\iota(q_0) = q'_0$ , for all  $\sigma \in \Sigma$  and  $q \in Q$ ,  $\iota(\delta(q, \sigma)) = \delta'(\iota(q), \sigma)$ , and  $\iota(F) = F'$ , where we naturally extend  $\iota$  to sets  $S \subseteq Q$ . In the following, we will identify an alphabet  $\Sigma$  of size  $k$  with  $[0, k[$  and the power set  $2^\Sigma$  with  $[0, 2^k[$  using a natural order for the subsets of  $\Sigma$ , with 0 corresponding to the emptyset and  $j \in [0, 2^k[$  corresponds to a set  $J \subseteq [0, k[$  if and only if  $i \in J$  then the  $i$ -th bit of the binary representation of  $j$  is 1. If two automata  $A$  and  $B$  are isomorphic, we write  $A \simeq B$ . A *semiautomaton*  $(Q, \Sigma, \delta, q_0)$  denotes an NFA without its final state information and is referred to as an  $\text{NFA}_\emptyset$ . Each semiautomaton, if  $|Q| = n$ , will be shared by  $2^n$  NFAs. An NFA is *deterministic* (DFA) if  $|\delta(q, \sigma)| \leq 1$ , for all  $(q, \sigma) \in Q \times \Sigma$  and is *complete* if the equality always holds. An initially-connected (complete) DFA is denoted by IC DFA and the corresponding semiautomaton by IC DFA $_\emptyset$ .

We review here the canonical string representation for nonisomorphic IC DFAs presented by Almeida *et al.* [1]. Given an IC DFA  $A = \langle Q, [0, k[, \delta, q_0, F \rangle$ , a canonical order for  $Q$  can be obtained by a breath-first traversal of  $A$ , starting with the initial state  $q_0$ , and in each state considering the transitions in the natural order of  $\Sigma$ . Let  $\varphi : Q \rightarrow [0, n[$  be the renaming of  $Q$  elements induced by that order. If we, by abuse of language, identify  $q$  and  $\varphi(q)$ , for  $q \in Q$ , a unique representation of the semiautomaton of  $A$  is  $S(A) = \langle \delta([i/k], i \bmod k)_{i \in [0, kn[} \rangle$ .

A canonical representation for  $A$  is obtained by adding a sequence  $(b_i)_{i \in [0, n[}$  of  $n$ -bits such that  $b_i = 1$  if state  $i$  is final, and  $b_i = 0$  otherwise.

*Example 1.* Consider the IC DFA,  $A = \langle \{A, B, C, D, E\}, \{a, b\}, \delta, E, \{B\} \rangle$ ,  $a < b$ , and  $\varphi$  the states renaming according to the induced order.



The canonical string  $S(A)$  for the corresponding IC DFA $_\emptyset$  is  $\underbrace{12}_{E} \underbrace{31}_{A} \underbrace{32}_D \underbrace{40}_B \underbrace{14}_C$ .

Adding the information pertaining to the final states as a sequence of bits, the canonical string for  $A$  is 123132401400010.



The characterisation of the canonical strings allows the enumeration, and the exact and random generation, as well as an optimal coding for ICDFAs, as defined by Lothaire [9]. Inspired by this model, we defined FIFA as NFAs having a unique breadth-first traversal order property according to the natural order of  $2^\Sigma$ .

### 3 Forward Injective Finite Automata

Even if one only considers initially-connected NFAs, with a single initial state, it is not possible to extend the previous representation to NFAs because it is not possible to induce an order for the set of states from the transition function. Starting with the initial state, whenever unvisited states are reached from an already visited state, we must ensure that the set of transition labels from the current state to the newly reached states are pairwise distinct. We denote NFAs with this property as *forward injective finite automata* (FIFA).

**Definition 1** *Let  $A = \langle Q, \Sigma, \delta, q_0, F \rangle$  be an initially-connected NFA and  $\Pi$  the bijection from  $2^\Sigma$  to  $[0, 2^k[$ , induced by the order on  $\Sigma$ . Consider the breadth-first search traversal of  $A$ , induced by the natural order on  $2^\Sigma$ , that starts in the initial state  $q_0$ . For each state  $s \in Q$ , let  $S(s)$  be the set of states that are image of a transition starting from a state already visited by the BFS. The labels for the transitions departing from  $s$  to any state in  $\delta(s) \setminus S(s)$  need to be unique. The automaton  $A$  is a forward injective finite automaton if it holds that:*

$$(\forall p, q \in \delta(s) \setminus S(s))(p \neq q \Rightarrow \ell(s, p) \neq \ell(s, q)). \quad (1)$$

This class of automata is expressive enough to recognise all regular languages, because deterministic automata trivially satisfy (1). However, not all (initially-connected) NFAs are FIFAs. Some experimental results suggest that for alphabets, of size at least 2, one can find a FIFA equivalent to an NFA that is not much larger than the NFA. These facts and the existence of a unique canonical representation show the interest in studying this class of NFAs. In particular, because of the existence of uniform random generators, one can use this model to obtain estimates of average performance of algorithms that manipulate NFAs.

#### 3.1 A Canonical State Order for FIFAs

Given a FIFA it is possible to obtain a canonical state order  $\varphi$  through a breadth first traversal starting in the initial state and ordering the newly reached states according to the total order defined in  $2^\Sigma$ . For that, one can disregard the set of final states and consider the semiautomaton  $\text{FIFA}_\emptyset$ . The canonical state order for a  $\text{FIFA}_\emptyset$  can be computed through Algorithm 1, where  $\Pi$  is the bijection from  $2^\Sigma$  to  $[0, 2^k[$ , *sorted* is a function that sorts integers in increasing order, and  $\varphi : Q \rightarrow [0, n[$  is the computed bijection. Note that (at line 7)  $\ell_{\varphi^{-1}(s)}^{-1}(\Pi^{-1}(j))$  is a single value by the injectivity of  $\ell$  restricted to the newly seen states in a FIFA.

---

**Algorithm 1.** FIFA state order algorithm

---

```

1: procedure STATEORDER(FIFA∅ ⟨Q, Σ, δ, q0⟩)
2:   φ(q0) ← 0
3:   i ← 0; s ← 0
4:   do
5:     M ← sorted{Π(S) | ∅ ≠ S = ℓ(φ-1(s), q) ∧ q ∈ Q \ φ-1([0, i])}
6:     for j ∈ M do
7:       φ(ℓφ-1(s)}-1(Π-1(j))) ← i + 1
8:       i ← i + 1
9:     s ← s + 1
10:  while s < i
      return φ

```

---

**Proposition 1** *Let  $A = \langle Q, \Sigma, \delta, q_0 \rangle$  be a FIFA<sub>∅</sub> with  $n$  states and  $k = |\Sigma|$ , there is a bijection  $\varphi : Q \rightarrow [0, n[$  that defines an isomorphism between  $A$  and  $\langle [0, n[, \Sigma, \delta', 0 \rangle$  with  $\delta'(i, \sigma) = \{\varphi(s) \mid s \in \delta(\varphi^{-1}(i), \sigma)\}$ , for  $i \in [0, n[$  and  $\sigma \in \Sigma$ .*

Throughout the paper we will now consider FIFA<sub>∅</sub> to have its states in their canonical order:  $A = \langle [0, n[, \Sigma, \delta, 0 \rangle$ .

### 3.2 Canonical String Representation

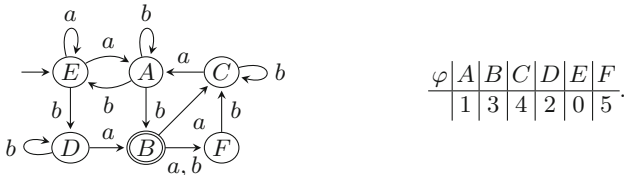
Let  $A = \langle [0, n[, \Sigma, \delta, 0, F \rangle$  be a FIFA such that  $\langle [0, n[, \Sigma, \delta, 0 \rangle$  is a FIFA<sub>∅</sub>. We can represent  $A$  by the canonical representation of its FIFA<sub>∅</sub> concatenated with the bitmap of the state finalities. The canonical representation of a FIFA<sub>∅</sub> is defined as follows.

**Definition 2** *Given a FIFA<sub>∅</sub>  $\langle [0, n[, \Sigma, \delta, 0 \rangle$  with  $|\Sigma| = k$ , its canonical representation is a sequence  $(r_i)_{i \in [0, n[}$  such that for each state  $i$ ,*

$$r_i = s_{i,1} s_{i,2} \dots s_{i,m_i} u_{i,1} \dots u_{i,\bar{m}_i},$$

and where  $m_i$  is the number of previously seen states,  $\bar{m}_i$  is the number of newly seen states,  $s_{i,j} = \Pi(\ell(i, j - 1))$  for  $j \in [1, m_i]$ , and  $u_{i,j} = \Pi(\ell(i, m_i + j - 1))$  for  $j \in [1, \bar{m}_i]$ . This means that, for each state  $i$ ,  $s_{i,j}$  correspond to the sets of transitions to states already seen (back transitions) and  $u_{i,j}$  correspond to the sets of transitions to newly seen states from state  $i$  (forward transitions).

*Example 2.* Consider the following FIFA on the left.



Let  $\Pi(\emptyset) = 0$ ,  $\Pi(\{a\}) = 1$ ,  $\Pi(\{b\}) = 2$  and  $\Pi(\{a, b\}) = 3$ . The state renaming according to the induced order on the states is given above. The canonical string  $(r_i)_{i \in [0,5]}$  for the corresponding  $\text{FIFA}_\emptyset$  is

$$\underbrace{[1][1, 2]}_E \underbrace{[2, 2, 0][2]}_A \underbrace{[0, 0, 2, 1]}_D \underbrace{[0, 0, 0, 0][1, 3]}_B \underbrace{[0, 1, 0, 0, 2, 0]}_C \underbrace{[0, 0, 0, 0, 2, 0]}_F,$$

where the transitions for each state are as indicated. The  $\text{FIFA}$  can be represented by its semiautomaton canonical string with the state finalities appended. Thus, this  $\text{FIFA}$  canonical string is

$$[1][1, 2][2, 2, 0][2][0, 0, 2, 1][0, 0, 0, 0][1, 3][0, 1, 0, 0, 2, 0][0, 0, 0, 0, 2, 0][0, 0, 0, 1, 0, 0].$$

**Lemma 3.** *Let  $A = \langle [0, n[, \Sigma, \delta, 0 \rangle$  be a  $\text{FIFA}_\emptyset$  with  $k = |\Sigma|$ . Let  $(r_i)_{i \in [0, n[}$  with  $r_i = s_{i,1}s_{i,2} \cdots s_{i,m_i}u_{i,1}u_{i,2} \cdots u_{i,\overline{m}_i}$  be the canonical representation for  $A$  as given above. Then the following rules are satisfied:*

$$s_{i,j} \in [0, 2^k[, \quad \forall i \in [0, n[, \forall j \in [1, m_i], \quad (\text{F1})$$

$$u_{i,j} \in [1, 2^k[, \quad \forall i \in [0, n[, \forall j \in [1, \overline{m}_i], \quad (\text{F2})$$

$$j < l \Rightarrow u_{i,j} < u_{i,l}, \quad \forall i \in [0, n[, \forall j, l \in [1, \overline{m}_i], \quad (\text{F3})$$

$$m_0 = 1, \quad (\text{F4})$$

$$m_i = m_{i-1} + \overline{m}_{i-1}, \quad \forall i \in [1, n[, \quad (\text{F5})$$

$$i < m_i \leq n, \quad \forall i \in [1, n[, \quad (\text{F6})$$

$$\overline{m}_{n-1} = 0. \quad (\text{F7})$$

*Proof.* The rule (F1) describes how transitions to previously seen states are represented, i.e.  $s_{i,j} = \Pi(\ell(i, j - 1))$ , possibly with  $s_{i,j} = 0$ . The rule (F2) considers the transitions from state  $i$  to states  $t_{i,j} = m_i + j - 1$  visited for the first time in  $i$ , which implies that  $\ell(i, t_{i,j}) \neq \emptyset$ . Consequently,  $u_{i,j} \in [1, 2^k[$ . For representation purposes, rule (F3) states that the set of states visited for the first time is represented with its transitions sorted in ascending order. This is a representation choice that ensures that all  $u_{i,j}$  are distinct. Rule (F4) is obvious as one starts at state 0 and thus 0 is the only seen state. Rule (F5) is a direct consequence of the definition of  $m_i$  in (2), and implies that  $m_i = 1 + \sum_{j=0}^{i-1} \overline{m}_j$  for  $i \in [1, n[$ . Rules (F6) and (F7) ensures that all states are seen, and that the  $\text{FIFA}_\emptyset$  is initially connected. It is a consequence of the definition of  $\varphi$  and also ensures that a state must be seen before its representation is given. Rule (F6) implies that  $m_{n-1} = n$ . By contradiction, suppose that there is  $i$  such that  $m_i \leq i$ , for  $i \in [1, n[$ . Then, there exist  $1 \leq j \leq i$  such that  $j$  is not accessible from 0 in paths that use states only in  $[0, i[$ . But that contradicts  $\varphi$  definition.  $\square$

**Lemma 4.** *Every string  $(r_i)_{i \in [0, n[}$  satisfying rules (F1)–(F7) represents a  $\text{FIFA}_\emptyset$  with states  $[0, n[$  over an alphabet of  $k$  symbols.*

*Proof.* There is at least one transition reaching each state in  $[1, n[$  and there is at least one transition from the state 0 to state 1. The transition function is defined

by  $\ell(i, j - 1) = \Pi^{-1}(s_{i,j})$  for  $i \in [0, n[$  and  $j \in [1, m_i]$ , and  $\ell(i, m_i + j - 1) = \Pi^{-1}(u_{i,j})$  for  $i \in [0, n[$  and  $j \in [1, \overline{m}_i]$ . The proof that the  $\text{FIFA}_\emptyset$  is initially connected is analogous to the one in Lemma 3.  $\square$

From these lemmas the following theorem holds.

**Theorem 5.** *For each  $n > 0$  and  $k > 0$ , there is a one-to-one mapping from sequences  $(r_i)_{i \in [0, n[}$  satisfying rules (F1)–(F7) and nonisomorphic  $\text{FIFA}_\emptyset$ s with  $n$  states over an alphabet of  $k$  symbols.*

The canonical form for  $\text{FIFA}_\emptyset$  is not a simple extension of the one for  $\text{ICDFA}_\emptyset$ s reviewed in Sect. 2 for several reasons. One needs to consider instead of the alphabet its power set, there are no restrictions for transitions to already seen states, and transitions to newly seen states must have different labels. However, the first occurrences of each state satisfy exactly the same rules (over an alphabet of  $2^k$  symbols) observed in the canonical representation of  $\text{ICDFA}_\emptyset$ s. This will be made evident in the next section.

### 3.3 Counting FIFAs

Our aim is to enumerate (exactly generate) and count all the nonisomorphic  $\text{FIFA}_\emptyset$  with  $n$  states and  $k$  symbols. This will also allow us to obtain a uniform random generator for the class of FIFAs. Let  $\Psi : [0, n[ \times [1, 2^k[ \rightarrow [0, n(2^k - 1)[$ , be defined by  $\Psi(i, j) = i(2^k - 1) + j - 1$ . The mapping  $\Psi$  is a bijection with  $\Psi^{-1}(p) = (\lfloor p / (2^k - 1) \rfloor, (p \bmod (2^k - 1)) + 1)$ . Let  $(r_i)_{i \in [0, n[}$  be a sequence satisfying rules (F1)–(F7), thus, representing a  $\text{FIFA}_\emptyset$ . Let us denote by *flag of a state*  $t \in [1, n[$  the pair  $(i, u_{i,j})$ , occurring in state  $i$ , such that  $t = m_i + j - 1$  (and  $\ell(i, t) = \Pi^{-1}(u_{i,j})$ ). According to (F3), if in a state  $i$  two different flags  $(i, u_{i,j})$  and  $(i, u_{i,l})$  occur, we know that  $j < l \Rightarrow u_{i,j} < u_{i,l}$ . For the sake of readability, given  $t \in [1, n[$ , we denote by  $(i_t, u_t)$  its flag, and let  $\Psi(i_t, u_t) = f_t$ . Then, by (F3), one has

$$(\forall t \in [2, n[)(i_t = i_{t-1} \Rightarrow u_t > u_{t-1}) \vee (i_t > i_{t-1}),$$

which implies

$$(\forall t \in [2, n[)(f_t > f_{t-1}), \tag{G1}$$

$$(\forall t \in [1, n[)(f_t < t(2^k - 1)). \tag{G2}$$

Rules (G1)–(G2) are satisfied by the positions of the first occurrence of a state in the canonical strings for  $\text{ICDFA}_\emptyset$ s, considering  $k$  instead of  $2^k - 1$  in rule (G2). The following theorem computes the number of allowed sequences of flags.

**Proposition 2 (Theorem 6 of [1])** *Given  $k > 0$  and  $n > 0$ , the number of sequences  $(f_t)_{t \in [1, n[}$ ,  $F_{2^k-1, n}$ , is given by:*

$$F_{2^k-1, n} = \sum_{f_1=0}^{2^k-1-1} \sum_{f_2=f_1+1}^{2(2^k-1)-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{(2^k-1)(n-1)-1} 1 = C_n^{(2^k-1)},$$

where  $C_n^{(2^k-1)} = \binom{n(2^k-1)}{n}_{(2^k-2)n+1}$  are the generalized Fuss-Catalan numbers.

*Example 6.* For the FIFA of Example 2,  $((0, 1), (0, 2), (1, 2), (3, 1), (3, 3))$  is the sequence of flags and  $(f_t)_{t \in [1,5]} = (0, 1, 4, 9, 11)$ .

Given a sequence of flags  $(f_t)_{t \in [1,n]}$ , the set of possible canonical strings that represent  $FIFA_\emptyset$ s can be easily enumerated: each state  $i$  has unconstrained transitions for states already seen ( $m_i$ ) and has the transitions to new states given by the flags occurring in its description (forward transitions).

The number of canonical strings with a given sequence of flags is given by

$$\prod_{i \in [0,n[} (2^k)^{m_i}. \tag{2}$$

**Theorem 7.** *The total number of  $FIFA_\emptyset$ s with  $n$  states over a  $k$ -ary alphabet is*

$$b_{k,n} = \sum_{f_1=0}^{2^k-1} \sum_{f_2=f_1+1}^{2(2^k-1)-1} \dots \sum_{f_{n-1}=f_{n-2}+1}^{(2^k-1)(n-1)-1} \prod_{i \in [0,n[} (2^k)^{m_i},$$

where  $m_i = 1 + \sum_{j=0}^{i-1} \bar{m}_j$  and  $\bar{m}_j = |\{f_t \mid i_t = j\}|$  for  $i \in [0, n[$  and  $j \in [1, n[$ .

This can be adapted for the exact generation/enumeration of all canonical representations. Each  $FIFA_\emptyset$  corresponds to a number between 1 and  $b_{k,n}$ . In Table 1 we present the values of  $b_{k,n}$  for  $n \in [2, 7]$  and  $k \in [2, 3]$ . An equivalent recursive definition for  $b_{k,n}$  is given in the next section for uniform random generation.

**Table 1.** Values of  $b_{k,n}$

$n$	$k = 2$	$k = 3$
2	192	3584
3	86016	56885248
4	321912832	32236950781952
5	10382009696256	738091318939425439744
6	3073719939819896832	733871593861464877408622477312
7	8715818304405159932854272	32686722749179979231494144786993701191680

**Corollary 8.** *The number of nonisomorphic FIFAs with  $n$  states and  $k$  alphabetic symbols is  $B_{k,n} = b_{k,n}2^n$ .*

## 4 Uniform Random Generation

The canonical representation for FIFAs allows an easy uniform random generation for this class of automata. Given the number of flags occurring in a prefix of a canonical string we count the number of valid suffixes. To count the number of automata with a given prefix a recursive counting formula for  $FIFA_\emptyset$  is needed.

With these partial values, we can reconstruct any  $\text{FIFA}_\emptyset$  by knowing its number, which varies from 1 to  $b_{k,n}$ . The process of uniform randomly generating a FIFA consists, thus, in four steps: creation of a table with partial counts for each prefix; uniformly sample a number between 1 and  $b_{k,n}$ ; construct the  $\text{FIFA}_\emptyset$  representation using the table; random generation of values from 0 to  $2^n - 1$  for the state finalities and return the FIFA. Let  $m$  be the number of already seen states for the state  $i$  of a canonical string of a  $\text{FIFA}_\emptyset$ . We count the number  $N_{m,i}$  of  $\text{FIFA}_\emptyset$ s for each value of  $m$  and  $i$ . This gives us the following recursive formula for fixed  $n$  and  $k$ :

$$\begin{aligned} N_{m,i} &= (2^k)^m \sum_{j=0}^{n-m} \binom{2^k-1}{j} N_{m+j,i+1}, & m \in [1, n], i \in [0, m[, \\ N_{m,i} &= 0, & m \in [1, n], i \notin [0, m[, \\ N_{n,n} &= 1. \end{aligned}$$

**Proposition 3**  $b_{k,n} = N_{1,0}$ , for all  $k \geq 1$  and  $n \geq 1$ .

*Proof.* Immediate consequence of the canonical representation and Theorem 7.  $\square$

---

**Algorithm 2.** Random  $\text{FIFA}_\emptyset$  algorithm.

---

```

1: procedure RANDOMFIFA( $n, k$ )
2:    $r \leftarrow \text{Random}(0, N_{1,0} - 1)$  ▷ number of the  $\text{FIFA}_\emptyset$ 
3:    $m_0 \leftarrow 1$ 
4:   for  $q \in [0, n[$  do ▷ reconstruct  $\text{FIFA}_\emptyset$  flags
5:      $\overline{m}_q \leftarrow 0$ 
6:     while  $N_{m_q + \overline{m}_q, q} \leq r$  do
7:        $\overline{m}_q \leftarrow \overline{m}_q + 1$ 
8:        $m_{q+1} \leftarrow m_q + \overline{m}_q$ 
9:        $b \leftarrow r \bmod (2^k)^{\sum_{i=0}^{n-1} m_i}$  ▷ number representing back transitions
10:       $f \leftarrow r / (2^k)^{\sum_{i=0}^{n-1} m_i}$  ▷ number representing forward transitions
11:      for  $q \in [0, n[$  do
12:        for  $p \in [1, m_q]$  do ▷ reconstruct back transitions
13:           $s_{q,p} \leftarrow b \bmod 2^k$ 
14:           $b \leftarrow b / 2^k$ 
15:          if  $\overline{m}_q \neq 0$  then
16:             $c \leftarrow \binom{2^k-1}{\overline{m}_q}$ 
17:             $t \leftarrow f \bmod c$ 
18:             $f \leftarrow f / c$ 
19:            for  $p \in [1, \overline{m}_q]$  do
20:               $u_{q,p} \leftarrow t \bmod (2^k - 1)$ 
21:               $t \leftarrow \lfloor t / (2^k - 1) \rfloor$ 
22:      return  $(s_{i,1} s_{i,2} \cdots s_{i,m_i} u_{i,1} u_{i,2} \cdots u_{i,m_i})_{i \in [0, n[}$ 

```

---

**Proposition 4** *Algorithm 2* presents a uniform random generator for a  $\text{FIFA}_\emptyset$  with  $n$  states and  $k$  symbols.

*Proof.* (Sketch) Given an arbitrary integer  $r$  representing a  $\text{FIFA}_\emptyset$  we determine the values  $\bar{m}_i$  for each  $i \in [0, n[$ , using the precalculated table  $N_{m,i}$ . For each state  $i$  and  $m$  previously seen states,  $\bar{m}_i$  is the first value such that  $N_{m+\bar{m}_i,i} > r$ , that is,  $N_{m+\bar{m}_i-1,i} \leq r < N_{m+\bar{m}_i,i}$  (lines 4–8). Then, we count the number of available back transitions in each state  $i$  using  $m_i$  which can be determined by  $\bar{m}_i$  (by F5). The total number of possible back transitions is  $\sum_{i=0}^{n-1} m_i$ . With this number (lines 9 and 10) one obtains the integers that represent all the  $s_{i,j}$  and the  $u_{i,j}$ , respectively. For a given  $i \in [0, n[$ , in lines 12–14 we calculate  $s_{i,j}$  for  $j \in [1, m_i]$ . If state  $i$  has forward transitions, their values are computed in lines 16–21.  $\square$

To obtain a random  $\text{FIFA}$  from the  $\text{FIFA}_\emptyset$  we can generate a random number from  $[0, 2^n[$  and reconstruct the state finalities according to the corresponding choice. Using dynamic programming techniques it is possible to generate a table indexed by the values of  $m$  and  $i$  ( $N_{m,i}$ ) with time complexity  $O(n^3 \log((2^k)^{n^2})) = O(n^5 k)$ . The amount of memory used is  $O(n^4 k)$ , and this is a limiting factor for the dimension of the  $\text{FIFA}_\emptyset$  being generated. This is justified by the huge number of  $\text{FIFA}_\emptyset$ s for a given  $n$  and  $k$ . For example,  $b_{2,100}$  is greater than  $10^{11531}$ . In Table 2 we present the execution times for the generation of 10000  $\text{FIFA}$ s for  $n \in \{1, 20, 30, 50, 75, 100\}$  and  $k \in \{1, 2, 3, 4\}$ , using Python 2.7 interpreted by Pypy, with a Intel Xeon CPU X5550 at 2.67 GHz. Comparing with some experiments presented by Héam and Joly [6, Table 1], these times correspond, approximately, to the generation of a single NFA.

**Table 2.** Execution times for the generation of 10000 random  $\text{FIFA}$ .

Times	$k = 1$	$k = 2$	$k = 3$	$k = 4$
$n = 10$	0.77 s	1.05 s	0.95 s	8.59 s
$n = 20$	1.06 s	2.33 s	3.13 s	3.96 s
$n = 30$	1.15 s	5.01 s	7.38 s	9.52 s
$n = 50$	2.84 s	16.86 s	26.64 s	40.43 s
$n = 75$	7.11 s	47.62 s	71.92 s	91.70 s
$n = 100$	15.86 s	100.25 s	156.24 s	202.41 s

## 5 Converting an NFA into a $\text{FIFA}$

In this section we discuss a process of converting an arbitrary NFA to a  $\text{FIFA}$  and the asymptotic time complexity bounds of such a procedure. The algorithm has an NFA as input and outputs an equivalent  $\text{FIFA}$ . It is based on the subset construction for NFA determinisation, with addition of an heuristic that attempts to create back transitions whenever possible. This gives us a  $\text{FIFA}$  that is not only forward injective but also forward deterministic. It may be also possible to add an heuristic for nondeterministic forward injective transitions or to have

other procedures that are not based on the subset construction. However this one had a good performance in our experiments.

**Proposition 5** *The procedure NFATOFIFA in Algorithm 3 computes a FIFA equivalent to a given NFA.*

This algorithm has time complexity  $O(|\Sigma|2^{2|Q|})$ , which is justified by  $|Q'|$  having space complexity  $O(2^{|Q|})$  due to the determinisation based algorithm. It is an open problem whether these bounds are tight for this algorithm.

---

**Algorithm 3.** An NFA to FIFA algorithm

---

```

1: procedure NFATOFIFA(NFA  $\langle Q, \Sigma, \delta, I, F \rangle$ )
2:    $Q' \leftarrow \{I\}$ 
3:    $w \leftarrow \{I\}$  ▷ states to be processed
4:   while  $w \neq \emptyset$  do
5:      $S \leftarrow \text{POP}(w)$  ▷ popping  $S$  from  $w$ 
6:     for  $\sigma \in \Sigma$  do
7:        $P \leftarrow \emptyset$ 
8:        $\delta'(S, \sigma) \leftarrow \emptyset$ 
9:       for  $s \in S$  do
10:         $P \leftarrow P \cup \delta(s, \sigma)$ 
11:        for  $R \in \text{SORTEDBYSIZEDESC}(Q')$  do
12:          if  $R \subseteq P$  then ▷ nondeterministic back transitions
13:             $\delta'(S, \sigma) \leftarrow \delta'(S, \sigma) \cup \{R\}$ 
14:             $P \leftarrow P \setminus R$ 
15:          if  $P \neq \emptyset$  then ▷ forward transition
16:             $Q' \leftarrow Q' \cup \{P\}$ 
17:             $w \leftarrow w \cup \{P\}$ 
18:             $\delta'(S, \sigma) \leftarrow \delta'(S, \sigma) \cup \{P\}$ 
19:   return FIFA  $\langle Q', \Sigma, \delta', I, \{S \cap F \neq \emptyset, S \in Q'\} \rangle$ 

```

---

## 6 Experimental Results

The algorithm defined in the previous section was implemented within the FAdo package [5]. We performed some experiments to compare the sizes of the input and output automata. The input NFAs were obtained from uniform random generated regular expressions, for a fixed (standard) grammar, of a given syntactic tree size  $m$  over an alphabet of  $k$  symbols. The conversion method used was the partial derivative automata [2]. For each  $m$  and  $k$ , 10000 random regular expressions were generated to ensure a 95% confidence level within a 1% error margin [7, pp. 38–41]. For each sample, we calculated the minimal, the average and the maximum sizes of the obtained automata. For each partial derivative automaton (PD) we applied the algorithm NFATOFIFA and obtained a FIFA (FIFA). We also computed the DFA obtained by determinisation of PD, by the usual subset construction, (DT), and the minimal DFA (MIN). Results for



$m \in \{50, 100, 250, 500\}$  and  $k \in \{2, 3, 10\}$  are presented in Table 3. In general the FIFA computed is not much larger than the PD, although the determinised automata can be significantly larger.

**Table 3.** State complexities of automata where  $m$ , syntactic size of RE;  $k$ , size of alphabet; PD, size of partial derivative NFA; DT size of DFA from PD; MIN, size of minimal DFA; FIFA, size of FIFA from PD.

$m, k$	Type	min	avg	max	$m, k$	Type	min	avg	max	$m, k$	Type	min	avg	max
50, 2	FIFA	3	10.1684	25	50, 3	FIFA	3	12.4948	25	50, 10	FIFA	6	14.1252	24
	DT	3	10.1338	62		DT	3	13.9339	56		DT	7	15.3764	33
	MIN	1	5.0762	51		MIN	1	9.1225	41		MIN	1	13.8138	30
	PD	3	10.6904	19		PD	4	11.6522	19		PD	6	13.3556	21
100, 2	FIFA	3	19.2113	46	100, 3	FIFA	5	24.3173	45	100, 10	FIFA	14	27.4189	43
	DT	3	19.7193	158		DT	5	34.289	166		DT	15	32.5608	66
	MIN	1	6.2814	116		MIN	1	18.2094	148		MIN	1	28.7841	58
	PD	9	20.0239	30		PD	11	21.8518	32		PD	13	25.2294	36
250, 2	FIFA	9	48.3731	107	250, 3	FIFA	23	60.7792	110	250, 10	FIFA	44	67.8454	99
	DT	12	185.6424	1120		DT	12	185.6424	1586		DT	55	102.6683	329
	MIN	1	7.0256	630		MIN	1	59.0185	988		MIN	1	88.8932	253
	PD	34	47.998	66		PD	34	52.5888	70		PD	41	60.8428	78
500, 2	FIFA	35	99.8889	189	500, 3	FIFA	42	122.2247	198	500, 10	FIFA	102	135.1439	170
	DT	13	186.1518	6451		DT	37	1143.2134	11687		DT	128	277.5053	1122
	MIN	1	6.8369	745		MIN	1	92.8985	2343		MIN	1	237.4012	869
	PD	72	94.6422	124		PD	79	103.6871	126		PD	94	120.0465	150

## 7 Conclusions

We presented a class of initially-connected NFAs with a single initial state for which it is possible to test isomorphism in polynomial time. The definition of these automata (FIFA) is based on the possibility to order the set of states in a breath-first search traversal from the initial state. The uniform random generator can efficiently sample datasets to test the performance of algorithms dealing with NFAs. Moreover one can extend it with a parameter for the density of transitions in order to avoid the high frequency of NFAs recognising the universal language. The class of FIFAs can be further studied in its own. One can obtain asymptotic bounds for the number of FIFAs of given  $n$  and  $k$ . Upper bounds for the size of the minimal FIFAs for a given language will be of major interest. Operational state complexity with respect to this class of automata is also an open problem.

## References

1. Almeida, M., Moreira, N., Reis, R.: Enumeration and generation with a string automata representation. *Theor. Comput. Sci.* **387**(2), 93–102 (2007)
2. Antimirov, V.M.: Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.* **155**(2), 291–319 (1996)
3. Bassino, F., Nicaud, C.: Enumeration and random generation of accessible automata. *Theor. Comput. Sci.* **381**(1–3), 86–104 (2007)
4. Champarnaud, J.M., Paranthoën, T.: Random generation of DFAs. *Theor. Comput. Sci.* **330**(2), 221–235 (2005)

5. FAdo, P.: FAdo: tools for formal languages manipulation. <http://fado.dcc.up.pt>. Accessed 13 2018
6. Héam, P.-C., Joly, J.-L.: On the uniform random generation of non deterministic automata up to isomorphism. In: Drewes, F. (ed.) CIAA 2015. LNCS, vol. 9223, pp. 140–152. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-22360-5\\_12](https://doi.org/10.1007/978-3-319-22360-5_12)
7. Knuth, D.E.: The Art of Computer Programming. Seminumerical Algorithms, vol. 2, 2nd edn. Addison Wesley, Boston (1981)
8. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov Chain and Mixing Times. American Mathematical Society (2008). <http://pages.uoregon.edu/dlevin/MARKOV/markovmixing.pdf>
9. Lothaire, M.: Applied Combinatorics on Words, vol. 105. Encyclopedia of Mathematics and its Applications. Cambridge University Press, New York (2005)
10. Nicaud, C.: Random deterministic automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 5–23. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-44522-8\\_2](https://doi.org/10.1007/978-3-662-44522-8_2)



# On the Generation of 2-Polyominoes

Enrico Formenti<sup>1</sup>(✉) and Paolo Massazza<sup>2</sup>

<sup>1</sup> Université Côte d'Azur (UCA), CNRS, I3S, Nice, France

`enrico.formenti@unice.fr`

<sup>2</sup> Department of Theoretical and Applied Sciences - Computer Science Section,  
Università degli Studi dell'Insubria, Via Mazzini 5, 21100 Varese, Italy

`paolo.massazza@uninsubria.it`

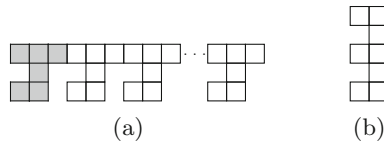
**Abstract.** The class of 2-polyominoes contains all polyominoes  $P$  such that for any integer  $i$ , the first  $i$  columns of  $P$  consist of at most 2 polyominoes. We provide a decomposition that allows us to exploit suitable discrete dynamical systems to define an algorithm for generating all 2-polyominoes of area  $n$  in constant amortized time and space  $O(n)$ .

## 1 Introduction

A *polyomino* is a finite and connected union of unitary squares (*cells*) in the plane  $\mathbb{Z} \times \mathbb{Z}$ , considered up to translations [8]. The number of cells of a polyomino is its *area*. The problem of counting the number of polyominoes of area  $n$  (*i.e.* determining a closed formula for  $|\text{Pol}(n)|$ ) is still open and the most used algorithm to generate  $\text{Pol}(n)$  runs in exponential time and uses exponential space [9]. Due to the space requirement, such algorithm has been used to compute  $|\text{Pol}(n)|$  only for  $n \leq 56$  [10]. So, it is interesting to study suitable representations of polyominoes that allow to design efficient generating algorithms with strict constraints on the space. To classify the polyominoes and to tackle some difficult questions about them, several subclasses have been introduced in literature. For instance, the class of convex polyominoes and its subclasses have been studied under different points of view [1, 4–6]. Recently, some efficient algorithms for the exhaustive generation by area of some subclasses of convex polyominoes have been presented [2, 3, 12, 14].

The problem of efficiently generating non-convex polyominoes is particularly difficult. Indeed, no Constant Amortized Time (CAT) algorithm using space  $O(n)$  is known for the generation of  $\text{Pol}(n)$ . In order to answer this question, we tackled some special cases in the hope of detecting all the conceptual obstacles and trying to remove them. Indeed, we are interested in the class  $\text{Pol}_k(n)$  containing all polyominoes  $P$  of area  $n$  such that, for any  $i > 0$ , the first  $i$  columns of  $P$  are made of at most  $k$  polyominoes. A CAT algorithm for the exhaustive generation of  $\text{Pol}_1(n)$  (called partially directed animals in [15]), has been proposed in [7]. This paper provides a CAT algorithm that generates  $\text{Pol}_2(n)$  using space  $O(n)$ . Remark that  $\text{Pol}_2(n)$  is a strictly new class which neither coincides

with the class of partially directed animals with  $k$  sources (see Fig. 1) nor with other known subclasses of polyominoes.



**Fig. 1.** (a) a 2-polyomino corresponding to a partially directed animal with  $k + 1$  sources ( $k$  is the number of repetitions of the gray pattern). (b) a partially directed animal with 3 sources which is not a 2-polyomino.

The algorithm is based on an idea that was already used for other exhaustive generation algorithms (for instance, [11–13]) and it is inspired by discrete dynamical systems theory. Indeed, it proceeds to generate a polyomino column by column, and each column is uniquely decomposed into at most four parts, namely the skeleton and three regions. Each region corresponds to the state of a suitable dynamical system. Proving that the dynamical system is surjective and that it has one orbit grants exhaustive generation. Granting that the next state can be computed from the previous one in constant amortized time and in linear space leads to the result. This last step is accomplished through a careful choice of the data structure used to represent a polyomino.

Because of the lack of space, most of the proofs and the pseudo-code have been omitted. They will appear in the long version of the paper.

## 2 Preliminaries

The *area*  $A(P)$  of a polyomino  $P$  is the number of its cells. A polyomino can be seen as a (finite) sequence of columns. A *column* consists of a sequence of *vertical segments* separated by empty unit squares. A *vertical segment* is a sequence of unit cells  $q_1, q_2, \dots, q_k$  that are in the same column and such that  $q_i$  is edge-adjacent to  $q_{i+1}$ , for  $1 \leq i < k$ . The *position* of a cell is its y-coordinate. The position of the top (resp., bottom) cell of a segment  $s$  is denoted by  $\text{Top}(s)$  (resp.,  $\text{Bot}(s)$ ). We represent a segment  $s$  of a column by means of the pair  $(A(s), \text{Top}(s))$ . Segments belonging to the same column are numbered from the top to the bottom, thus a column with  $p$  segments is simply a sequence of disjoint segments  $\mathbf{c} = (s_1, \dots, s_p)$ , with  $\text{Top}(s_i) < \text{Top}(s_{i-1}) - A(s_{i-1}) - 1$  for  $1 < i \leq p$ . The *area* of  $\mathbf{c}$  is  $A(\mathbf{c}) = \sum_{i=1}^p A(s_i)$ . Furthermore, the position of  $\mathbf{c}$  is the position of its first segment,  $\text{Top}(\mathbf{c}) = \text{Top}(s_1)$ . Given a segment  $s$  and an integer  $j$  such that  $\text{Top}(s) > j \geq \text{Bot}(s)$ , we denote by  $s_{>j}$  (resp.,  $s_{\leq j}$ ) the subsegment consisting of the cells of  $s$  with position greater than  $j$  (resp., smaller than or equal to  $j$ ). The part of a column  $\mathbf{c}$  that is above a position  $j$  is  $\mathbf{c}_{>j}$  ( $\mathbf{c}_{\geq j}$ ,  $\mathbf{c}_{\leq j}$  and  $\mathbf{c}_{<j}$  are defined similarly). Segments can be ordered with respect to their position and their area.

**Definition 1 (< on segments).** Let  $u$  and  $v$  be two segments. Then, one has  $u < v$  if and only if  $Top(u) > Top(v)$  or  $Top(u) = Top(v)$  and  $A(u) > A(v)$ .

A total order on columns, denoted by  $\prec$ , can be easily obtained by extending  $<$ .

**Definition 2 (< on columns).** Let  $\mathbf{b} = (s_1, \dots, s_p)$  and  $\mathbf{c} = (t_1, \dots, t_q)$  be two columns. Then, one has  $\mathbf{b} \prec \mathbf{c}$  if and only if either  $A(\mathbf{b}) > A(\mathbf{c})$ , or  $A(\mathbf{b}) = A(\mathbf{c})$  and there exists  $m$  with  $1 \leq m \leq \min(p, q)$  such that  $s_j = t_j$  for all  $j < m$  and  $s_m < t_m$ .

Consider a polyomino  $P = (\mathbf{c}_1, \dots, \mathbf{c}_q)$ . We assume that the position of the bottom cell of the last segment of  $\mathbf{c}_1$  is 0. We indicate by  $P_{\leq i}$  (resp.,  $P_i$ ) the sequence of the first  $i$  columns of  $P$  from the left (resp., the  $i$ -th column of  $P$ ), also called the  $i$ -prefix of  $P$ . We also write  $P_{< i}$  for  $P_{\leq i-1}$ . The area of  $P$  is  $A(P) = \sum_{i=1}^w A(P_i)$  where  $w$  is the number of columns of  $P$ . From here on,  $\text{Pol}(n)$  denotes the set of the polyominoes of area  $n$ .

**Definition 3 (The class  $\text{Pol}_k(n)$ ).** Let  $k > 0$ . The class of  $k$ -polyominoes of area  $n$ ,  $\text{Pol}_k(n)$ , contains all  $P \in \text{Pol}(n)$  such that, for all  $i$  with  $1 \leq i < w$ ,  $P_{\leq i}$  consists of at most  $k$  polyominoes, where  $w$  is the number of columns of  $P$ .

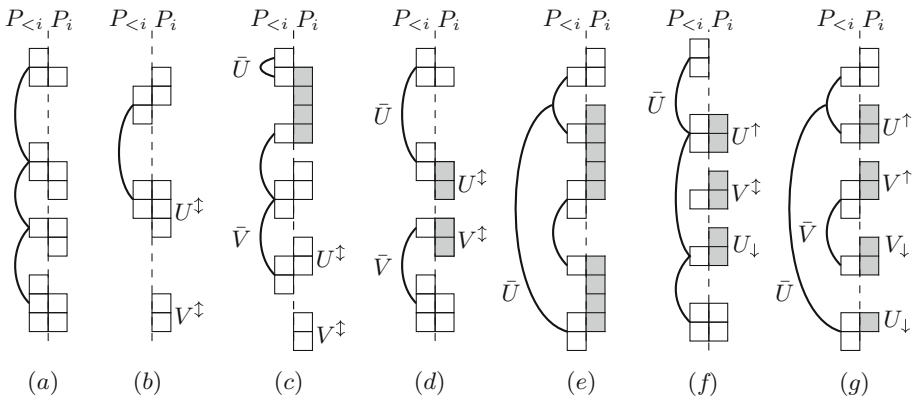
The class  $\text{Pol}_1(n)$  corresponds to the class of partially directed animals [15–17], for which a CAT generation algorithm has been provided in [7]. Here, we are interested into an efficient exhaustive generation algorithm for the class  $\text{Pol}_2(n)$ .

Two segments  $s$  and  $t$  are *adjacent* if they belong to adjacent columns and there exists at least one cell of  $s$  that is edge-adjacent to a cell of  $t$ . The set  $\text{LAdj}(s)$  of *left-adjacencies* of a segment  $s$  contains the positions of the cells of  $s$  that are edge-adjacent to some cells of segments in  $P_{i-1}$ . A segment  $s$  is *l-detached* if  $\text{LAdj}(s) = \emptyset$ , whereas it is *l-adjacent* if  $\text{LAdj}(s) \neq \emptyset$ . A column that does not contain l-detached segments is called *l-detached-free*. The segment of  $P_i$  immediately above (resp., below)  $s$  is  $s^\uparrow$  (resp.,  $s_\downarrow$ ). If  $s$  is l-detached then  $\downarrow s$  (resp.,  $\uparrow s$ ) indicates the first (resp., last) segment  $v$  in  $P_{i-1}$  such that  $Top(v) < Bot(s)$  (resp.,  $Bot(v) > Top(s)$ ). Otherwise, if  $s$  is l-adjacent then  $\uparrow s$  (resp.,  $\downarrow s$ ) indicates the segment in  $P_{i-1}$  that has a cell in the position  $\min(\text{LAdj}(s))$  (resp.,  $\max(\text{LAdj}(s))$ ). For any segment  $s$  of  $P_i$ ,  $\text{Con}(s)$  is the largest polyomino comprised in  $P_{\leq i}$  that contains  $s$ . Let  $P_{\leq i}$  consist of two polyominoes  $U$  and  $V$ . Then,  $U^\uparrow$  (resp.,  $V^\uparrow$ ) indicates the *upper extreme* of  $U$  (resp.,  $V$ ) in  $P_i$ , that is, the first segment  $u \in U$  (resp.,  $v \in V$ ) of  $P_i$  such that  $u^\uparrow \in V$  or  $u_\downarrow \in V$  (resp.,  $v^\uparrow \in U$  or  $v_\downarrow \in U$ ). Analogously,  $U_\downarrow$  (resp.,  $V_\downarrow$ ) denotes the *lower extreme* of  $U$  (resp.,  $V$ ) in  $P_i$  that is, the last segment  $u \in U$  (resp.,  $v \in V$ ) of  $P_i$  such that  $u^\uparrow \in V$  or  $u_\downarrow \in V$  (resp.,  $v^\uparrow \in U$  or  $v_\downarrow \in U$ ). If  $U_\downarrow = U^\uparrow$  then  $U$  has only one extreme in  $P_i$ , denoted by  $U^\downarrow$ . We say that  $U$  and  $V$  are *independent* if both  $U$  and  $V$  have only one extreme in  $P_i$  ( $U^\downarrow$  is above  $V^\downarrow$  or vice versa), see Cases (b), (c) and (d) of Fig. 2. Similarly,  $U$  *includes*  $V$  if  $V^\uparrow$  is immediately below  $U^\uparrow$  and  $V_\downarrow$  is immediately above  $U_\downarrow$ , see Cases (f) and (g) of Fig. 2. Thus,  $U$  and  $V$  are *enclosing* if  $U$  includes  $V$  or vice versa.

A segment  $s$  of  $P_i$  is a *bridge* if  $P_{<i}$  consists of two polyominoes  $\bar{U}$  and  $\bar{V}$  and  $s$  is l-adjacent to two segments  $v$  and  $v_\downarrow$ , which are extremes of  $\bar{U}$  and  $\bar{V}$ , respectively. The *distance*  $\text{Dist}(s)$  of an extreme  $s$  is the number of empty unit squares that separate  $s$  from the nearest segment  $t \in \{s_\downarrow, s^\uparrow\}$  with  $\text{Con}(t) \neq \text{Con}(s)$ . In other words,  $\text{Dist}(s) + 2$  is the area of the smallest bridge that can join  $s$  and  $t$ . The *vertical distance* of two segments  $s, t$  with  $\text{Bot}(s) \geq \text{Top}(t)$  is  $d(s, t) = \text{Bot}(s) - \text{Top}(t) - 1$ . If  $P_{<i}$  consists of two distinct polyominoes  $U$  and  $V$ , we also set  $\text{Dist}(U, V) = \min\{\text{Dist}(s) \mid s \text{ is an extreme of } U\}$ .

**Notation:** From here on,  $P_{\leq i}$  (resp.,  $P_{<i}$ ) denotes an  $i$ -prefix (resp.,  $(i - 1)$ -prefix) of a polyomino in  $\text{Pol}_2(n)$ . A bar over a segment indicates that it is a bridge,  $\bar{s}$ .

Given a sequence  $S$  of  $i - 1$  columns and a column  $\mathbf{c}$ , the operation  $|$  of *column concatenation* (left associative) produces a new sequence  $S' = S|\mathbf{c}$  with  $i$  columns. A column  $\mathbf{c}$  is *compatible* with  $P_{i-1}$ , denoted by  $\mathbf{c} \square P_{i-1}$ , if and only if  $P_{<i}|\mathbf{c}$  is the  $i$ -prefix of a polyomino in  $\text{Pol}_2(n)$ . Obviously, one has  $\mathbf{c} \square P_{i-1}$  if and only if  $P_{<i}|\mathbf{c} \in \text{Pol}_2(n)$  or there exists a column  $\mathbf{d}$  such that  $P_{<i}|\mathbf{c}|\mathbf{d} \in \text{Pol}_2(n)$ . We denote by  $\text{Comp}(P_{<i}, k, n)$  the set of columns of area  $k$  that are compatible with  $P_{i-1}$ . Bridges and l-adjacent extremes are called *special segments*. They determine a unique decomposition of  $\mathbf{b}$  into *regions*, and they form the so-called *skeleton* of  $\mathbf{b}$ . The skeleton of a column  $\mathbf{b} \in \text{Comp}(P_{<i}, k, n)$ , denoted by  $\text{Ske}(\mathbf{b})$ , is either the column consisting of the bridges in  $\mathbf{b}$  (if any), or the column containing all l-adjacent extremes in  $\mathbf{b}$  (if  $\mathbf{b}$  does not contain a bridge and  $P_{<i} \notin \text{Pol}$ ).



**Fig. 2.** Columns, compatibility and skeletons (represented by gray cells).

**Definition 4.** Let  $\mathbf{b} \in \text{Comp}(P_{<i}, k, n)$ . Then, the skeleton of  $\mathbf{b}$  is the column

$$\text{Ske}(\mathbf{b}) = \begin{cases} () & \text{if } P_{<i} \in \text{Pol} \\ (\bar{s}) & \text{if } \mathbf{b} \text{ contains only one bridge } \bar{s} \\ (\bar{s}, \bar{t}) & \text{if } \mathbf{b} \text{ contains two bridges } \bar{s}, \bar{t} \\ (s, s_{\downarrow}) & \text{if } P_{<i} | \mathbf{b} = U \cup V, s = U^{\uparrow}, s_{\downarrow} = V^{\downarrow} \\ (s, s_{\downarrow}, s_{\downarrow\downarrow}) & \text{if } P_{<i} | \mathbf{b} = U \cup V, s = U^{\uparrow}, s_{\downarrow} = V^{\downarrow}, s_{\downarrow\downarrow} = U^{\downarrow} \\ (s, s_{\downarrow}, u, u_{\downarrow}) & \text{if } P_{<i} | \mathbf{b} = U \cup V, s = U^{\uparrow}, s_{\downarrow} = V^{\downarrow}, u = V^{\downarrow}, u_{\downarrow} = U^{\downarrow} \end{cases}$$

where  $U^{\uparrow}, U^{\downarrow}, V^{\uparrow}, V^{\downarrow}, V^{\uparrow}, U^{\downarrow}$  are l-adjacent extremes of  $\mathbf{b}$  (with no bridges).

Notice that if  $\mathbf{b}$  contains a bridge  $t$  then it does not contain an l-adjacent extreme  $s$  with  $\text{Con}(t) \neq \text{Con}(s)$  (otherwise  $P_{<i}$  consists of at least three polyominoes). For any  $\mathbf{b} \in \text{Comp}(P_{<i}, k, n)$  such that  $\text{Ske}(\mathbf{b}) \neq ()$  one has  $\text{Ske}(\mathbf{b}) \in \text{Comp}(P_{<i}, h, n)$  for a suitable integer  $h \leq k$ .

**Definition 5.** The northern (resp., central, southern) region of  $\mathbf{b} = (s_1, \dots, s_q)$  is denoted by  $\mathbf{b}^{\uparrow}$  (resp.,  $\mathbf{b}^{\downarrow}, \mathbf{b}_{\downarrow}$ ) and it is defined as follows ( $\text{Ske}(\mathbf{b})$  to the left):

$$\begin{aligned} () : \mathbf{b}^{\uparrow} &= \mathbf{b}; \\ (\bar{s}_m) : \mathbf{b}^{\uparrow} &= (s_1, \dots, s_{m-1}) \text{ and } \mathbf{b}_{\downarrow} = (s_{m+1}, \dots, s_q); \\ (\bar{s}_m, \bar{s}_p) : \mathbf{b}^{\uparrow} &= (s_1, \dots, s_{m-1}), \mathbf{b}^{\downarrow} = (s_{m+1}, \dots, s_{p-1}), \mathbf{b}_{\downarrow} = (s_{p+1}, \dots, s_q); \\ (s_m, s_{m+1}) : \mathbf{b}^{\uparrow} &= (s_1, \dots, s_{m-1}), \mathbf{b}_{\downarrow} = (s_{m+2}, \dots, s_q); \\ (s_m, s_{m+1}, s_{m+2}) : \mathbf{b}^{\uparrow} &= (s_1, \dots, s_m), \mathbf{b}_{\downarrow} = (s_{m+3}, \dots, s_q); \\ (s_m, s_{m+1}, s_p, s_{p+1}) : \mathbf{b}^{\uparrow} &= (s_1, \dots, s_{m-1}), \mathbf{b}^{\downarrow} = (s_{m+2}, \dots, s_{p-1}), \mathbf{b}_{\downarrow} = (s_{p+2}, \dots, s_q); \end{aligned}$$

Remark that if  $s$  and  $t$  are l-adjacent segments belonging to the same region then  $\text{Con}(s) = \text{Con}(t)$ . The notion of skeleton induces an equivalence relation on  $\text{Comp}(P_{<i}, k, n)$ .

**Definition 6** ( $\diamond$ ). Let  $\mathbf{b}, \mathbf{c} \in \text{Comp}(P_{<i}, k, n)$ . Then, one has  $\mathbf{b} \diamond \mathbf{c}$  if and only if  $\text{Ske}(\mathbf{b}) = \text{Ske}(\mathbf{c})$ .

The set  $\text{Sk}(P_{<i}, k, n) = \{\mathbf{c} \in \text{Comp}(P_{<i}, j, n) | j \leq k \wedge \text{Ske}(\mathbf{c}) = \mathbf{c}\}$  contains the skeletons of area at most  $k$  that are compatible with  $P_{i-1}$ . Obviously, one has

$$\text{Comp}(P_{<i}, k, n) = \bigcup_{\mathbf{b} \in \text{Sk}(P_{<i}, j, n)} [\mathbf{b}]_{\diamond}.$$

Notice that  $\text{Sk}(P_{<i}, j, n) = \emptyset$  if  $\text{Sk}(P_{<i}, j+1, n) \setminus \text{Sk}(P_{<i}, j, n) = \emptyset$ . We define a total order  $\triangleleft$  on  $\text{Sk}(P_{<i}, k, n)$  as follows.

**Definition 7** ( $\triangleleft$  on  $\text{Sk}(P_{<i}, k, n)$ ). Let  $\mathbf{b}, \mathbf{c} \in \text{Sk}(P_{<i}, k, n)$ . Then, one has  $\mathbf{b} \triangleleft \mathbf{c}$  if and only if one of the following conditions holds:

- $|\mathbf{b}| < |\mathbf{c}|$ , (i.e.  $\mathbf{b}$  contains less segments than  $\mathbf{c}$ );
- $\mathbf{b}$  contains a bridge whereas  $\mathbf{c}$  does not;

- $\mathbf{b} = (\bar{s}), \mathbf{c} = (\bar{t})$  and  $\bar{s} < \bar{t}$ ;
- $\mathbf{b} = (s, t)$  (resp.,  $\mathbf{b} = (\bar{s}, \bar{t})$ ),  $\mathbf{c} = (v, w)$  (resp.,  $\mathbf{c} = (\bar{v}, \bar{w})$ ) and either  $s < v$  (resp.,  $\bar{s} < \bar{v}$ ), or  $s = v$  (resp.,  $\bar{s} = \bar{v}$ ) and  $t < w$  (resp.,  $\bar{t} < \bar{w}$ );
- $\mathbf{b} = (s, t, u)$ ,  $\mathbf{c} = (v, w, z)$  and either  $s < v$ , or  $s = v$  and  $t < w$ , or  $s = v$ ,  $t = w$  and  $u < z$ ;
- $\mathbf{b} = (q, r, s, t)$ ,  $\mathbf{c} = (u, v, w, z)$  and either  $q < u$ , or  $q = u$  and  $r < v$ , or  $q = u$ ,  $r = v$  and  $s < w$ , or  $q = u$ ,  $r = v$ ,  $s = w$  and  $t < z$ ;

In Sect. 4, polyominoes of  $\text{Pol}_2(n)$  are generated column by column. All the columns that are compatible with  $P_{i-1}$  are generated according to the following ordering.

**Definition 8 (< on columns).** Let  $\mathbf{b}, \mathbf{c} \in \text{Comp}(P_{< i}, k, n)$ . Then, one has  $\mathbf{b} < \mathbf{c}$  iff  $\text{Ske}(\mathbf{b}) \triangleleft \text{Ske}(\mathbf{c})$  or  $\text{Ske}(\mathbf{b}) = \text{Ske}(\mathbf{c})$  and one of the following holds:

- (**Ske**( $\mathbf{b}$ ) is null)  $\mathbf{b} \prec \mathbf{c}$ ;
- (**Ske**( $\mathbf{b}$ ) is  $(\bar{s})$  or  $(s, t)$  or  $(s, t, u)$ ) either  $\mathbf{b}^\uparrow \prec \mathbf{c}^\uparrow$ , or  $\mathbf{b}^\uparrow = \mathbf{c}^\uparrow \wedge \mathbf{b}_\downarrow \prec \mathbf{c}_\downarrow$ ;
- (**Ske**( $\mathbf{b}$ ) is  $(\bar{s}, \bar{t})$  or  $(s, t, u, v)$ ) either  $\mathbf{b}^\uparrow \prec \mathbf{c}^\uparrow$ , or  $\mathbf{b}^\uparrow = \mathbf{c}^\uparrow \wedge \mathbf{b}^\downarrow \prec \mathbf{c}^\downarrow$ , or  $\mathbf{b}^\uparrow = \mathbf{c}^\uparrow \wedge \mathbf{b}^\downarrow = \mathbf{c}^\downarrow \wedge \mathbf{b}_\downarrow \prec \mathbf{c}_\downarrow$ ;

### 3 Dynamical Systems for Regions

Given a column  $\mathbf{b}$ , its skeleton  $\text{Ske}(\mathbf{b})$  uniquely identifies the regions  $\mathbf{b}^\uparrow$ ,  $\mathbf{b}_\downarrow$  and  $\mathbf{b}^\downarrow$ . Recall that a region of  $\mathbf{b}$  is a particular subsequence of  $\mathbf{b}$  consisting of consecutive segments delimited by special segments in  $\text{Ske}(\mathbf{b})$ . In each region, we are going to use specific dynamical systems such that their current state represents a list of segments that when completed with the skeleton produces a compatible column. All compatible columns can be built in this way. Given an  $i$ -prefix  $P_{< i}|\mathbf{b}$ , with  $h = A(\mathbf{b})$  ( $1 \leq h \leq n - A(P_{< i})$ ), and a segment  $s$  of a region  $\mathbf{R}$  of  $\mathbf{b}$ , a *move* at  $j \in [\text{Bot}(s), \text{Top}(s)]$  is an operation which rearranges the cells of  $\mathbf{R}$  while respecting a constraint (expressed by a value  $\delta \geq 0$ ) regarding the creation of l-detached segments. More precisely, if  $\delta = 0$  then no new l-detached segment can be created (because an l-detached segment is already present in  $\mathbf{b}$ , or the number of remaining cells is not sufficient to create a bridge in the next column, that is,  $n - A(P_{< i}|\mathbf{b}) < 3$ , or  $\text{Ske}(\mathbf{b})$  contains at least two l-adjacent extremes), whereas  $\delta > 0$  allows the creation of a new l-detached segment with distance at most  $\delta$  only if  $\mathbf{b}$  is l-detached free (moreover,  $\text{Ske}(\mathbf{b})$  should not contain l-adjacent extremes and  $\delta + 2 = n - A(P_{< i}|\mathbf{b})$ ). Two types of moves are devised, namely *shift* and *split*, and both output a region  $\mathbf{R}'$  such that  $\mathbf{R} \prec \mathbf{R}'$ . Notice that, since  $\text{Ske}(\mathbf{b}) \in \text{Comp}(P_{< i}, j, n)$  (with  $j \leq h$ ), any rearrangement  $\mathbf{R}'$  of the cells of  $\mathbf{R}$  such that

1.  $\mathbf{R}'$  is l-detached free if  $\delta = 0$ ,
2. no cell of  $\mathbf{R}'$  becomes edge-adjacent to a segment of  $\text{Ske}(\mathbf{b})$ ,
3. if  $\delta > 0$  then  $\mathbf{R}'$  contains at most one l-detached segment  $s'$  and  $\text{Dist}(s') \leq \delta$



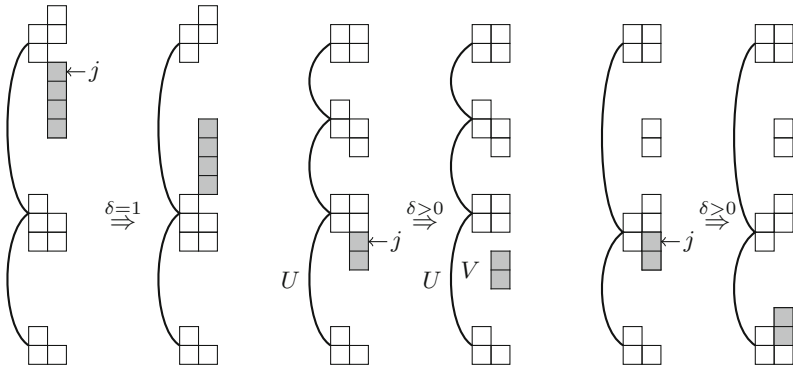
produces a column  $\mathbf{b}' \in \text{Comp}(P_{<i}, h, n)$ . A shift move may occur only in the position  $j = \text{Top}(s)$  of a segment  $s$ . It has the effect of shifting the segment  $s$  by  $e$  positions downwards, where  $e$  is the smallest integer such that the resulting region satisfies Conditions 1–3 above, see Fig. 2(left). Possibly,  $s$  and  $s_\downarrow$  are joined (if  $s_\downarrow \notin \text{Ske}(\mathbf{b})$ ). If such an  $e$  does not exist, then the move is undefined.

A split move may occur only in a position  $j \in [\text{Bot}(s), \text{Top}(s)]$  of a segment  $s$ . In this case,  $s$  is split into  $s_{>j}$  and  $s_{\leq j}$ . The segment  $s_{>j}$  stays in place whereas  $s_{\leq j}$  is shifted  $e$  positions downwards, where  $e$  is the smallest integer such that the resulting region  $\mathbf{R}'$  satisfies Conditions 1–3 above, see Fig. 2(center). Possibly,  $s_{\leq j}$  and  $s_\downarrow$  are joined (if  $s_\downarrow \notin \text{Ske}(\mathbf{b})$ ), see Fig. 2(right).

Denote by  $\text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta)$  the set of the positions where a move occurs on a segment of  $\mathbf{R}$ . The following lemma relates the non-emptiness of  $\text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta)$  to the number of segments in  $\mathbf{R}$ .

**Lemma 1.** *Let  $\mathbf{R}$  be a region of  $\mathbf{b}$ . If  $\mathbf{R}$  contains at least three segments, then  $\text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta) \neq \emptyset$ .*

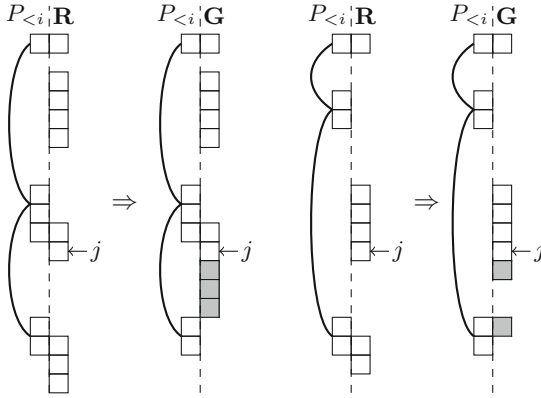
*Proof (outline).* Let  $s, s_\downarrow$  and  $s_{\downarrow\downarrow}$  be the last three segments of  $\mathbf{R}$ . If  $\mathbf{R}$  does not contain an l-detached segment then a shift move on  $s_\downarrow$  is always defined. Otherwise, if  $s$  (resp.,  $s_\downarrow$ ) is l-detached then a shift move on  $s$  (resp.,  $s_\downarrow$ ) is defined. Lastly, if  $s_{\downarrow\downarrow}$  is l-detached then  $s$  can always be shifted.  $\square$



**Fig. 3.** Examples of shift (left) and split moves (center and right).

Remark that in general several moves are possible in  $\mathbf{R}$ . However, we are interested in the one occurring in the lowest position. This leads to the following:

**Definition 9 (Grand ancestor).** *Let  $\mathbf{R}$  be a region of  $\mathbf{b}$  and suppose that  $j = \min(\text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta))$  is defined. The grand ancestor  $\text{GA}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta)$  of  $\mathbf{R}$  is a region  $\mathbf{G}$  delimited as  $\mathbf{R}$  and such that  $A(\mathbf{G}) = A(\mathbf{R})$ ,  $\mathbf{G}$  is identical to  $\mathbf{R}$  in the positions above  $j - 1$ ,  $\mathbf{G}$  admits a move at  $j$  and for all other regions  $\mathbf{R}'$  satisfying the previous properties, it holds  $\mathbf{G} \prec \mathbf{R}'$  (possibly,  $\mathbf{G} = \mathbf{R}'$ ).*



**Fig. 4.** Examples of grand ancestor (gray cells are those redeployed,  $\delta = 4$ ).

The lowest position where a move may occur in a segment  $s$  is characterized in the following Lemma.

**Lemma 2.** *Let  $s$  be a segment of a region  $\mathbf{R}$  of  $\mathbf{b}$  and let  $j$  be the lowest position of  $s$  such that  $j \in \text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta)$ . Then,*

$$j = \begin{cases} \text{Bot}(s) \text{ or } 1 + \min(\text{LAdj}(s)) \text{ or } \text{Top}(s) & \text{if } \delta = 0, \\ \text{Bot}(s) \text{ or } 1 + \text{Bot}(s) & \text{if } \delta > 0 \text{ and } s \text{ is } l\text{-adjacent}, \\ \text{Bot}(s) \text{ or } \text{Top}(s) & \text{if } \delta > 0 \text{ and } s \text{ is } l\text{-detached}, \end{cases}$$

We point out that the data structure described in Sect. 4.1 allows (in time  $O(1)$ ) both to find the lowest position  $j$  where a move may occur in a segment  $s$ , and to make the move at  $j$ . Furthermore, the grand ancestor can be effectively computed by exploiting Lemmas 1 and 2. Indeed, one can easily prove the following:

**Lemma 3.** *Given a region  $\mathbf{R} = (s_1, \dots, s_m)$  of  $\mathbf{b}$ , let  $s_h$  be the segment with a cell in position  $j = \min(\text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta))$ . Then,  $\text{GA}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta) = (s_1, \dots, s_{h-1}) \cdot (s', s'_\downarrow)$  where  $\text{Top}(s') = \text{Top}(s_h)$  and*

$$\begin{cases} A(s') = A(s_h) + w - 1, A(s'_\downarrow) = 1, \text{Top}(s'_\downarrow) = \text{Top}(\downarrow s_h) & \text{if } G, \\ A(s') = A(s_h) + w, s'_\downarrow = \epsilon & \text{otherwise,} \end{cases}$$

where  $w = \sum_{k=h+1}^m A(s_k)$  and  $G$  holds if and only if  $s_h$  is  $l$ -detached,  $d(s_{h-1}, s_h) > \delta$  and  $d(s_h, v) - w > \delta$ , where  $v$  is the segment below  $s_m$  (if any).

Given two regions  $\mathbf{R}$  and  $\mathbf{R}'$ , write  $\mathbf{R} \Rightarrow \mathbf{R}'$  if and only if  $\mathbf{R}'$  is obtained by making a move in  $\mathbf{R}$ . Moreover,  $\mathbf{R} \xrightarrow{j} \mathbf{R}'$  if  $j \in \text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta)$  and the move at the position  $j$  of  $\mathbf{R}$  produces the region  $\mathbf{R}'$ .

At this point we are ready to define a discrete dynamical system  $f_{P_{<i}|\mathbf{b}, \delta}$  that takes in input a region  $\mathbf{R}$  of  $\mathbf{b}$  and changes its content, producing a region  $\mathbf{R}'$  such that  $\mathbf{R} < \mathbf{R}'$  unless the input region contains no moves.

**Definition 10 (The dynamical system over regions).** *Given a region  $\mathbf{R}$  of  $\mathbf{b}$ , let  $j = \min(\text{Mcr}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta))$  and define*

$$f_{P_{<i}|\mathbf{b}, \delta}(\mathbf{R}) = \begin{cases} \mathbf{R}' & \text{if } j \neq \perp \text{ and } \text{GA}(P_{<i}|\mathbf{b}, \mathbf{R}, \delta) \xrightarrow{j} \mathbf{R}' \\ \mathbf{R} & \text{otherwise.} \end{cases}$$

Let  $\mathbf{R} = (s_1, \dots, s_m)$  be a region of a column  $\mathbf{b}$ . Then,  $RC(\text{Ske}(\mathbf{b}), \mathbf{R}, \delta)$  denotes the set of all sequences  $\mathbf{G}$  of segments, with  $A(\mathbf{G}) = A(\mathbf{R})$ , such that one can replace  $\mathbf{R}$  with  $\mathbf{G}$  obtaining a different column  $\mathbf{b}'$  with  $\mathbf{b}' \sqsubset P_{i-1}$ . Remark that  $RC(\text{Ske}(\mathbf{b}), \mathbf{R}, \delta)$  is finite and totally ordered by  $\prec$ . We call *initial region* its minimum element, denoted  $\mathbf{R}_{\min}$ . Indeed,  $\mathbf{R}_{\min}$  is delimited by  $u$  and  $v$ , where  $u = s_1^\uparrow$  and  $v^\uparrow = s_m$  (they both can be null), and it contains at most two segments  $s'$  and  $s'_\downarrow$  with

$$\begin{aligned} \text{Top}(s') &= \text{Bot}(u) - 2 \text{ if } u \neq \epsilon \wedge (\text{Bot}(\downarrow u) < \text{Bot}(u) - 1 \vee w > d(u, (\downarrow u)_\downarrow) \vee \delta > 0) \\ \text{Bot}(s') &= \text{Top}((\downarrow u)_\downarrow) \text{ if } u \neq \epsilon \wedge (w \leq d(u, (\downarrow u)_\downarrow) \wedge \text{Bot}(\downarrow u) \geq \text{Bot}(u) - 1 \wedge \delta = 0) \\ \text{Bot}(s') &= f \quad \text{if } u = \epsilon \wedge \delta = 0 \\ \text{Bot}(s') &= f + \delta + 1 \text{ if } u = \epsilon \wedge \delta > 0 \end{aligned} \quad (1)$$

where  $w = \sum_{k=1}^m A(s_k)$  and  $f$  is the position of the first segment of  $P_{i-1}$ . Moreover,  $A(s') = w$  and  $s'_\downarrow = \epsilon$  unless  $\delta > 0$  and  $u = \epsilon$ , in which case one has  $A(s') = w - 1$ ,  $A(s'_\downarrow) = 1$  and  $\text{Top}(s'_\downarrow) = \text{Top}(z)$ . The following lemma proves that  $f_{P_{<i}|\mathbf{b}, \delta}$  starts in an initial region and eventually ends.

**Lemma 4.** *Consider an  $i$ -prefix  $P_{<i}|\mathbf{b}$  of a polyomino in  $\text{Pol}_2(n)$  and a region  $\mathbf{R}$  of  $\mathbf{b}$ ,  $\mathbf{b} = \mathbf{b}' \cdot \mathbf{R} \cdot \mathbf{b}''$ . Then, one has*

1. *for all  $\mathbf{a} \in RC(\text{Ske}(\mathbf{b}), \mathbf{R}, \delta)$  such that  $\text{Mcr}(P_{<i}|\mathbf{b}' \cdot \mathbf{a} \cdot \mathbf{b}'', \mathbf{a}, \delta) \neq \emptyset$  it holds  $\mathbf{a} \prec f_{P_{<i}|\mathbf{b}, \delta}(\mathbf{a})$ ;*
2.  *$RC(\text{Ske}(\mathbf{b}), \mathbf{R}, \delta) = \bigcup_{n \in \mathbb{N}} f_{P_{<i}|\mathbf{b}, \delta}^n(\mathbf{R}_{\min})$ , where  $f_{P_{<i}|\mathbf{b}, \delta}^n(\mathbf{R}_{\min})$  corresponds to  $f_{P_{<i}|\mathbf{b}, \delta}(f_{P_{<i}|\mathbf{b}, \delta}^{n-1}(\mathbf{R}_{\min}))$ , with  $f_{P_{<i}|\mathbf{b}, \delta}^0(\mathbf{R}_{\min}) = \mathbf{R}_{\min}$ .*

In other words, the previous lemma grants that if one wants to exhaustively generate all possible regions, it is enough to effectively simulate the dynamical system  $f_{P_{<i}|\mathbf{b}, \delta}$ . There will be neither missing regions nor repetitions.

## 4 Exhaustive Generation of $\text{Pol}_2(n)$

An inductive approach is used. Given  $i, k \in \mathbb{N}$  and a skeleton  $\mathbf{b} \in \text{Sk}(P_{<i}, k, n)$  (where  $P_{<i}$  is the current prefix), suppose that all the polyominoes  $Q \in \text{Pol}_2(n)$  with  $Q_{\leq i} = P_{<i}|\mathbf{c}$  have been already generated, for all columns  $\mathbf{c}$  such that  $A(\mathbf{c}) = k$  and  $\text{Ske}(\mathbf{c}) = \mathbf{b}'$ , where  $\mathbf{b}' \in \text{Sk}(P_{<i}, k, n)$  and  $\mathbf{b}' \triangleleft \mathbf{b}$ . We proceed by generating all the columns  $\mathbf{c} \in \text{Comp}(P_{<i}, k, n) \cap [\mathbf{b}]_\diamond$  and then, for each  $\mathbf{c}$ , we (recursively) generate all the polyominoes  $Q \in \text{Pol}_2(n)$  with  $Q_{\leq i} = P_{<i}|\mathbf{c}$ .

Thus, the problem is reduced to design an efficient algorithm for generating the set  $\text{Comp}(P_{<i}, k, n)$ . To this aim, we propose an algorithm that works in two steps:

1. an outer loop that generates all skeletons  $\mathbf{b}$  in  $\text{Sk}(P_{<i}, k, n)$ ;
2. an inner loop that, for each  $\mathbf{b}$ , generates all colums  $\mathbf{c} \in \text{Comp}(P_{<i}, k, n) \cap [\mathbf{b}]_{\delta}$ .

The algorithm is a smart combination of two main iterators (functions that run through ordered sequences), one for skeletons and one for columns. The pseudocode of all iterators and functions will appear in the long version of this paper. Here we briefly describe how they works.

The iterator  $\text{NEXTSKEL}(P_{<i}, k, \mathbf{b}, \delta)$  takes  $P_{<i}$ , an integer  $k$ , a skeleton  $\mathbf{b} \in \text{Sk}(P_{<i}, k, \delta)$  and  $\delta$ , and returns the smallest skeleton larger than  $\mathbf{b}$  in  $\text{Sk}(P_{<i}, k, \delta)$ . The function is based on Definition 7 and exploits different iterators for different types of skeletons. More precisely, the first skeleton  $\mathbf{b}_{\text{init}}$  is either null (if  $P_{<i} \in \text{Pol}$ ), or it consists of a suitable bridge  $\mathbf{b}_{\text{init}} = (\bar{s})$  with  $A(\bar{s}) = k$  (if  $P_{<i}$  consists of two polyominoes  $U$  and  $V$  with  $\text{Dist}(U, V) + 2 < k$ ), or it consists of two suitable extremes,  $\mathbf{b}_{\text{init}} = (s, t)$  with  $A(s) = k - 1$  and  $A(t) = 1$  (if  $\text{Dist}(U, V) + 2 > k$ ). If  $\mathbf{b}_{\text{init}} = ()$  then, for any  $k \geq 0$ , one has  $\text{Sk}(P_{<i}, k, \delta) = \{()\}$  and we are done. Otherwise, the successor  $\mathbf{b}'$  of a non-null skeleton  $\mathbf{b}$  depends on the number of its segments and on their types - either bridges or extremes, see Definition 4. Indeed, if  $\mathbf{b} = (\bar{s})$  then  $\mathbf{b}'$  is obtained by using an iterator  $\text{NEXTBRIDGE}$  that works as follows. If  $\mathbf{b}$  is not the last skeleton consisting of one bridge, then  $\mathbf{b}'$  is obtained either by shifting  $\bar{s}$  (if after a shift by  $e$  positions downwards  $\bar{s}$  remains a bridge -  $e$  is always 1 if  $P_{i-1}$  contains only two extremes), or by replacing  $\bar{s}$  with a smaller bridge  $\bar{t}$ , with  $A(\bar{t}) = A(\bar{s}) - 1$ , placed in the highest possible position (*i.e.*  $\bar{t}$  is 1-adjacent to the first two extremes  $e$  and  $e_1$  in  $P_{i-1}$  with  $d(e, e_1) + 2 \leq A(\bar{t})$ ). Otherwise,  $\mathbf{b}$  is the last element in the ordered sequence of skeletons consisting of one bridge, and  $\mathbf{b}'$  should be the smallest skeleton consisting of two bridges of total area  $k$ . A function  $\text{INITBRIDGES}$  is used for this purpose. Otherwise, one can not place two bridges of total area  $k$  in column  $i$  (because the number of cells  $k$  is not sufficient or  $P_{i-1}$  has only two extremes or it has three extremes  $e_1, e_2$  and  $e_3$  with  $A(e_2) = 2$ ), and then  $\mathbf{b}'$  should be the smallest skeleton consisting of two 1-adjacent extremes of total area  $k$ . Similarly, if  $\mathbf{b} = (s, t)$  then  $\mathbf{b}'$  is obtained either by using an iterator  $\text{NEXTTWOEXTREMES}$  (if  $\mathbf{b}$  is not the greatest skeleton consisting of two extremes), or a function  $\text{INITTHREEEXTREMES}$ .

$\text{NEXTTWOEXTREMES}$  either shifts  $t$ , or it shifts  $s$  and places  $t$  in the highest possible position below  $s$  (if  $t$  can not be shifted), or it constructs the smallest skeleton comprising two extremes  $s'$  and  $t'$  with  $A(s') = A(s) - 1$  and  $A(t') = A(t) + 1$  (if neither  $t$  nor  $s$  can be shifted), or it constructs the smallest skeleton of area  $k - 1$  that consists of two extremes (if  $A(s') = 1$ ). Otherwise,  $\mathbf{b}$  is greatest skeleton consisting of two extremes and  $\mathbf{b}'$  is the smallest skeleton with three 1-adjacent extremes of total area  $k$ . The remaining cases  $\mathbf{b} = (\bar{s}, \bar{t})$ ,  $\mathbf{b} = (s, t, u)$  and  $\mathbf{b} = (s, t, u, v)$  are treated similarly.

In addition to the previous iterators, an iterator  $\text{NEXT}$  is used to run through a sequence of regions.  $\text{NEXT}(\mathbf{R}, \delta)$  takes a region  $\mathbf{R}$  and returns a region  $\mathbf{R}'$

obtained according to the dynamical system provided in Definition 10. Furthermore, let  $\text{INIT}(\text{Ske}(\mathbf{b}), r, m, \delta)$  be the function that takes a skeleton, the type  $r$  of the region (northern, central, southern), an integer  $m$  and  $\delta$ , and returns the smallest region of type  $r$  and area  $m$ , see the characterization (1) for  $\mathbf{R}_{\min}$ .

The iterator  $\text{NEXTCOL}(P_{<i}, \mathbf{c}, \delta, n)$  takes a column  $\mathbf{c} \in \text{Comp}(P_{<i}, k, n)$  and  $\delta$ , and returns the smallest column  $\mathbf{c}'$  in  $\text{Comp}(P_{<i}, k, n)$  such that  $\mathbf{c} < \mathbf{c}'$  and  $\text{Ske}(\mathbf{c}) = \text{Ske}(\mathbf{c}')$ .  $\text{NEXTCOL}$  works by distinguishing six cases, depending on  $\text{Ske}(\mathbf{c})$ . Let  $\text{DF}(\mathbf{b}) = 1$  if the column  $\mathbf{b}$  is l-detached free, 0 otherwise. The simplest case is when  $\text{Ske}(\mathbf{c})$  is null, hence  $\mathbf{c} = \mathbf{c}^\uparrow$ . Here, the next column is simply  $\text{NEXT}(\mathbf{c}, \delta)$ . Now, consider the case  $\text{Ske}(\mathbf{c}) = (\bar{s})$  where  $\mathbf{c}$  has at most two regions (the case  $\text{Ske}(\mathbf{c}) = (s, t, u)$  is treated in the same way). One has  $\mathbf{c} = \mathbf{c}^\uparrow \cdot (\bar{s}) \cdot \mathbf{c}_\downarrow$ . So, if  $\mathbf{c}^\uparrow = \epsilon$  then  $\mathbf{c}'$  is either  $(\bar{s}) \cdot \mathbf{b}$  where  $\mathbf{b} = \text{NEXT}(\mathbf{c}_\downarrow, \delta)$  (if  $\mathbf{b} \neq \perp$ ), or  $\perp$  (if  $\mathbf{b} = \perp$ ). Otherwise, one has  $\mathbf{c}^\uparrow \neq \epsilon$  and  $\mathbf{c}'$  is either  $\mathbf{c}^\uparrow \cdot (\bar{s}) \cdot \mathbf{e}'$  (if  $\mathbf{e}' = \text{NEXT}(\mathbf{c}_\downarrow, \text{DF}(\mathbf{c}^\uparrow) \cdot \delta) \neq \perp$ ) or  $\mathbf{e} \cdot (\bar{s}) \cdot \mathbf{e}''$  where  $\mathbf{e} = \text{NEXT}(\mathbf{c}^\uparrow, \delta)$  and  $\mathbf{e}'' = \text{INIT}(P_{<i}, \text{south}, A(\mathbf{c}_\downarrow), \text{Ske}(\mathbf{c}), \text{DF}(\mathbf{e}) \cdot \delta)$  (if  $\mathbf{e}' = \perp$  and  $\mathbf{e} \neq \perp$ ), or (if  $\mathbf{e} = \mathbf{e}' = \perp$ )  $\text{INIT}(P_{<i}, \text{north}, A(\mathbf{c}^\uparrow) - 1, \text{Ske}(\mathbf{c}), \delta) \cdot (\bar{s}) \cdot \text{INIT}(P_{<i}, \text{south}, A(\mathbf{c}_\downarrow) + 1, \text{Ske}(\mathbf{c}), 0)$ .  $\text{NEXTCOL}$  works similarly also in the cases where three regions can be present (skeletons of type  $(\bar{s}, \bar{t})$  or  $(s, t, u, v)$ ). Lastly, if  $\text{NEXTCOL}(P_{<i}, \mathbf{c}, \delta, n) = \perp$  then  $\mathbf{c}'$  consists of the smallest skeleton larger than  $\text{Ske}(\mathbf{c})$ , with  $A(\mathbf{c}') = k$ , which is returned by  $\text{NEXTSKEL}(P_{<i}, k, \text{Ske}(\mathbf{c}), \delta)$ . The correctness of all the iterators directly follows from their definition and from the order on skeletons and on columns.

### 4.1 The Data Structure

We represent  $P_{\leq i}$  by an array of  $i$  records, one per column. The record for the  $i$ -th column is a tuple  $(L, U^\uparrow, U^\downarrow, V^\uparrow, V^\downarrow, R_1, R_2, R_3)$ . The first field  $L$  is a doubly-linked list for the sequence of segments representing the column (as many nodes as segments), then we have four links to the nodes associated with the extremes of the

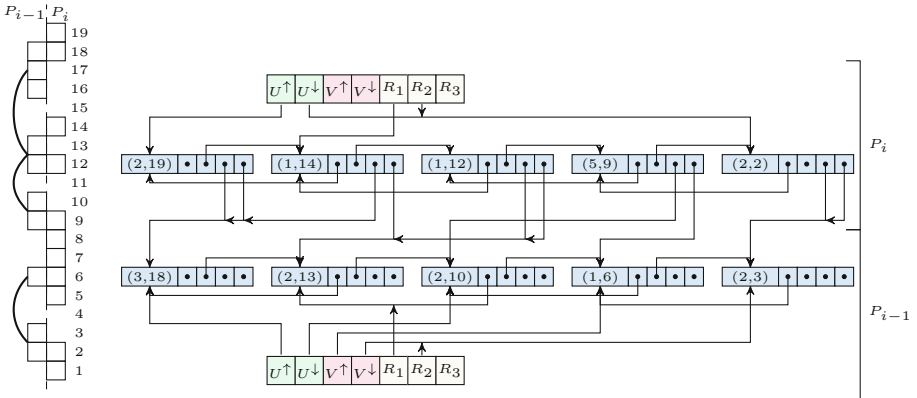


Fig. 5. Two columns (left) and the associated data structure (right).

column (at least one is non-null). Lastly, we have three links to the last node of each region (at least one is non-null). Each node of the list  $L$  corresponds to a segment  $s$  (represented by  $(A(s), \text{Top}(s))$ ) and contains five entries  $(s, l_1, l_2, l_3, l_4)$ , where  $l_1$  is a link to the preceding node (for  $s^\uparrow$ ),  $l_2$  is the link to the next node (for  $s_\downarrow$ ),  $l_3$  and  $l_4$  are the links to the nodes (in the list representing the preceding column) associated with  $^\uparrow s$  and  $_\downarrow s$ , respectively. Given a region  $\mathbf{R}$  of  $P_i$ , by Lemma 1 the move in the position  $\min(\text{Mcr}(P_{\leq i}, \mathbf{R}, \delta))$  regards one of the last three segments of  $\mathbf{R}$ . Notice that this data structure allows the execution of any move (either Shift or Split) in time  $O(1)$  (Lemmas 2 and 3). Figure 5 illustrates the data structure associated with a polyomino.

## 4.2 Complexity

The execution of the algorithm that generates  $\text{Pol}_2(n)$  is represented by a tree  $T(n)$  where an internal node  $v$  at level  $i$  corresponds to a suitable  $i$ -prefix  $P_{\leq i}$ . Furthermore, the children of  $v$  correspond to the  $(i+1)$ -prefixes  $P_{\leq i}|\mathbf{c}$ , for all  $\mathbf{c}$  in  $\bigcup_k \text{Comp}(P_{\leq i}, k, n)$ . Then, the *complexity* of  $v$ , denoted by  $C(v)$ , is the time taken to generate the set  $\bigcup_k \text{Comp}(P_{\leq i}, k, n)$  (as many columns as children of  $v$ ), with  $C(v) = O(1)$  if  $v$  is a leaf. Since  $f(n) = O(g(n))$  and  $g(n) = |\text{Pol}_2(n)|$ , where  $f(n)$  (resp.,  $g(n)$ ) is the number of internal nodes (resp., leaves) of  $T(n)$ , it follows that the algorithm is CAT if one proves that  $C(T(n)) = O(|T(n)|)$ , where  $C(T(n)) = \sum_{v \in T(n)} C(v)$ . Indeed, it holds the following result.

**Theorem 1.**  $C(T(n)) = O(|T(n)|)$ .

## 5 Conclusions

This paper further deepens the new approach to polyominoes generation started in [7], by presenting a new CAT generation algorithm for  $\text{Pol}_2(n)$ , which takes linear space. We strongly believe that all conceptual obstacles have been detected and that our approach might lead to a CAT algorithm working in linear space for the generation of the full set  $\text{Pol}(n)$ .

## References

1. Bousquet-Mélou, M.: A method for the enumeration of various classes of column-convex polygons. *Discrete Math.* **154**(1–3), 1–25 (1996)
2. Brocchi, S., Castiglione, G., Massazza, P.: On the exhaustive generation of  $k$ -convex polyominoes. *Theor. Comput. Sci.* **664**, 54–66 (2017)
3. Castiglione, G., Massazza, P.: An efficient algorithm for the generation of  $Z$ -convex polyominoes. In: Barneva, R.P., Brimkov, V.E., Šlapal, J. (eds.) *IWCIA 2014*. LNCS, vol. 8466, pp. 51–61. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-07148-0\\_6](https://doi.org/10.1007/978-3-319-07148-0_6)
4. Castiglione, G., Restivo, A.: Reconstruction of  $L$ -convex polyominoes. *Electron. Notes Discrete Math.* **12**, 290–301 (2003)

5. Del Lungo, A., Duchi, E., Frosini, A., Rinaldi, S.: On the generation and enumeration of some classes of convex polyominoes. *Electron. J. Comb.* **11**(1) (2004)
6. Delest, M.P., Viennot, G.: Algebraic languages and polyominoes enumeration. *Theor. Comput. Sci.* **34**(1–2), 169–206 (1984)
7. Formenti, E., Massazza, P.: From tetris to polyominoes generation. *Electron. Notes Discrete Math.* **59**, 79–98 (2017)
8. Golomb, S.W.: Checker boards and polyominoes. *Am. Math. Monthly* **61**, 675–682 (1954)
9. Jensen, I.: Enumerations of lattice animals and trees. *J. Stat. Phys.* **102**(3), 865–881 (2001)
10. Jensen, I.: Counting polyominoes: a parallel implementation for cluster computing. In: Sloot, P.M.A., Abramson, D., Bogdanov, A.V., Gorbachev, Y.E., Dongarra, J.J., Zomaya, A.Y. (eds.) ICCS 2003. LNCS, vol. 2659, pp. 203–212. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-44863-2\\_21](https://doi.org/10.1007/3-540-44863-2_21)
11. Mantaci, R., Massazza, P.: On the exhaustive generation of plane partitions. *Theor. Comput. Sci.* **502**, 153–164 (2013)
12. Mantaci, R., Massazza, P.: From linear partitions to parallelogram polyominoes. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 350–361. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22321-1\\_30](https://doi.org/10.1007/978-3-642-22321-1_30)
13. Mantaci, R., Massazza, P., Yunès, J.B.: An efficient algorithm for generating symmetric ice piles. *Theor. Comput. Sci.* **629**(C), 96–115 (2016)
14. Massazza, P.: On the generation of convex polyominoes. *Discrete Appl. Math.* **183**, 78–89 (2015)
15. Privman, V., Barma, M.: Radii of gyration of fully and partially directed lattice animals. *Z. Phys. B Condens. Matter* **57**(1), 59–63 (1984)
16. Privman, V., Forgacs, G.: Exact solution of the partially directed compact lattice animal model. *J. Phys. A Math. Gen.* **20**(8), L543 (1987)
17. Redner, S., Yang, Z.R.: Size and shape of directed lattice animals. *J. Phys. A Math. Gen.* **15**(4), L177 (1982)



# A Local Limit Property for Pattern Statistics in Bicomponent Stochastic Models

Massimiliano Goldwurm<sup>1</sup>, Jianyi Lin<sup>2(✉)</sup>, and Marco Vignati<sup>1</sup>

<sup>1</sup> Dipartimento di Matematica, Università degli Studi di Milano, Milano, Italy

<sup>2</sup> Department of Mathematics, Khalifa University, Abu Dhabi, United Arab Emirates  
jianyi.lin@ku.ac.ae

**Abstract.** We present a non-Gaussian local limit theorem for the number of occurrences of a given symbol in a word of length  $n$  generated at random. The stochastic model for the random generation is defined by a rational formal series with non-negative real coefficients. The result yields a local limit towards a uniform density function and holds under the assumption that the formal series defining the model is recognized by a weighted finite state automaton with two primitive components having equal dominant eigenvalue.

## 1 Introduction

A classical counting problem concerning formal languages is the evaluation of the number of occurrences of a given symbol  $a$  in a word  $w$  of length  $n$  generated at random by a suitable stochastic source. Denoting by  $Y_n$  such a random variable, traditional goals of interest are the asymptotic evaluation of mean value and variance of  $Y_n$  as well as its limit distribution and local limit evaluations for its probability function. Clearly these properties depend on the stochastic model used for generating the random text  $w$ . Classical models are the Bernoulli and Markovian ones [11, 13]. Here we consider the rational stochastic models, which are defined by rational formal series in non-commutative variables with coefficients in  $\mathbb{R}_+$  [3]. In these models the computation of a random word  $w$  can be done easily in linear time [8] once we know a  $\mathbb{R}_+$ -weighted finite state automaton that recognizes the series. Such a probabilistic source is rather general since it includes as special cases the traditional Bernoulli and Markovian models and also encompasses the random generation of words of length  $n$  in any regular language under uniform distribution.

The problem above is of interest for several reasons. First, it has been studied in connection with the analysis of pattern statistics and in particular those occurring in computational biology [3, 11–13]. It turns out that evaluating the frequency of patterns from a regular expression in a random text generated by a Markovian model can be reduced to determining the frequency of a single symbol in a word over a binary alphabet generated by a rational stochastic model [3, 11].



Moreover, it is well-known that the average number of occurrences of symbols in words of regular and context-free languages plays a relevant role in the analysis of the descriptive complexity of languages and computational models [5,6]. Clearly the limit distributions of these quantities (also in the local form) yield a more complete information and in particular they allow to evaluate their dispersion around the average values.

Our problem is also related to the asymptotic estimate of the coefficients of rational formal series in commutative variables. In particular the local limit properties of  $Y_n$  enabled to show that the maximum coefficients of monomials of size  $n$  for some of those series is of the order  $\Theta(n^{k-3/2}\lambda^n)$  for  $\lambda \geq 1$  and any positive  $k \in \mathbb{N}$  [4, Corollary 16]. Similar consequences hold for the study of the degree of ambiguity of rational trace languages (subset of free partially commutative monoids) [3].

The asymptotic behaviour of  $Y_n$  assuming the rational stochastic model defined by a series  $r$  depends on the finite state automata that recognize  $r$ . The main results known in the literature concern the case when such an automaton has a primitive transition matrix. In this case asymptotic expressions for the mean value  $E(Y_n)$  and the variance  $var(Y_n)$  are known. In particular, there is a real value  $\beta$ ,  $0 < \beta < 1$ , such that  $E(Y_n) = \beta n + O(1)$  and a similar result holds for  $var(Y_n)$ . Under the same hypothesis it is also proved that  $Y_n$  has a Gaussian limit distribution (i.e. it satisfies a central limit theorem) and it admits local limit properties intuitively stating that its probability function approximates a normal density [3,4].

The properties of  $Y_n$  have been studied also when the transition matrix consists of two primitive components. A variety of results on the asymptotic behaviour of  $Y_n$  are obtained in this case [7], but none of them concerns local limit properties. Here, we extend the previous results by showing a non-Gaussian local limit theorem that holds assuming that the two components have equal dominant eigenvalue while the main constants of the average value,  $\beta_1$  and  $\beta_2$  (associated with the first and the second component, respectively) are different. Under these hypotheses, it is known that  $Y_n/n$  converges in distribution to a random variable  $U$  uniformly distributed over the interval  $[\min\{\beta_1, \beta_2\}, \max\{\beta_1, \beta_2\}]$  [7]. In the present work, assuming a further natural aperiodicity condition on the transition matrix, we prove that, as  $n$  grows to  $+\infty$  and for any integer expression  $k = k(n)$  such that  $k/n$  converges to a value  $x$  different from  $\beta_1$  and  $\beta_2$ , we have  $nPr(Y_n = k) = f_U(x) + o(1)$ , where  $f_U$  is the density function of the uniform random variable  $U$  defined above.

The proof of our result is based on the analysis of the characteristic function of  $Y_n$  and it is obtained by adapting to our settings the so-called Saddle Point Method, traditionally used for proving Gaussian local limit properties [9].

The material we present is organized as follows. In Sect. 2 we recall the rational stochastic model and other preliminary notions. In Sect. 3 we revisit the properties of  $Y_n$  when the transition matrix of the automaton is primitive (Gaussian case). In Sect. 4 we introduce the bicomponent model and prove the main result

comparing it with the convergence in distribution given in [7]. In the last section we discuss possible extensions and future work.

## 2 Preliminary Notions

In this section we give some preliminary notions and define our problem.

Given the binary alphabet  $\{a, b\}$ , for every word  $w \in \{a, b\}^*$  we denote by  $|w|$  the length of  $w$  and by  $|w|_a$  the number of occurrences of  $a$  in  $w$ . For each  $n \in \mathbb{N}$ , we also represent by  $\{a, b\}^n$  the set  $\{w \in \{a, b\}^* : |w| = n\}$ . A *formal series* in the non-commutative variables  $a, b$  with coefficients in the set  $\mathbb{R}_+$  of non-negative real numbers is a function  $r : \{a, b\}^* \rightarrow \mathbb{R}_+$ , usually represented in the form  $r = \sum_{w \in \{a, b\}^*} (r, w)w$ , where each coefficient  $(r, w)$  is the value of  $r$  at  $w$ . The set  $\mathbb{R}_+ \langle\langle a, b \rangle\rangle$  of all such formal series forms a semiring with respect to the operations of sum and Cauchy product. A series  $r \in \mathbb{R}_+ \langle\langle a, b \rangle\rangle$  is called *rational* [14] if for some integer  $m > 0$  there is a monoid morphism  $\mu : \{a, b\}^* \rightarrow \mathbb{R}_+^{m \times m}$  and two arrays  $\xi, \eta \in \mathbb{R}_+^m$ , such that  $(r, w) = \xi' \mu(w) \eta$ , for every  $w \in \{a, b\}^*$ . In this case, as the morphism  $\mu$  is generated by matrices  $A = \mu(a)$  and  $B = \mu(b)$ , we say that the 4-tuple  $(\xi, A, B, \eta)$  is a *linear representation* [2] of  $r$ .

Now, consider a rational formal series  $r \in \mathbb{R}_+ \langle\langle a, b \rangle\rangle$  with linear representation  $(\xi, A, B, \eta)$  and let  $\mu$  be the morphism generated by  $A$  and  $B$ . Assume that the set  $\{w \in \{a, b\}^n : (r, w) > 0\}$  is not empty for every positive integer  $n$ . Then we can consider the probability measure  $\text{Pr}$  over the set  $\{a, b\}^n$  given by

$$\text{Pr}(w) = \frac{(r, w)}{\sum_{x \in \{a, b\}^n} (r, x)} = \frac{\xi' \mu(w) \eta}{\xi' (A + B)^n \eta} \quad \forall w \in \{a, b\}^n$$

Note that, if  $r$  is the characteristic series of a language  $L \subseteq \{a, b\}^*$  then  $\text{Pr}$  is the uniform probability function over the set  $L \cap \{a, b\}^n$ . Moreover, it is easy to see that any Bernoullian or Markovian source, for the random generation of words in  $\{a, b\}^*$ , produces strings in  $\{a, b\}^n$  with probability  $\text{Pr}$  for some rational series  $r \in \mathbb{R}_+ \langle\langle a, b \rangle\rangle$ . We also recall that there are linear time algorithms that on input  $n$  generate a random word in  $\{a, b\}^n$  according with probability  $\text{Pr}$  [8].

Then we can define the random variable (r.v.)  $Y_n$  representing the number of occurrences of the symbol  $a$  in a word  $w$  chosen at random in  $\{a, b\}^n$  with probability  $\text{Pr}(w)$ . In this work we are interested in the asymptotic properties of  $\{Y_n\}$ . It is clear that, for every  $k \in \{0, 1, \dots, n\}$ ,

$$p_n(k) := \text{Pr}(Y_n = k) = \frac{\sum_{|w|=n, |w|_a=k} (r, w)}{\sum_{w \in \{a, b\}^n} (r, w)}$$

Since  $r$  is rational also the previous probability can be expressed by using its linear representation. It turns out that

$$p_n(k) = \frac{[x^k] \xi' (Ax + B)^n \eta}{\xi' (A + B)^n \eta} \quad \forall k \in \{0, 1, \dots, n\} \tag{1}$$

where  $[x^k]q(x)$  denotes the coefficient of  $x^k$  in a polynomial  $q \in \mathbb{R}[x]$ . For sake of brevity we say that  $Y_n$  is *defined* by the linear representation  $(\xi, A, B, \eta)$ . Then the distribution of each  $Y_n$  can be characterized by function  $h_n(z)$  given by

$$h_n(z) = \xi'(Ae^z + B)^n \eta$$

Indeed, setting  $M = A + B$ , the moment generating function of  $Y_n$  is given by

$$F_n(z) = \sum_{k=0}^n p_n(k) e^{zk} = \frac{\xi'(Ae^z + B)^n \eta}{\xi' M^n \eta} = \frac{h_n(z)}{h_n(0)} \quad \forall z \in \mathbb{C}$$

This leads to determine mean value and variance of  $Y_n$  as

$$E(Y_n) = F'_n(0) = \frac{h'_n(0)}{h_n(0)}, \quad Var(Y_n) = \frac{h''_n(0)}{h_n(0)} - \left( \frac{h'_n(0)}{h_n(0)} \right)^2 \quad (2)$$

Analogously, the characteristic function of  $Y_n$  is given by

$$\Psi_n(t) = \sum_{k=0}^n p_n(k) e^{itk} = \frac{\xi'(Ae^{it} + B)^n \eta}{\xi' M^n \eta} = \frac{h_n(it)}{h_n(0)} \quad \forall t \in \mathbb{R} \quad (3)$$

It turns out that the limit distribution of  $Y_n$  depends on the properties of the matrix  $M = A + B$ . A relevant case occurs when  $M$  is primitive (i.e.  $\exists k \in \mathbb{N} : M^k > 0$ ). In this case it is known that  $Y_n$  has a Gaussian limit distribution [1, 3] and satisfies a local limit theorem (in the sense of De Moivre - Laplace Theorem [10]) we recall in the next section.

### 3 Primitive Case

In this section we consider the case when  $M = A + B$  is a primitive matrix [15] and recall some properties proved in [3, 4, 11] that are useful in the sequel.

Since  $M$  is primitive, by Perron-Frobenius Theorem, it admits a real eigenvalue  $\lambda > 0$  that is greater than the modulus of any other eigenvalue of  $M$ . Thus, we can consider the function  $u = u(z)$  implicitly defined by the equation

$$\text{Det}(Iu - Ae^z - B) = 0$$

such that  $u(0) = \lambda$ . It turns out that, in a neighbourhood of  $z = 0$ ,  $u(z)$  is analytic, is a simple root of the characteristic polynomial of  $Ae^z + B$  and  $|u(z)|$  is strictly greater than the modulus of all other eigenvalues of  $Ae^z + B$ .

Now, consider the bivariate matrix-valued function  $H(x, y)$  given by

$$H(x, y) = \sum_{n=0}^{+\infty} (Ax + B)^n y^n = (I - (Ax + B)y)^{-1}$$

Clearly,  $\xi' H(e^z, y) \eta$  is the generating function of  $\{h_n(z)\}_n$ , i.e.

$$\xi' H(e^z, y) \eta = \sum_{n=0}^{+\infty} h_n(z) y^n = \frac{\xi' \text{Adj}(I - (Ae^z + B)y) \eta}{\text{Det}(I - (Ae^z + B)y)}$$

Thus, for every  $z$  near 0, the singularities of  $\xi'H(e^z, y)\eta$  are the values  $\mu^{-1}$  for all (non-null) eigenvalues  $\mu$  of  $Ae^z + B$  and hence  $u(z)^{-1}$  is its (unique) singularity of minimum modulus. Then, by the properties of  $u(z)$  one can get the following

**Proposition 1** ([3]). *If  $M$  is primitive then there are two positive constants  $c, \rho$  and a function  $r(z)$  analytic and non-null at  $z = 0$ , such that for every  $|z| \leq c$*

$$h_n(z) = r(z) u(z)^n + O(\rho^n)$$

and  $\rho < |u(z)|$ . In particular  $\rho < \lambda$ .

Mean value and variance of  $Y_n$  can be estimated from Eq. (2). It turns out that the constants  $\beta = u'(0)/\lambda$  and  $\gamma = \frac{u''(0)}{\lambda} - \left(\frac{u'(0)}{\lambda}\right)^2$  are positive and satisfy the equalities  $E(Y_n) = \beta n + O(1)$  and  $var(Y_n) = \gamma n + O(1)$  [3]. Explicit expressions of  $\beta$  and  $\gamma$  are also obtained in [3] that depend on the matrices  $A, M$ , and in particular on  $\lambda$  and the corresponding left and right eigenvectors.

Other properties concern the function  $y(t) = u(it)/\lambda$  used in Sect. 4, defined for real  $t$  in a neighbourhood of 0. By Proposition 1, for any  $t$  near 0,  $y(t)^n$  is the leading term of the characteristic function  $\Psi_n(t)$ . Moreover, for some  $c > 0$  and every  $|t| \leq c$ , the following relations hold [3]<sup>(1)</sup>:

$$|y(t)| = 1 - \frac{\gamma}{2}t^2 + O(t^4), \quad \arg y(t) = \beta t + O(t^3), \quad |y(t)| \leq e^{-\frac{\gamma}{4}t^2} \quad (4)$$

The behaviour of  $y(t)$  can be estimated precisely when  $t$  tends to 0. For any  $q$  such that  $1/3 < q < 1/2$  it can be proved that

$$y(t)^n = e^{-\frac{\gamma}{2}t^2n + i\beta tn} (1 + O(t^3)n) \quad \text{for } |t| \leq n^{-q} \quad (5)$$

The previous properties can be used to prove a local limit theorem for  $\{Y_n\}$  when  $M$  is primitive [3]. The result holds under a further assumption (introduced to avoid periodicity phenomena) stating that for every  $0 < t < 2\pi$ .

$$|\mu| < \lambda \quad \text{for every eigenvalue } \mu \text{ of } Ae^{it} + B \quad (6)$$

Such a property is studied in detail in [4] and is often verified. For instance it holds true whenever there are two indices  $i, j$  such that  $A_{ij} > 0$  and  $B_{ij} > 0$ , or  $A_{ii} > 0$  and  $B_{jj} > 0$ . Intuitively, it corresponds to an aperiodicity property of the oriented graph defined by matrices  $A$  and  $B$  concerning the number of occurrences of the label  $a$  in cycles of equal length.

The local limit theorem in the primitive case can be stated as follows.

**Theorem 2.** *Let  $\{Y_n\}$  be defined by a linear representation  $(\xi, A, B, \eta)$  such that the matrix  $M = A + B$  is primitive and assume that property (6) holds for every  $0 < t < 2\pi$ . Moreover, let  $\beta$  and  $\gamma$  be defined as above. Then, as  $n$  tends to  $+\infty$ , the following equation holds uniformly for every  $k = 0, 1, \dots, n$ ,*

$$Pr\{Y_n = k\} = \frac{e^{-\frac{(k-\beta n)^2}{2\gamma n}}}{\sqrt{2\pi\gamma n}} + o\left(\frac{1}{\sqrt{n}}\right) \quad (7)$$

---

<sup>1</sup> Here, for every interval  $I \subseteq \mathbb{R}$  and functions  $f, g : I \rightarrow \mathbb{C}$ , by “ $g(t) = O(f(t))$  for  $t \in I$ ” we mean “ $|g(t)| \leq b|f(t)|$  for all  $t \in I$ ”, for some constant  $b > 0$ .

### 4 Bicomponent Models

In this section we study the behaviour of  $\{Y_n\}_{n \in \mathbb{N}}$  in the bicomponent model. We first recall some notions and properties introduced in [7] for this model: in particular we need a sort of analogous of Proposition 1 in this case.

Here  $\{Y_n\}_{n \in \mathbb{N}}$  is defined by a linear representation  $(\xi, A, B, \eta)$  of size  $m$ , such that the matrix  $M = A + B$  consists of two primitive components. More precisely, there are two linear representations  $(\xi_1, A_1, B_1, \eta_1)$ ,  $(\xi_2, A_2, B_2, \eta_2)$ , of size  $m_1$  and  $m_2$ , respectively, with  $m = m_1 + m_2$ , such that for some  $A_0, B_0 \in \mathbb{R}_+^{m_1 \times m_2}$

$$\xi' = (\xi'_1, \xi'_2), \quad A = \begin{pmatrix} A_1 & A_0 \\ 0 & A_2 \end{pmatrix}, \quad B = \begin{pmatrix} B_1 & B_0 \\ 0 & B_2 \end{pmatrix}, \quad \eta = \begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix} \quad (8)$$

Moreover we assume the following conditions:

- (A) The matrices  $M_1 = A_1 + B_1$  and  $M_2 = A_2 + B_2$  are primitive and we denote by  $\lambda_1$  and  $\lambda_2$  the corresponding Perron-Frobenius eigenvalues;
- (B)  $\xi_1 \neq 0 \neq \eta_2$  and  $A_0 + B_0 \neq 0$ ;
- (C)  $A_1 \neq 0 \neq B_1$  and  $A_2 \neq 0 \neq B_2$ .

Since the two components are primitive the properties presented in the previous section hold for each of them. In particular, for  $j = 1, 2$ , we can define  $H^{(j)}(x, y)$ ,  $h_n^{(j)}(z)$ ,  $u_j(z)$ ,  $y_j(t)$ ,  $\beta_j$ , and  $\gamma_j$ , respectively, as the values  $H(x, y)$ ,  $h_n(z)$ ,  $u(z)$ ,  $y(t)$ ,  $\beta$ ,  $\gamma$  referred to component  $j$ . Note that condition (C) guarantees that  $0 < \beta_j < 1$  and  $0 < \gamma_j$  for every  $j = 1, 2$ , while condition (B) implies that both components contribute to probability values of  $Y_n$ .

In such a bicomponent model the limit distribution of  $\{Y_n\}$  mainly depends on whether  $\lambda_1 \neq \lambda_2$  or  $\lambda_1 = \lambda_2$ . If  $\lambda_1 > \lambda_2$  then  $\frac{Y_n - \beta_1 n}{\sqrt{\gamma_1 n}}$  converges in distribution to a standard normal r.v. (the case  $\lambda_1 < \lambda_2$  is symmetric) [7]. If  $\lambda_1 = \lambda_2$  and  $\beta_1 \neq \beta_2$  then  $Y_n/n$  converges in distribution to a random variable  $U$  uniformly distributed over the interval  $[b_1, b_2]$ , where  $b_1 = \min\{\beta_1, \beta_2\}$  and  $b_2 = \max\{\beta_1, \beta_2\}$  [7, Theorem 15]. This means that, for every  $x \in \mathbb{R}$ ,

$$\lim_{n \rightarrow +\infty} \Pr(Y_n/n \leq x) = \Pr(U \leq x) \quad (9)$$

However this relation does not give information about the probability that  $Y_n$  takes a specific value  $k \in \mathbb{N}$  (possibly depending on  $n$ ). Here we want to show that adding a further condition on the model such a probability can be estimated at least for reasonable expressions  $k = k(n)$ . To this end, we still consider the case  $\lambda_1 = \lambda_2$  and  $\beta_1 \neq \beta_2$  and assume a further hypothesis analogous to condition (6): for every  $0 < t < 2\pi$

$$|\mu| < \lambda \text{ for all eigenvalues } \mu \text{ of the matrices } A_1 e^{it} + B_1 \text{ and } A_2 e^{it} + B_2 \quad (10)$$

where we set  $\lambda = \lambda_1 = \lambda_2$ .

In this case, following [7], the matrix-valued function  $H(x, y)$  is given by

$$H(x, y) = \sum_{n=0}^{+\infty} (Ax + B)^n y^n = \begin{bmatrix} H^{(1)}(x, y) & G(x, y) \\ 0 & H^{(2)}(x, y) \end{bmatrix}, \quad \text{where}$$

$$H^{(1)}(x, y) = \frac{\text{Adj}(I - (A_1x + B_1)y)}{\text{Det}(I - (A_1x + B_1)y)}, \quad H^{(2)}(x, y) = \frac{\text{Adj}(I - (A_2x + B_2)y)}{\text{Det}(I - (A_2x + B_2)y)}$$

$$\text{and } G(x, y) = H^{(1)}(x, y) (A_0x + B_0)y H^{(2)}(x, y). \quad (11)$$

Thus, the generating function of  $\{h_n(z)\}_n$  is now given by

$$\sum_{n=0}^{\infty} h_n(z)y^n = \xi' H(e^z, y)\eta = \xi'_1 H^{(1)}(e^z, y)\eta_1 + \xi'_1 G(e^z, y)\eta_2 + \xi'_2 H^{(2)}(e^z, y)\eta_2$$

An analysis of the singularities of  $\xi' H(e^z, y)\eta$  is presented in [7, Sect. 7.2] where the following property is proved.

**Proposition 3.** *For some constant  $c > 0$  and every  $z \in \mathbb{C}$  such that  $|z| \leq c$ , we have*

$$h_n(z) = s(z) \sum_{j=0}^{n-1} u_1(z)^j u_2(z)^{n-1-j} + O(u_1(z)^n) + O(u_2(z)^n)$$

where  $s(z)$  is a function analytic and non-null for  $|z| \leq c$ .

Since  $u_1(0) = \lambda = u_2(0)$  the previous proposition implies

$$h_n(0) = s(0)n\lambda^{n-1} + O(\lambda^n) \quad (s(0) \neq 0) \quad (12)$$

### 4.1 Analysis of the Characteristic Function

Here we study the characteristic function  $\Psi_n(t) = \frac{h_n(it)}{h_n(0)}$ , for  $-\pi \leq t \leq \pi$ . We split this interval in three sets:

$$|t| \leq n^{-q}, \quad n^{-q} < |t| < c, \quad c \leq |t| \leq \pi$$

where  $c$  and  $q$  are positive constants and  $\frac{1}{3} < q < \frac{1}{2}$ . We observe that such a splitting is typical of the ‘‘Saddle Point Method’’, and it is often used to derive local limit properties in the Gaussian case [9].

**Proposition 4.** *For every  $0 < c < \pi$  there exists  $0 < \varepsilon < 1$  such that*

$$|\Psi_n(t)| = O(\varepsilon^n) \quad \text{for all } c \leq |t| \leq \pi.$$

*Proof.* From Eq. (11) it is clear that, for every  $z \in \mathbb{C}$ , the singularities of the generating function  $\xi' H(e^z, y)\eta$  are the inverses of the eigenvalues of the matrices  $(A_1e^z + B_1)$  and  $(A_2e^z + B_2)$ . Then, by condition (10), for every  $0 < c < \pi$ , all singularities of  $\xi' H(e^{it}, y)\eta$ , for any  $c \leq |t| \leq \pi$ , are in modulus greater than a value  $\tau^{-1}$  such that  $0 < \tau < \lambda$ , and hence  $|h_n(it)| = O(\tau^n)$ . Thus, by equality (12), for some  $0 < \varepsilon < 1$  we have

$$|\Psi_n(t)| = \left| \frac{h_n(it)}{h_n(0)} \right| = \frac{O(\tau^n)}{\Theta(n\lambda^n)} = O(\varepsilon^n) \quad \text{for any } c \leq |t| \leq \pi \quad \square$$

Now, let us study  $\Psi_n(t)$  for  $t$  in a neighbourhood of 0. We recall that in such a set both functions  $y_1(t) = u_1(it)/\lambda$  and  $y_2(t) = u_2(it)/\lambda$  satisfy equations (4). Then, for some  $c > 0$  and every  $|t| \leq c$ , we have

$$y_1(t) = 1 + i\beta_1 t + O(t^2), \quad y_2(t) = 1 + i\beta_2 t + O(t^2) \tag{13}$$

$$|y_1(t)| \leq e^{-\frac{\gamma_1}{4}t^2}, \quad |y_2(t)| \leq e^{-\frac{\gamma_2}{4}t^2} \tag{14}$$

Moreover, by Proposition 3 we immediately get, for  $|t| \leq c$ , with  $t \neq 0$ ,

$$h_n(it) = s(it) \frac{u_1(it)^n - u_2(it)^n}{u_1(it) - u_2(it)} + O(u_1(it)^n) + O(u_2(it)^n)$$

Thus from equalities (12), (13) and (14), we have

$$\Psi_n(t) = \frac{h_n(it)}{h_n(0)} = (1 + O(t)) \left( \frac{y_1(t)^n - y_2(t)^n}{it(\beta_1 - \beta_2)n} \right) + \sum_{j=1,2} O\left(\frac{e^{-\frac{\gamma_j}{4}t^2 n}}{n}\right) \tag{15}$$

This leads to evaluate  $\Psi_n(t)$  in the second set, i.e. for  $n^{-q} < |t| < c$ .

**Proposition 5.** *Let  $0 < q < 1/2$ . Then there are two positive constants  $a, c$  such that, for every real  $t$  satisfying  $n^{-q} < |t| < c$ ,*

$$|\Psi_n(t)| = O\left(e^{-an^{1-2q}}\right)$$

*Proof.* From Eq. (15), taking  $a = \min\{\gamma_1, \gamma_2\}/4$ , we obtain for some  $c > 0$

$$|\Psi_n(t)| \leq \frac{|y_1(t)|^n + |y_2(t)|^n}{\Theta(n^{1-q})} + O\left(e^{-at^2 n}/n\right) \quad \text{for all } n^{-q} < |t| < c$$

and by (14) we get  $|\Psi_n(t)| = O\left(n^{q-1}e^{-an^{1-2q}}\right)$  proving the result. □

Now, let us evaluate  $\Psi_n(t)$  in the first set, that is for  $|t| \leq n^{-q}$  where  $1/3 < q < 1/2$ . First note that, by simple computations, the following relations can be proved:

$$\int_{|t| \leq n^{-q}} O\left(e^{-\frac{\gamma_j}{4}t^2 n}/n\right) dt = O(n^{-1-q}) = o(n^{-4/3}) \quad \text{for } j = 1, 2,$$

$$\int_{|t| \leq n^{-q}} O(t) \frac{y_1(t)^n - y_2(t)^n}{it(\beta_1 - \beta_2)n} dt = \int_{|t| \leq n^{-q}} O(1/n) dt = o(n^{-4/3})$$

Therefore, by Eq. (15), for every  $k \in \{0, 1, \dots, n\}$  we get

$$\int_{|t| \leq n^{-q}} \Psi_n(t) e^{-ikt} dt = \int_{|t| \leq n^{-q}} \left( \frac{y_1(t)^n - y_2(t)^n}{it(\beta_1 - \beta_2)n} \right) e^{-ikt} dt + o(n^{-4/3}) \tag{16}$$

Also observe that both  $y_1(t)$  and  $y_2(t)$  satisfy Eq. (5), whence

$$y_j(t)^n = e^{-\frac{\gamma_j}{2}t^2 n + i\beta_j t n} (1 + O(t^3)n) \quad \text{for all } |t| \leq n^{-q}, \quad j = 1, 2$$

Thus, replacing these values in (16), after some computations (similar to the previous ones) we obtain the following

**Proposition 6.** *For every  $k \in \{0, 1, \dots, n\}$  and every  $1/3 < q < 1/2$  we have*

$$\int_{|t| \leq n^{-q}} \Psi_n(t) e^{-ikt} dt = \int_{|t| \leq n^{-q}} \left( \frac{e^{-\frac{\gamma_1}{2} t^2 n + i\beta_1 t n} - e^{-\frac{\gamma_2}{2} t^2 n + i\beta_2 t n}}{it(\beta_1 - \beta_2)n} \right) e^{-ikt} dt + o(1/n)$$

### 4.2 Main Result

Without loss of generality assume  $\beta_1 < \beta_2$ , and denote by  $f_U(x)$  the density function of a uniform r.v.  $U$  in the interval  $[\beta_1, \beta_2]$ , that is

$$f_U(x) = \frac{1}{\beta_2 - \beta_1} \chi_{[\beta_1, \beta_2]}(x) \quad \forall x \in \mathbb{R}$$

where  $\chi_I$  denotes the indicator function of the interval  $I \subset \mathbb{R}$ .

For our purpose we need the following property.

**Lemma 7.** *For  $k, m \in \mathbb{N}$ ,  $k < m$ , let  $g : [2k\pi, 2m\pi] \rightarrow \mathbb{R}_+$  be a monotone function, and let  $I_{k,m} = \int_{2k\pi}^{2m\pi} g(x) \sin(x) dx$ . Then:*

- (a) *if  $g$  is non-increasing we have  $0 \leq I_{k,m} \leq 2[g(2k\pi) - g(2m\pi)]$ ;*
  - (b) *if  $g$  is non-decreasing we have  $2[g(2k\pi) - g(2m\pi)] \leq I_{k,m} \leq 0$ .*
- In both cases  $|I_{k,m}| \leq 2|g(2k\pi) - g(2m\pi)|$ .*

*Proof.* If  $g$  is non-increasing, for each integer  $k \leq j < m$  the following relations hold

$$I_{j,j+1} = \int_{2j\pi}^{(2j+1)\pi} g(x) \sin(x) dx - \int_{(2j+1)\pi}^{2(j+1)\pi} g(x) |\sin(x)| dx \leq 2[g(2j\pi) - g(2(j+1)\pi)]$$

while  $0 \leq I_{j,j+1}$  is obvious. Thus (a) follows by summing the expressions above for  $j = k, \dots, m - 1$ .

Part (b) follows by applying (a) to the function  $h(x) = g(2m\pi) - g(x)$ . □

Now, we can state our main result.

**Theorem 8.** *Let  $(\xi, A, B, \eta)$  be a linear representation of the form (8) satisfying conditions (A), (B), (C) above; also assume  $\lambda_1 = \lambda_2$ ,  $\beta_1 \neq \beta_2$  together with the aperiodicity condition (10). Then, the r.v.  $Y_n$  satisfies the relation*

$$\lim_{n \rightarrow +\infty} n \Pr\{Y_n = k\} = f_U(x) \tag{17}$$

*for every integer  $k = k(n)$ , provided that  $k/n \rightarrow x$  for a constant  $x$  such that  $\beta_1 \neq x \neq \beta_2$  (as  $n \rightarrow +\infty$ ).*

*Proof.* It is known [10] that the probability  $p_n(k) = \Pr\{Y_n = k\}$ , for every  $k \in \{0, 1, \dots, n\}$ , can be obtained from  $\Psi_n(t)$  by the inversion formula

$$p_n(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \Psi_n(t) e^{-itk} dt$$



To evaluate the integral above let us split the interval  $[-\pi, \pi]$  into the three sets

$$[-n^{-q}, n^{-q}], \quad \{t \in \mathbb{R} : n^{-q} < |t| < c\}, \quad \{t \in \mathbb{R} : c \leq |t| \leq \pi\}$$

with  $c$  as in Proposition 5 and some  $1/3 < q < 1/2$ . Then, by Propositions 4, 5, 6, we obtain

$$p_n(k) = \frac{1}{2\pi} \int_{|t| \leq n^{-q}} \left( \frac{e^{-\frac{\gamma_2}{2} t^2 n + i\beta_2 t n} - e^{-\frac{\gamma_1}{2} t^2 n + i\beta_1 t n}}{it(\beta_2 - \beta_1)n} \right) e^{-ikt} dt + o(1/n) \quad (18)$$

Now, set  $v = k/n$  and note that for  $n \rightarrow +\infty$ ,  $v$  approaches a value different from  $\beta_1$  and  $\beta_2$ . Thus, defining

$$\Delta_n(v) = \int_{|t| \leq n^{-q}} \frac{e^{i(\beta_2 - v)tn - \frac{\gamma_2}{2} t^2 n} - e^{i(\beta_1 - v)tn - \frac{\gamma_1}{2} t^2 n}}{i(\beta_2 - \beta_1)t} dt$$

we have to prove that

$$\Delta_n(v) = 2\pi f_U(v) + o(1) \quad (19)$$

Since  $\beta_1 < \beta_2$  set  $\delta = \beta_2 - \beta_1$ . Then,  $\Delta_n(v)$  is the integral of the difference of two functions of the form

$$A_n(t, v) = \frac{e^{i(\beta - v)tn - \frac{\gamma}{2} t^2 n} - 1}{i\delta t}$$

where  $\beta$  and  $\gamma$  take the values  $\beta_2, \gamma_2$  and  $\beta_1, \gamma_1$ , respectively. Using the symmetries of real and imaginary part of  $A_n$ , by a change of variable we get

$$\begin{aligned} \int_{|t| \leq n^{-q}} A_n(t, v) dt &= \frac{2}{\delta} \int_0^{n^{-q}} \frac{e^{-\frac{\gamma}{2} t^2 n} \sin((\beta - v)tn)}{t} dt \\ &= \frac{2}{\delta} \int_0^{(\beta - v)n^{1-q}} \frac{\sin(u)}{u} du - \frac{2}{\delta} \int_0^{(\beta - v)n^{1-q}} \left( 1 - e^{-\frac{\gamma u^2}{2(\beta - v)^2 n}} \right) \frac{\sin(u)}{u} du \quad (20) \end{aligned}$$

As  $\int_0^{+\infty} \frac{\sin(u)}{u} du = \pi/2$ , for  $n \rightarrow +\infty$  the first term converges to  $\frac{\pi}{\delta} \operatorname{sgn}(\beta - v)$ . Now we show that the second term of (20) tends to 0 as  $n \rightarrow +\infty$ . This term is equal to

$$\frac{2}{\delta} \int_0^{(\beta - v)n^{1-q}} B_n(u) \sin(u) du \quad (21)$$

where  $B_n(u) = u^{-1} \left( 1 - e^{-\frac{\gamma u^2}{2(\beta - v)^2 n}} \right)$ . To evaluate (21) we use Lemma 7. Note that  $B_n(u) > 0$  for all  $u > 0$ , and  $\lim_{u \rightarrow 0} B_n(u) = 0 = \lim_{u \rightarrow +\infty} B_n(u)$ . Moreover in the set  $(0, +\infty)$  its derivative is null only at the point  $u_n = \alpha|\beta - v|\sqrt{n/\gamma}$ , for a constant  $\alpha \in (1, 2)$  independent of  $n$  and  $v$ . Thus, for  $n$  large enough,  $u_n$  belongs to the interval  $(0, |\beta - v|n^{1-q})$ ,  $B_n(u)$  is increasing in the set  $(0, u_n)$  and decreasing in  $(u_n, +\infty)$ , while its maximum value is

$$B_n(u_n) = \frac{1 - e^{-\frac{\alpha^2}{2}}}{\alpha|\beta - v|} \sqrt{\frac{\gamma}{n}} = \Theta(n^{-1/2})$$

Defining  $k_n = \lfloor \frac{u_n}{2\pi} \rfloor$  and  $K = \lfloor \frac{|\beta-v|n^{1-q}}{2\pi} \rfloor$ , we can apply Lemma 7 to the intervals  $[0, 2k_n]$  and  $[2k_n + 2, 2K]$ , to get

$$\begin{aligned} & \left| \int_0^{|\beta-v|n^{1-q}} B_n(u) \sin u \, du \right| \leq 2B_n(2k_n\pi) + \left| \int_{2k_n\pi}^{2(k_n+1)\pi} B_n(u) \sin u \, du \right| \\ & \quad + 2[B_n(2(k_n + 1)\pi) - B_n(2K\pi)] + \int_{2K\pi}^{|\beta-v|n^{1-q}} B_n(u) \sin u \, du \\ & \leq 2B_n(2k_n\pi) + 2B_n(u_n) + 2[B_n(2(k_n + 1)\pi) - B_n(2K\pi)] + 2B_n(2K\pi) \\ & \leq 6B_n(u_n) = \frac{c \sqrt{\gamma}}{|\beta - v| \sqrt{n}} \end{aligned}$$

where  $c$  is a positive constant independent of  $v$  and  $n$ .

This implies that, for any  $v$  approaching a constant different from  $\beta_1$  and  $\beta_2$ , the second term of (20) is  $O(n^{-1/2})$ . Therefore, we get

$$\begin{aligned} \Delta_n(v) &= \frac{2}{\delta} \left[ \int_0^{(\beta_2-v)n^{1-q}} \frac{\sin u}{u} \, du - \int_0^{(\beta_1-v)n^{1-q}} \frac{\sin u}{u} \, du \right] + O(n^{-1/2}) \\ &= \frac{\pi}{\delta} [\operatorname{sgn}(\beta_2 - v) - \operatorname{sgn}(\beta_1 - v)] + o(1) = 2\pi f_U(v) + o(1) \end{aligned}$$

which proves Eq. (19) and the proof is complete. □

A typical consequence of this result is that  $n\Pr(Y_n = \lfloor xn \rfloor)$  converges to  $f_U(x)$  for every real  $x$  different from  $\beta_1$  and  $\beta_2$ . Intuitively equalities of the form (17) are considered more precise than convergence in distribution since they estimate the probability that the  $n$ -th random variable of the sequence takes a specific value rather than lying on an interval.

On the other hand we observe that (without condition (10)) the convergence in distribution (9) does not imply our equality (17). In particular if there are periodicity phenomena in the occurrences of letter  $a$  it may happen that (9) holds while (17) does not. For instance if the overall series  $r$  of linear representation  $(\xi, A, B, \eta)$  has non-zero coefficients  $(r, w)$  only for words  $w$  with even  $|w|_a$ , then  $\Pr(Y_n = k) = 0$  for all odd integers  $k$ , and hence (17) cannot hold while (9) may still be valid. This observation also shows that condition (10) prevents such periodicity phenomena in the stochastic model.

## 5 Conclusions

In this work we have presented a non-Gaussian local limit property for the number of occurrences of a symbol in words generated at random according with a rational stochastic model defined by a linear representation with two primitive components. Our result concerns the case when the two components have the same dominant eigenvalue but different main constants of the respective mean value ( $\beta_1$  and  $\beta_2$ ). We expect that in case of different dominant eigenvalues a



Gaussian local limit property holds, where the main terms of mean value and variance correspond to the dominant component. On the contrary, we conjecture that results similar to ours (that is of a non-Gaussian type) hold for other rational stochastic models, defined by assuming different hypotheses on the key parameters associated to mean value and variance of the statistic of interest (e.g.  $\beta_1 = \beta_2$ ), or assuming more than two primitive components with equal dominant eigenvalues.

## References

1. Bender, E.A.: Central and local limit theorems applied to asymptotic enumeration. *J. Comb. Theory* **15**, 91–111 (1973)
2. Berstel, J., Reutenauer, C.: *Rational Series and Their Languages*. Springer, New York (1988)
3. Bertoni, A., Choffrut, C., Goldwurm, M., Lonati, V.: On the number of occurrences of a symbol in words of regular languages. *Theoret. Comput. Sci.* **302**, 431–456 (2003)
4. Bertoni, A., Choffrut, C., Goldwurm, M., Lonati, V.: Local limit properties for pattern statistics and rational models. *Theory Comput. Syst.* **39**, 209–235 (2006)
5. Broda, S., Machiavelo, A., Moreira, N., Reis, R.: A hitchhiker’s guide to descriptive complexity through analytic combinatorics. *Theory Comput. Syst.* **528**, 85–100 (2014)
6. Broda, S., Machiavelo, A., Moreira, N., Reis, R.: On the average complexity of strong star normal form. In: Pighizzini, G., Câmpeanu, C. (eds.) *DCFS 2017*. LNCS, vol. 10316, pp. 77–88. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-60252-3\\_6](https://doi.org/10.1007/978-3-319-60252-3_6)
7. de Falco, D., Goldwurm, M., Lonati, V.: Frequency of symbol occurrences in bicomponent stochastic models. *Theoret. Comput. Sci.* **327**(3), 269–300 (2004)
8. Denise, A.: Génération aléatoire uniforme de mots de langages rationnels. *Theoret. Comput. Sci.* **159**, 43–63 (1996)
9. Flajolet, P., Sedgewick, R.: *Analytic Combinatorics*. Cambridge University Press, Cambridge (2009)
10. Gnedenko, B.V.: *Theory of Probability*. Gordon and Breach Science Publisher, Amsterdam (1997)
11. Nicodeme, P., Salvy, B., Flajolet, P.: Motif statistics. *Theoret. Comput. Sci.* **287**(2), 593–617 (2002)
12. Prum, B., Rudolphe, F., Turckheim, E.: Finding words with unexpected frequencies in deoxyribonucleic acid sequence. *J. Roy. Stat. Soc. Ser. B* **57**, 205–220 (1995)
13. Régnier, M., Szpankowski, W.: On pattern frequency occurrences in a Markovian sequence. *Algorithmica* **22**(4), 621–649 (1998)
14. Salomaa, A., Soittola, M.: *Automata-Theoretic Aspects of Formal Power Series*. Springer, New York (1978). <https://doi.org/10.1007/978-1-4612-6264-0>
15. Seneta, E.: *Non-negative Matrices and Markov Chains*. Springer, New York (1981). <https://doi.org/10.1007/0-387-32792-4>



# Linear-Time Limited Automata

Bruno Guillon<sup>(✉)</sup>  and Luca Prigioniero 

Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy  
{guillonb,prigioniero}@di.unimi.it

**Abstract.** The time complexity of 1-limited automata is investigated from a descriptional complexity view point. Though the model recognizes regular languages only, it may use quadratic time in the input length. We show that, with a polynomial increase in size and preserving determinism, each 1-limited automaton can be transformed into an halting linear-time equivalent one. We also obtain polynomial transformations into related models, including weight-reducing Hennie machines, and we show exponential gaps for converse transformations in the deterministic case.

## 1 Introduction

One classical topic of computer science is the investigation of computational models operating under restrictions. Finite automata or pushdown automata, for instance, can be considered as particular Turing machines in which the access to memory storage is limited. Other kinds of restrictions follow from finer analysis of the computational resources an abstract device requires to recognize certain languages. For example, in the case of Turing machines, classical complexity classes such as P, NP, LOGSPACE, *etc.* are defined by introducing a limit on the amount of resources, namely time or space, at disposal of the model.

Usually, such limitations reduce the expressive power. For instance, it is well-known that one-tape nondeterministic Turing machines operating within a space bounded by the length of the input, namely *linear bounded automata*, capture exactly the class of *context-sensitive languages*, *e.g.* [5]. Phenomena like this, where limiting an abstract model reduces its expressiveness to the level of some standard class, are of great interest, as they provide alternative characterizations of standard classes. Another example of this kind has been observed by Hennie in 1965. He indeed proved that deterministic one-tape Turing machines operating in *linear time* (*i.e.*, time  $O(m)$  over inputs of length  $m$ ), which can be converted into linear bounded automata operating in linear time, recognize exactly the class of *regular languages* [3]. The result has then been extended to the nondeterministic case [15], see also [8] for further improvements.<sup>1</sup> As a consequence, each *Hennie machine*, namely nondeterministic linear bounded automata working in linear time, is equivalent to some finite automaton. From the opposite point of view, this means that providing *two-way finite automata*

<sup>1</sup> In nondeterministic linear-time devices each accepting computation has linear length.

with the ability to overwrite the tape cells does not extend the expressiveness of the model, as long as the time is linearly bounded in the length of the input.

Unfortunately, Hennie proved that it is undecidable, given a deterministic one-tape Turing machine, to check whether it works in linear time over all input strings, namely, whether it is actually a Hennie machine. To avoid this drawback, Průša proposed a variant of Hennie machine, called *weight-reducing Hennie machine*, in which the time limitation is syntactic [12]. In this model, each visit of a cell should overwrite its content with a symbol in a decreasing way, with respect to some fixed order on the working alphabet. As a consequence, the number of visits of a cell by the head is bounded by some constant (*i.e.*, not depending on the input length) whence the device works in linear time over every input string.

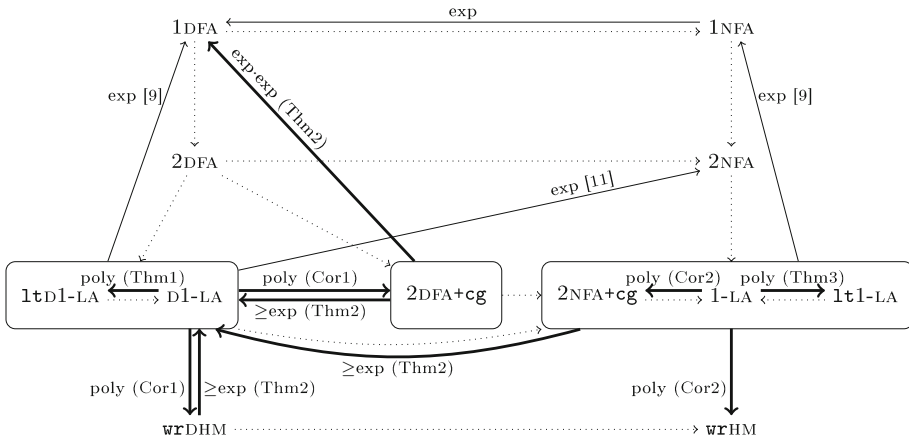
By contrast to Hennie machines, the *d-scan limited automata* (or simply *d-limited automata*) introduced by Hibbard, restrict nondeterministic linear bounded automata by allowing overwriting of each tape cell during its first  $d$  visits only, for some fixed  $d \geq 0$  [4]. Contrary to weight-reducing Hennie machines, the head is still allowed to visit a cell after the  $d$ -th visit, but cannot rewrite its content anymore. This allows to use super-linear time. Hence, *limited automata* (namely, *d-limited automata* for some  $d$ ) live midway between linear-space Turing machines and weight-reducing Hennie machines. For each  $d \geq 2$ , Hibbard proved that *d-limited automata* recognize exactly the class of *context-free languages*. He furthermore showed the existence of an infinite hierarchy of deterministic *d-limited automata*, whose first level (*i.e.*, corresponding to deterministic 2-limited automata) has been later proved to coincide with the class of *deterministic context-free languages* [10]. (See [7] and references therein for further connections between limited automata and context-free languages.)

Clearly, 0-limited automata are no more than two-way finite automata. Hence, they characterize the class of regular languages. Wagner and Wechsung extended this result to the case  $d = 1$ : 1-limited automata recognize exactly the class of regular languages [16]. From that point, the question of the cost of their simulation by classical finite automata has been studied by Pighizzini and Pisoni in [9], where a tight doubly-exponential simulation by deterministic one-way finite automata is proved. This cost reduces to a single exponential when starting from a deterministic 1-limited automaton. Also, an exponential lower bound, using a single-letter input alphabet, has been obtained in [11], for the simulation of deterministic 1-limited automata by nondeterministic two-way finite automata.

Like *d-limited automata*, 1-limited automata can operate in super-linear time (*cf.* Example 1). This contrasts with Hennie machines which operate in linear time by definition. The question we address in this paper is whether this ability of 1-limited automata with respect to Hennie machines yields a gap between the two models in terms of the size of their representations.

We show that, with a polynomial increase in size, each 1-limited automaton can be transformed into an halting linear-time 1-limited automaton, or alternatively, into a weight-reducing Hennie machine, while preserving determinism.

We also observe that the 1-limited automaton resulting from this construction can be easily transformed into an equivalent one whose behavior can be divided into two phases: (1) an initial phase consisting in a left-to-right one-way traversal of the input, during which each input symbol is nondeterministically overwritten; (2) a second phase consisting in a read-only two-way computation. Similar behaviors have been considered in the context of *regular transductions* (i.e., transductions computed by, for instance, two-way transducers), because of their correspondence with global existential quantification in *monadic second order logic*, see, e.g. [1]. Using terminology from [1], we define the model of *two-way automaton with common guess* in order to capture these particular behaviors of 1-limited automata. Formally, such machines are not 1-limited automata, but the composition of an initial *common guess* (i.e., a nondeterministic marking of the input symbols using symbols from a finite alphabet, computed, for instance, by a 1-state *letter-to-letter one-way transducer*) with a two-way automaton working on the enriched alphabet. Reformulating the above-mentioned result, each 1-limited automaton can be simulated by a two-way automaton with common guess of polynomial size. Furthermore, the underlying two-way finite automaton in the resulting device is deterministic, when starting from a deterministic 1-limited automaton. (Be aware that a deterministic two-way automaton with common guess, is not a deterministic device, since it initially performs a common guess which is nondeterministic by definition.) A direct consequence of this last result, is that *reversing* a 1-limited automaton, i.e., transforming it into another one recognizing the reverse of its accepted language, has polynomial cost. This fails in the deterministic case, for which we exhibit an exponential lower bound. As a consequence, we obtain exponential lower bounds for the simulation of



**Fig. 1.** Relationships between the main models studied in the paper. Here, *1t* and *wr* mean linear-time and weight-reducing, while *D1-LA* and *(D)HM* stand for deterministic 1-LA and (deterministic) Hennie machine, respectively. Deterministic and nondeterministic two-way automata with common guess are denoted by *2DFA+cg* and *2NFA+cg*. Dotted arrows indicate trivial connections while thick arrows indicate our results.

deterministic weight-reducing Hennie machines or deterministic two-way automata with common guess by deterministic 1-limited automata. The results are summarized in Fig. 1.

The paper is organized as follows. In Sect. 2 are gathered the main definitions and notations needed in the subsequent sections. The main ideas of the construction used for proving our results are detailed in Sect. 3, when considering the deterministic case. The results obtained therein are then extended to the nondeterministic case in Sect. 4.

## 2 Preliminaries

In this section we recall some basic definitions and notations useful in the paper. In particular, we assume the reader familiar with notions from formal languages and automata theory (see, *e.g.*, [5]). Given a set  $S$ ,  $\#S$  denotes its cardinality and  $2^S$  the family of all its subsets. Given an alphabet  $\Sigma$ , we denote by  $|w|$  the length of a string  $w \in \Sigma^*$ , by  $w^R$  the reversal of  $w$  and by  $\epsilon$  the empty string. For a language  $L \subseteq \Sigma^*$ ,  $L^R$  denotes the *reversal* of  $L$ , namely  $L^R = \{w^R \mid w \in L\}$ .

A *two-way nondeterministic finite automaton* (2NFA) is defined as a quintuple  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ , where  $Q$  is a finite *set of states*,  $\Sigma$  is a finite *input alphabet*,  $q_0 \in Q$  is the *initial state*,  $F \subseteq Q$  is a *set of final states*, and  $\delta : Q \times \Sigma_{\triangleright\triangleleft} \rightarrow 2^{Q \times \{-1, 0, +1\}}$  is a *nondeterministic transition function* where  $\Sigma_{\triangleright\triangleleft}$  denotes the set  $\Sigma \cup \{\triangleright, \triangleleft\}$  with the two special symbols  $\triangleright, \triangleleft \notin \Sigma$  respectively called the *left* and the *right endmarkers*. The input is written on the tape surrounded by the two endmarkers, the left endmarker being at the position zero. Hence, on input  $w$ , the right endmarker is at position  $|w| + 1$ . In one move,  $\mathcal{A}$  reads an input symbol, changes its state, and moves the input head one position backward, forward or keeps it in position depending on whether  $\delta$  returns  $-1$ ,  $+1$  or  $0$ , respectively. Furthermore, the head cannot violate the endmarkers, except at the end of computation, to accept the input, as now explained. The machine *accepts* the input, if there exists a computation path starting from the initial state  $q_0$  with the head on the first tape cell (*i.e.*, scanning the left endmarker) and ending in a final state  $q \in F$  after violating the right endmarker. The language accepted by  $\mathcal{A}$  is denoted by  $L(\mathcal{A})$ . A 2NFA  $\mathcal{A}$  is said to be *deterministic* (2DFA), whenever  $\#\delta(q, \sigma) \leq 1$ , for any  $q \in Q$  and  $\sigma \in \Sigma_{\triangleright\triangleleft}$ . It is called *one-way* if its head can never move backward, *i.e.*, if no transition returns  $-1$ . By 1NFAs and 1DFAs we denote one-way nondeterministic and deterministic finite automata, respectively.

A *1-limited automaton* (1-LA) is a tuple  $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ , where  $Q$ ,  $\Sigma$ ,  $q_0$  and  $F$  are defined as for 2NFAs,  $\Gamma$  is a finite *working alphabet* such that  $\Sigma \subset \Gamma$ ,  $\delta : Q \times \Gamma_{\triangleright\triangleleft} \rightarrow 2^{Q \times \Gamma_{\triangleright\triangleleft} \times \{-1, 0, +1\}}$  is the *nondeterministic transition function* where  $\Gamma_{\triangleright\triangleleft}$  denotes the set  $\Gamma \cup \{\triangleright, \triangleleft\}$  with  $\triangleright, \triangleleft \notin \Gamma$  the left and the right endmarkers as for 2NFAs. In one move, according to  $\delta$ ,  $\mathcal{A}$  reads a symbol from the tape, changes its state, replaces the symbol just read by a new symbol, and moves its head one position backward or forward or keeps it in place. However, replacing symbols is subject to some restrictions, which,

essentially, allow to modify the content of a cell during the first visit only. Technically, symbols from  $\Sigma$  shall be replaced by symbols from  $\Gamma \setminus \Sigma$ , while symbols from  $\Gamma_{\triangleright\triangleleft} \setminus \Sigma$  are never overwritten. In particular, at any time, both special symbols  $\triangleright$  and  $\triangleleft$  occur exactly once on the tape and exactly at the respective left and right boundaries. *Acceptance* for 1-LAS as well as *deterministic 1-LAS* are defined exactly as for 2NFAS, and the language accepted by a given 1-LA  $\mathcal{A}$  is denoted by  $L(\mathcal{A})$ .

Some 1-LAS have a particular behavior, which can be decomposed into two phases. In the first phase, they nondeterministically rewrite the content of the whole tape during a left-to-right traversal of the input. Then, in the second phase, they perform a two-way read-only computation over the overwritten tape. To formally define this kind of 1-LAS, we introduce the following model. A *2NFA (resp. 2DFA) with common guess* (*2NFA+cg, resp. 2DFA+cg*)<sup>2</sup> is a tuple  $\langle \mathcal{A}, \Sigma, \Delta \rangle$  where  $\Sigma$  and  $\Delta$  are two alphabets and  $\mathcal{A}$  is a 2NFA (resp. 2DFA) over the product alphabet  $\Sigma \times \Delta$ . The model is aimed to recognize languages from  $\Sigma^*$ . Its dynamics is defined as for two-way automata, but a nondeterministic pre-computation initially marks each input symbol with a symbol from  $\Delta$ . Hence, the read-only automaton  $\mathcal{A}$  has access to both the input symbol and the guessed additional information. The language accepted, denoted  $L(\langle \mathcal{A}, \Sigma, \Delta \rangle)$ , is defined as the projection, denoted  $\pi_1$ , of  $L(\mathcal{A})$  to the alphabet  $\Sigma$ , i.e.,  $L(\langle \mathcal{A}, \Sigma, \Delta \rangle) = \pi_1(L(\mathcal{A}))$ . In other words, a word is accepted by  $\langle \mathcal{A}, \Sigma, \Delta \rangle$  if for some guess, the enriched word is accepted by  $\mathcal{A}$ . We point out that, due to the common guess, 2DFA+cg's are nondeterministic devices.

For each of the above-defined models, a *configuration* is represented as a string  $z \cdot p \cdot z'$ , meaning that  $p$  is the current state,  $zz' \in \triangleright \Pi^* \triangleleft$  is the content of the tape (here  $\Pi$  denotes the alphabet  $\Sigma$ ,  $\Gamma$ , or  $\Sigma \times \Delta$  depending on the model under consideration) and the head is scanning the first symbol of  $z'$ . The transition relation between configurations is denoted by  $\vdash$ , and its reflexive-transitive closure by  $\vdash^*$ . Notice that, in case  $|z'| = 0$ , the machine has reached the end of the computation. We also represent *partial configurations* as  $u \cdot p \cdot v$ , where  $p$  is the current state and  $uv \in \{\epsilon, \triangleright\} \Pi^* \{\epsilon, \triangleleft\}$  is a factor of the tape content. The relations  $\vdash$  and  $\vdash^*$  naturally extend onto partial configurations.

For each model under consideration, we evaluate its size as the total number of symbols used to define it. Hence, the *size* of  $n$ -state 2NFAS, 1-LAS or 2NFA+cg's are given by some polynomial in the parameters  $n$ ,  $\#\Sigma$ , and possibly  $\#\Gamma$  or  $\#\Delta$ .

*Example 1.* We consider the language

$$L_n = \{x_0 x_1 \cdots x_k \mid k \in \mathbb{N}, x_i \in \{a, b\}^n, \#\{i > 0 \mid x_i = x_0\} \text{ is odd}\}.$$

A deterministic 1-LA  $\mathcal{A}_n$  may recognize  $L_n$  as follows. It first overwrites the factor  $x_0$ , replacing each input symbol with a marked copy. Then,  $\mathcal{A}_n$  repeats a subroutine which overwrites a factor  $x_i$  with some fixed symbol  $\sharp$ , while checking in the meantime whether  $x_i$  equals  $x_0$  or not. This can be achieved as follows. Before overwriting the  $j$ -th symbol of  $x_i$ , first,  $\mathcal{A}_n$ , with the help of a

<sup>2</sup> 2DFA+cg's also correspond to *synchronous two-way deterministic finite verifiers* [6].



counter modulo  $n$ , moves the head leftward to the position  $j$  of  $x_0$  and stores the unmarked scanned symbol  $\sigma$  in its finite control; second, it moves the head rightward until reaching the position  $j$  of  $x_i$ , namely, the leftmost position that has not been overwritten so far. At this point,  $\mathcal{A}_n$  compares the scanned symbol (*i.e.*, the  $j$ -th symbol of  $x_i$ ) with  $\sigma$  (*i.e.*, the  $j$ -th symbol of  $x_0$ ). By counting modulo 2 the number of factors equal to  $x_0$ , and finally checking that the input string has length multiple of  $n$ ,  $\mathcal{A}_n$  can decide the membership of the input to  $L_n$ .

It is possible to implement  $\mathcal{A}_n$  with a number of states linear in  $n$  and  $\#\Sigma + 1$  working symbols. Since for each position of a factor  $x_i$ ,  $i > 0$ , the head has to move back to the factor  $x_0$ , we observe that  $\mathcal{A}_n$  works in quadratic time in the length of the input string.

### 3 A Linear-Time Simulation for Deterministic 1-LAs

If a linear-space Turing machine can visit a tape cell only a constant number of times, it necessarily works in linear time. Conversely, Turing machines working in linear time (*i.e.*, Hennie machines), have been shown to visit each tape cell only a constant number of times during a computation [3]. This contrasts with the case of 1-LAs, which can use quadratic time, as shown in Example 1. However, our main contribution states that, with a polynomial increase in size of the model, we can recover the above property, and therefore obtain equivalent 1-LAs working in linear time.

**Theorem 1.** *For each deterministic 1-LA  $\mathcal{A}$ , there exists an equivalent deterministic 1-LA  $\mathcal{A}'$  satisfying:*

1.  $\mathcal{A}'$  has polynomial size with respect to  $\mathcal{A}$ ;
2. in every computation of  $\mathcal{A}'$ , each tape cell is visited a number of times which is bounded by some polynomial in the size of  $\mathcal{A}$ ;
3.  $\mathcal{A}'$  works in linear time: on every input string  $w$ , it halts within  $O(|w|)$  steps.

*Proof.* Clearly, Item 3 is implied by Item 2. It remains to prove that Item 2 can be achieved, while keeping a polynomial size of the device, namely Item 1. The key idea is to ensure that, in any computation, the simulating device works in a “virtual” window of fixed size, that is shifted along the configuration in a one-way manner. More precisely, in every computation and for each cell  $c$ , there exists a time  $t$  after which  $c$  is not visited anymore and furthermore, at this time, the rightmost cell that has been overwritten is at bounded distance to the right of  $c$ . This requires to detect local loops, that, thanks to this local window boundaries on space, can be done by using a polynomial number of states.

In [9], the authors presented a construction to simulate any deterministic 1-LA  $\mathcal{A}$  by a 1DFA  $\mathcal{A}''$ , using classic ideas from the simulation of 2DFAs by 1DFAs [13]. The main ingredient is to store in the finite control of  $\mathcal{A}''$ , a “transition table” describing the possible behaviors of  $\mathcal{A}$  that may occur to the left of the current head position. Since the part of the tape to the left of the current head position has necessarily already been visited, its “frozen” content

belongs to  $\triangleright(\Gamma \setminus \Sigma)^*$ . Hence, the above-mentioned behaviors to the left of the current head position, are read-only computations. To represent them, for each word  $z'X \in \triangleright(\Gamma \setminus \Sigma)^*$  with  $|X| = 1$ , a function  $\tau_{z'X} : Q \rightarrow Q$ , where  $Q$  denotes the set of states of  $\mathcal{A}$ , is considered. The function maps a state  $p$  to a state  $q$ , if, starting from the state  $p$  with the head scanning the last symbol of  $z'X$ ,  $\mathcal{A}$  eventually reaches the state  $q$  one cell to the right of  $z'X$ . Formally,

$$\tau_{z'X}(p) = q \quad \text{if} \quad z' \cdot p \cdot X \vdash^* z'X \cdot q.$$

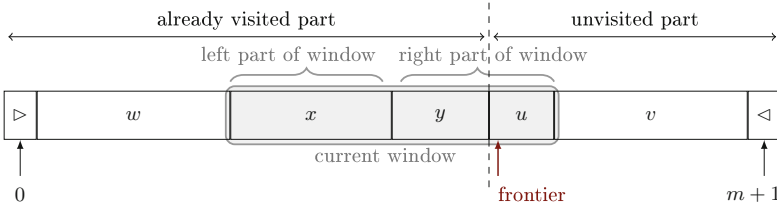
By determinism of  $\mathcal{A}$ ,  $\tau_{z'X}$  is a function which is partial in general.<sup>3</sup> We complete it by setting all undefined images to some fixed symbol  $\perp \notin Q$ . With the information of  $\tau_{z'X}$ ,  $\mathcal{A}''$  has no need to read the part of the tape containing  $z'X$ , that is, to move its head leftward. Moreover, as acceptance is done by violating the right endmarker, if the configuration  $z' \cdot p \cdot X$  occurs in some accepting computation, then  $\tau_{z'X}(p) \in Q$ . Hence,  $\mathcal{A}''$  cannot miss accepting computations. Finally, given a string  $x \in (\Gamma \setminus \Sigma)^*$ , we can construct  $\tau_{z'Xx}$  from  $\tau_{z'X}$  by scanning  $x$ .

In [9], the table of size  $(n+1)^n$  corresponding to the function  $\tau_{z'X}$  was stored in the finite control of the simulating 1DFA and it was updated at each step. This yielded an exponential number of states for the simulating device (that was shown to be necessary for the considered simulation). Here, as our simulating device  $\mathcal{A}'$  is a deterministic 1-LA, we take advantage of its ability to write on the tape, and we store the table onto the  $n$  cells following the last position of  $z'X$ . Thus, the  $i$ -th position to the right of the tape part containing  $z'X$  will contain the image of the  $i$ -th state of  $\mathcal{A}$  by  $\tau_{z'X}$ , as part of its written symbol. However, updating the table when moving to the right is done block by block rather than cell by cell, for a decomposition of the input into blocks of length  $\#Q$ . We consider the cell containing the left endmarker as a whole block, while the last block containing the right endmarker may be shorter than  $\#Q$ .

We now gather the two ideas presented above. Let  $Q = \{q_0, q_1, \dots, q_{n-1}\}$  be the set of states of  $\mathcal{A}$ ,  $\perp \notin Q$  be a fixed symbol and  $Q_\perp$  denote the set  $Q \cup \{\perp\}$ . At any time in a computation of  $\mathcal{A}$  we consider a “virtual window” of size  $2n$  which covers two successive blocks of length  $n = \#Q$ . The right block covered by the window contains the leftmost cell that has not been visited so far, to which we refer as *current frontier*. A typical situation is depicted in Fig. 2.

In order to simulate  $\mathcal{A}$ , the linear-time 1-LA overwrites each block with a word  $\tilde{x} \in ((\Gamma \setminus \Sigma) \times Q_\perp)^n$  whose projection to  $(\Gamma \setminus \Sigma)$  is the word  $x$  written by  $\mathcal{A}$  on the corresponding block, and the projection to  $Q_\perp$  is exactly the table  $\tau_z$ , where  $z$  is the content of the tape to the left of the block. (In Fig. 2,  $z = \triangleright w$  when considering the left block covered by the window, whose “frozen” content is  $x$ .) Roughly, when the window covers such a block as left part,  $\mathcal{A}'$  has to fill

<sup>3</sup>  $\tau_{z'X}(p)$  is undefined if one of the two following cases of the computation starting in  $z' \cdot p \cdot X$  occurs: either, after a finite number of steps, no successive transition is defined (incompleteness of  $\mathcal{A}$ ), or the computation eventually enters a deterministic loop (non-haltingness of  $\mathcal{A}$ ).



**Fig. 2.** Typical description of the window during a computation of  $\mathcal{A}$ :  $m$  denotes the length of the input word, the current frontier occurs in the right block as first position of  $u$ ,  $w \in ((\Gamma \setminus \Sigma)^n)^*$ ,  $x \in (\Gamma \setminus \Sigma)^n$ ,  $y \in (\Gamma \setminus \Sigma)^*$ ,  $u \in \Sigma^+$  with  $|yu| = n$ , and  $v \in \Sigma^*$ .

the next block, cell by cell, with  $\tau_{zx}$ . To this end, it has read-only access to the left block, containing all the required information, namely  $\tau_z$  and  $x$ . In parallel,  $\mathcal{A}'$  should also recover the simulated computation of  $\mathcal{A}$ . As soon as the right block is completely filled, the window is shifted to the right, in such a way that it covers the block just treated (as left part) and its successor (as right part).

We now describe the formal implementation of the above-explained procedure. By using a state component of size  $2n$ , named *relative position*,  $\mathcal{A}'$  can store the exact position of its head relative to the current window. We represent it as a pair  $(i, s)$ , where  $i \in \{0, \dots, n - 1\}$  is the position in the scanned block of length  $n$  and  $s \in \{L, R\}$  is equal to L (resp. R) if the head is scanning a position in the left (resp. right) block of the window. We suppose that the component is updated at each head move. Using this component,  $\mathcal{A}'$  can avoid moving to the left of the current window. More precisely, from a relative position  $(0, L)$  (i.e., the leftmost position covered by the window), in order to simulate a backward move of  $\mathcal{A}$  from  $p$  to  $q$ ,  $\mathcal{A}'$  enters a special mode to determine the state  $\tau_z(q)$  (if it exists), where  $z$  is the content of the tape to the left of the window. Hence, it simulates not only the backward step from  $p$  to  $q$ , but also the complete computation segment to the left of the window from state  $q$ , namely, it simulates  $z'X \cdot p \vdash z' \cdot q \cdot X \vdash^* z'X \cdot \tau_z(q)$ , where  $z = z'X$ . This special mode, called *readFromTable*, which starts and ends in relative position  $(0, L)$ , consists in a simple read of the table  $\tau_z$  that has been written on the left block covered by the current window (in the factor  $\tilde{x}$  corresponding to the factor  $x$  in Fig. 2).

In addition to the relative position,  $\mathcal{A}'$  stores in its finite control the relative position of the current frontier, to which we refer as *relative frontier*. Since this position always occurs in the right block of the window, it is enough to represent it as an index  $\rho \in \{0, \dots, n - 1\}$ . Much like the relative position component, we suppose that it is updated each time a new cell is visited. Observe that such updates are increments modulo  $n$ . As explained below, incrementing  $\rho = n - 1$  means shifting the window  $n$  cells to the right. Using both  $\rho$  and  $(i, s)$ ,  $\mathcal{A}'$  can ensure that entering a cell for the first time, may be done only once all necessary information (that is required to determine the symbol to write on the cell at its first visit) has been gathered. This information corresponds to a pair  $(q, \tau_{zx}(q_\rho))$ , where  $q$  is the state entered by  $\mathcal{A}$  when visiting for the first time the frontier

cell in the simulated computation, and  $zx$  is the content of the tape to the left of the right block of the current window.

We now describe the subroutine `simulateLeft` that is used for recovering this information. The procedure takes a state  $p$  as argument and starts from and ends in some relative position  $(i, s)$  which indicates either one cell to the left of the right block, or one cell to the left of the frontier. Notice that the two positions coincide when  $\rho = 0$ . Hence,  $(i, s)$  belongs to  $\{(n - 1, L), (\rho - 1, R)\}$ . Denoting  $\gamma \in (\Gamma \setminus \Sigma)$  the symbol contained in this cell, the procedure returns  $q \in Q_\perp$  such that  $\tau_{zx'\gamma}(p) = q$ , where  $zx'$  is the content of the tape to the left of the relative position  $(i, s)$ . (In Fig. 2,  $z = \triangleright w$  and  $x'\gamma$  is equal to  $x$  or to  $xy$ , respectively if  $(i, s)$  is equal to  $(n - 1, L)$  or to  $(\rho - 1, R)$ .) During the computation, `simulateLeft` has access to the content of the window up to position  $(i, s)$ . It basically performs a direct simulation of  $\mathcal{A}$  on the corresponding part of the tape, which uses the procedure `readFromTable` in order to simulate (in one step) computations that occur to the left of the window, as explained above. Moreover, if  $zx' \cdot p \cdot \gamma \vdash^* zx'\gamma \cdot q$  in  $\mathcal{A}$ , namely if  $q = \tau_{zx'\gamma}(p) \neq \perp$ , then it halts in the relative position  $(i, s)$ , before the last step of the simulated computation. At this point, it is possible to determine the return value  $q$ . The direct simulation might fail for two reasons: (1) the simulation halts on some previous position (incompleteness of  $\mathcal{A}$ ); (2) the simulation enters a deterministic loop and never exits (non-haltingness of  $\mathcal{A}$ ). For the first case, it is sufficient to enter a sub-mode which moves the head to the right until reaching the relative position  $(i, s)$  and returns  $\perp$ . The second case requires to detect loops. Since the simulating computation takes place in a read-only window of size at most  $2n$ , any loop-free computation has length bounded by some polynomial of degree 2. Hence, by using a clock of size  $O(n^2)$ , we can enforce the procedure to halt. Only runs that halted before this time limit may return a state value, while “killed” runs will return  $\perp$ .<sup>4</sup>

Hence, before visiting for the first time the cell at relative position  $(\rho, R)$ :

- first**,  $\mathcal{A}'$  calls `simulateLeft`( $p$ ) from the position  $(\rho - 1, R)$ , and saves its return value  $q$  in its finite control, where  $p$  is the state that was entered by  $\mathcal{A}$  when visiting for the first time the cell at relative position  $(\rho - 1, R)$  in the simulated computation;
- then**,  $\mathcal{A}'$  calls `simulateLeft`( $q_\rho$ ) from the position  $(n - 1, L)$ , and stores the returned value  $r$  in its finite control.

Once  $\mathcal{A}'$  has gathered the pair of states  $(q, \tau_{zx}(q_\rho)) = (q, r)$ , it moves to the cell at relative position  $(\rho, R)$ , and reads the input symbol  $\sigma \in \Sigma \cup \{\triangleleft\}$ . If  $\sigma = \triangleleft$ ,  $\mathcal{A}'$  calls `simulateLeft`( $q$ ) and accepts, after violating the endmarker, if the return value is a final state of  $\mathcal{A}$  (it rejects otherwise). If  $\sigma \neq \triangleleft$ ,  $\mathcal{A}'$  writes the symbol  $(r, \gamma)$  where  $\gamma \in (\Gamma \setminus \Sigma)$  is the symbol returned by  $\delta(q, \sigma)$ , and repeats

<sup>4</sup> We could do a finer construction, based on Sipser’s backward construction [14], which has linear cost (without counting the relative position and relative frontier components) instead of the expensive  $O(n^3)$  cost of the clocked simulation presented here. For an adaptation to finite automata, see [2].

the procedure with the updated relative frontier. In the case  $\rho = n - 1$ , the window is shifted to the right, in such a way that the head is positioned on the rightmost cell of its left block. This is formally done by setting the relative position to  $(n - 1, L)$  and the relative frontier to  $(0, R)$ .

Let us describe the initial configuration of  $\mathcal{A}'$ . At the beginning of the computation, the head is scanning the left endmarker, which is considered as the left block of the current window. Hence, the initial relative position and relative frontier are set to  $(n - 1, L)$  and  $(0, R)$ , respectively. Since the head of  $\mathcal{A}$  cannot move backward from the left endmarker, the procedure `simulateLeft` never calls the subroutine `readFromTable`, as long as the window is in its initial place.

We have shown how  $\mathcal{A}'$  simulates  $\mathcal{A}$  in an halting manner, by shifting a virtual window to the right along its computation, and by restricting local head moves to the current window. Moreover,  $\mathcal{A}'$  only uses a polynomial number of states in  $n$ , and working alphabet  $(\Gamma \setminus \Sigma) \times Q_{\perp} \cup \Sigma$ . We now evaluate the time used by  $\mathcal{A}'$ . Let fix a cell  $c$ . As  $\mathcal{A}'$  is loop-free, each time the head visits  $c$  it must have a different state or a different tape content. A tape modification between two visits of  $c$  is restricted to cells from the right block of the current window containing  $c$ . The number of successive tape modifications in a window is linear in  $n$  (after  $n$  overwritings, the window is shifted), and  $c$  may occur in two successive windows. Thus, the number of visits to the cell  $c$  is bounded by some polynomial in  $n$ . The number of visits to each cell is hence bounded by a polynomial in  $n$ . As a consequence,  $\mathcal{A}'$  operates in linear time with respect to the input length.  $\square$

Linear-time 1-LAS are particular cases of Hennie machines, hence, it follows from the above result that any 1-LA can be transformed into a Hennie machine of size polynomial in the size of the 1-LA. Using Item 2 we can actually obtain the stronger result that the obtained 1-LA can be transformed into a *weight-reducing Hennie machine*. Informally, weight-reducing Hennie machines are Hennie machines in which each overwriting is decreasing with respect to some fixed order on the working alphabet. As a consequence, after overwriting a cell with a minimal symbol, such a machine cannot visit the cell again. See [12] for formal definition and study of the model. It is also possible to modify the construction of Theorem 1 in order to obtain an equivalent 2DFA+cg.

**Corollary 1.** *For each deterministic 1-LA, there exists an equivalent deterministic weight-reducing Hennie machine or halting 2DFA+cg of size polynomial in the size of the 1-LA.*

Concerning the converse simulation, using the language  $L_n$  from Example 1, we can prove an exponential gap in the deterministic case.

**Theorem 2.** *Let  $L_n$  be the language of Example 1. Hence*

$$L_n^R = \{x_k x_{k-1} \cdots x_0 \mid k > 0, x_i \in \{a, b\}^n, \#\{i > 0 \mid x_i = x_0\} \text{ is odd}\}.$$

Then,

1.  $L_n^R$  is accepted by a 2DFA+cg, a linear-time nondeterministic 1-LA, or a deterministic weight-reducing Hennie machine of size polynomial in  $n$ ;
2. any 1DFA recognizing  $L_n^R$  requires  $2^{2^n}$  states;
3. any deterministic 1-LA recognizing  $L_n^R$  requires  $O(2^n)$  states.

*Proof (outline).* Example 1 describes a deterministic 1-LA recognizing  $L_n$ , whose size is linear in  $n$ . By applying Corollary 1, we respectively obtain equivalent weight-reducing Hennie machine and 2DFA+cg of polynomial size. Both models can be transformed with a constant increase in size, in order to accept the reverse of the language, thus proving Item 1. Using a distinguishability argument, we can prove Item 2. Finally, Item 3 can be deduced from this previous point and the exponential blowup from deterministic 1-LA to 1DFA given in [9].  $\square$

## 4 A Linear-Time Simulation for Nondeterministic 1-LAS

The techniques presented in the proof of Theorem 1 can be used for the nondeterministic case. However, this time, the table  $\tau_z$  does no longer represent a partial function, but a binary relation on  $Q$ , whose size is thus  $2^{n^2}$ . To handle this size increase, we store the table on the  $n^2$  cells following the last cell of the tape part containing  $z$  (one bit by cell). The “virtual” window described in the previous section is hence extended to the size  $2n^2$ .

**Theorem 3.** *For each nondeterministic 1-LA  $\mathcal{A}$ , there exists an equivalent nondeterministic 1-LA  $\mathcal{A}'$ , satisfying:*

1.  $\mathcal{A}'$  has polynomial size with respect to  $\mathcal{A}$ ;
2. in every computation of  $\mathcal{A}'$ , each tape cell is visited a number of times which is bounded by some polynomial in the size of  $\mathcal{A}$ ;
3.  $\mathcal{A}'$  works in linear time: on every input string  $w$ , every branch of the computation of  $\mathcal{A}'$  halts within  $O(|w|)$  steps.

*Proof.* Again, Item 2 implies Item 3. The proof that Item 2 can be achieved while preserving a polynomial size, namely Item 1, is analogous to those given in the deterministic case (Theorem 1). We emphasize the main differences below.

First, for  $z \in \triangleright(\Gamma \setminus \Sigma)^*$ , a pair  $(p, q)$  belongs to the **relation**  $\tau_z$ , if and only if  $z' \cdot p \cdot X \vDash z'X \cdot q$  where  $z'X = z$  with  $|X| = 1$ . Hence, the virtual window is extended to the size  $2n^2$ , in order to store a table of size  $n^2$  (one bit by cell) representing a relation  $\tau_z \subseteq Q \times Q$ . The working alphabet of  $\mathcal{A}'$  is therefore set to  $(\Gamma \setminus \Sigma) \times \{0, 1\} \cup \Sigma$  where  $\Gamma$  and  $\Sigma$  are the working and input alphabets of  $\mathcal{A}$ .

Second, the `readFromTable` subroutine, which takes a state  $p$  as argument, returns a nondeterministically chosen state  $q$  such that the pair  $(p, q)$  has image 1 in the table stored in the left block of the window.<sup>5</sup> Also the subroutine `simulateLeft` is nondeterministic, as it performs a direct simulation of  $\mathcal{A}$ , and

<sup>5</sup> We implicitly fix a bijection from  $\{0, \dots, n^2 - 1\}$  to  $Q^2$ .

possibly calls the subroutine `readFromTable`. As a consequence, the table stored on the left block of the current window does not encode the complete relation  $\tau_z$  but a subset of it. The reason behind this loss of information, is that checking the nonexistence of computations from  $z' \cdot p \cdot X$  to  $z'X \cdot q$  requires a universal quantification over computations, for which nondeterminism (which corresponds to existential quantification over computations) is unsuitable. Nevertheless, the relation encoded is a subset of  $\tau_z$  and when making perfect guesses, it is equal to  $\tau_z$ . Thus, no accepting computation is missed.

We point out that the simulating automaton can still use a clock of polynomial size (but this time, Sipser's construction cannot apply) to limit the time of direct simulations of  $\mathcal{A}$  that necessarily occur in the space bounded locally by the virtual window. Doing so, the resulting 1-LA is halting, has polynomial size with respect to  $\mathcal{A}$ , and works in linear time.  $\square$

By analogy to Corollary 1, it follows from Theorem 3:

**Corollary 2.** *For each 1-LA, there exists an equivalent weight-reducing Hennie machine or halting 2NFA+cg of polynomial size.*

This last result is of particular interest. Indeed, 2NFA+cg's are particular cases of 1-LAS. (It is not the case for 2DFA+cg's with respect to deterministic 1-LAS.) Hence, Corollary 2 gives a kind of normal form for nondeterministic 1-LAS.

**Acknowledgement.** We are very indebted to Giovanni Pighizzini for suggesting the problem and for many stimulating conversations.

## References

1. Bojańczyk, M., Daviaud, L., Guillon, B., Penelle, V.: Which classes of origin graphs are generated by transducers. In: ICALP 2017. LIPIcs, vol. 80, pp. 114:1–114:13 (2017)
2. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. *Inf. Comput.* **205**(8), 1173–1187 (2007)
3. Hennie, F.C.: One-tape, off-line Turing machine computations. *Inf. Comput.* **8**(6), 553–578 (1965)
4. Hibbard, T.N.: A generalization of context-free determinism. *Inf. Comput.* **11**(1/2), 196–238 (1967)
5. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Boston (1979)
6. Kapoutsis, C.A.: Predicate characterizations in the polynomial-size hierarchy. In: Beckmann, A., Csuhaj-Varjú, E., Meer, K. (eds.) CiE 2014. LNCS, vol. 8493, pp. 234–244. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08019-2\\_24](https://doi.org/10.1007/978-3-319-08019-2_24)
7. Kutrib, M., Pighizzini, G., Wendlandt, M.: Descriptive complexity of limited automata. *Inf. Comput.* **259**(2), 259–276 (2018)
8. Pighizzini, G.: Nondeterministic one-tape off-line Turing machines. *J. Autom. Lang. Comb.* **14**(1), 107–124 (2009)
9. Pighizzini, G., Pisoni, A.: Limited automata and regular languages. *Int. J. Found. Comput. Sci.* **25**(07), 897–916 (2014)

10. Pighizzini, G., Pisoni, A.: Limited automata and context-free languages. *Fundamenta Informaticae* **136**(1–2), 157–176 (2015)
11. Pighizzini, G., Prigioniero, L.: Limited automata and unary languages. In: Charlier, É., Leroy, J., Rigo, M. (eds.) *DLT 2017*. LNCS, vol. 10396, pp. 308–319. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-62809-7\\_23](https://doi.org/10.1007/978-3-319-62809-7_23)
12. Průša, D.: Weight-reducing hennie machines and their descriptive complexity. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) *LATA 2014*. LNCS, vol. 8370, pp. 553–564. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-04921-2\\_45](https://doi.org/10.1007/978-3-319-04921-2_45)
13. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.* **3**(2), 198–200 (1959)
14. Sipser, M.: Halting space-bounded computations. *Theor. Comput. Sci.* **10**(3), 335–338 (1980)
15. Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time Turing machines. *Theor. Comput. Sci.* **411**(1), 22–43 (2010)
16. Wagner, K.W., Wechsung, G.: *Computational Complexity*. D. Reidel Publishing Company, Dordrecht (1986)





# Cover Complexity of Finite Languages

Stefan Hetzl and Simon Wolfsteiner<sup>(✉)</sup>

Institute of Discrete Mathematics and Geometry, TU Wien,  
Wiedner Hauptstraße 8–10, 1040 Wien, Austria  
{stefan.hetzl,simon.wolfsteiner}@tuwien.ac.at

**Abstract.** We consider the notion of cover complexity of finite languages on three different levels of abstraction. For arbitrary cover complexity measures, we give a characterisation of the situations in which they collapse to a bounded complexity measure. Moreover, we show for a restricted class of context-free grammars that its grammatical cover complexity measure w.r.t. a finite language  $L$  is unbounded and that the cover complexity of  $L$  can be computed from the exact complexities of a finite number of covers  $L' \supseteq L$ . We also investigate upper and lower bounds on the grammatical cover complexity of the language operations intersection, union, and concatenation on finite languages for several different types of context-free grammars.

## 1 Introduction

The grammatical complexity of a formal language in the classical sense is the complexity of a minimal grammar generating this language. Depending on the type of grammar and the notion of complexity, one obtains a variety of different grammatical complexity measures. The study of the grammatical complexity of context-free languages can be traced back to [12], where, among other things, it was shown that context-free definability with  $n$  nonterminals forms a strict hierarchy. This line of research has been continued in [7, 13–15], where, among others, the number of productions of a grammar has been considered as complexity measure. In [4], a theory of the grammatical complexity of finite languages in terms of production complexity was initiated by giving a relative succinctness classification for various kinds of context-free grammars. Investigations along these lines have been continued in, e.g., [1–3, 8, 9, 21].

We are interested in the cover complexity of a finite language  $L$ , i.e., the minimal number of productions of a grammar  $G$  such that  $L(G)$  is finite and  $L(G) \supseteq L$ . Note that this condition is similar to (but different from) the one imposed on cover automata [5, 6]: there, an automaton  $A$  is sought such that  $L(A) \supseteq L$ , but in addition it is required that  $L(A) \setminus L$  consists only of

---

Supported by the Vienna Science Fund (WWTF) project VRG12-004 and the Austrian Science Fund (FWF) project P25160.

words longer than any word in  $L$ . Our interest in this problem is primarily motivated by applications in proof theory. As shown in [16], there is an intimate relationship between a certain class of formal proofs (those with  $\Pi_1$ -cuts) in first-order predicate logic and a certain class of grammars (totally rigid acyclic tree grammars). In particular, the number of production rules in the grammar characterises the number of certain inference rules in the proof. This relationship has been exploited for a number of results in proof theory and automated deduction [17–19]. In particular, [10, 11] shows a non-trivial lower bound on the complexity of cut-introduction. The interest in such a result is partially motivated by the experience that the length of proofs with cuts is notoriously difficult to control (for propositional logic this is considered the central open problem in proof complexity [20]). The combinatorial center of this result is the construction of a sequence of finite word languages which are incompressible in the sense of the cover formulation of grammatical complexity.

In this paper, we investigate the notion of cover complexity of finite languages on three different levels. First, in Sect. 3, we consider the cover complexity from an abstract point of view for arbitrary complexity measures and we characterise the situations in which it collapses to a bounded measure. Secondly, in Sect. 4, we consider the cover complexity of a finite language as the minimal number of productions a context-free grammar needs to cover the language with a finite language. In particular, we show that a cover complexity measure is unbounded if it is induced by a class of context-free grammars with a bounded number of nonterminals on the right-hand side of their productions. Moreover, unboundedness allows to reduce the cover complexity of a finite language  $L$  to the minimum of the exact complexities over a finite number of supersets  $L'$  of  $L$ . Thirdly, and yet more specifically, in Sect. 5, we investigate the grammatical cover complexity of the language operations intersection, union, and concatenation on finite languages for context-free, (strict) linear, and (strict) regular grammars.

## 2 Cover Complexity

In this section, we introduce the basic definitions of the notion of cover complexity from both an abstract and grammatical point of view. Moreover, in order to fix notation and terminology, we also introduce the basic notions of formal language theory.

For a set  $A$ , we write  $\mathcal{P}_{\text{fin}}(A)$  for the set of finite subsets of  $A$ . Let  $\Sigma$  be an alphabet, then a function  $\mu : \mathcal{P}_{\text{fin}}(\Sigma^*) \rightarrow \mathbb{N}$  is called  $\Sigma$ -complexity measure. If the alphabet is irrelevant or clear from the context, we will just speak about a complexity measure. Let  $\mu$  be a  $\Sigma$ -complexity measure. The *cover complexity measure induced by  $\mu$*  is the  $\Sigma$ -complexity measure  $\mu\mathcal{C}$  defined as

$$\mu\mathcal{C}(L) = \min\{\mu(L') \mid L \subseteq L' \in \mathcal{P}_{\text{fin}}(\Sigma^*)\}.$$

Note that the minimum is well-defined even though there are infinitely many  $L' \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  with  $L \subseteq L'$ , since  $\mu$  maps to the natural numbers. We have  $\mu\mathcal{C}(L) \leq \mu(L)$ , for all  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ . Moreover, for every  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ , there is an  $L' \supseteq L$

such that  $\mu c(L) = \mu(L')$ . A  $\Sigma$ -complexity measure  $\mu$  is called *bounded* if there is a  $k \in \mathbb{N}$  such that  $\mu(L) \leq k$ , for all  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ , and unbounded otherwise.

A *context-free (CF)* grammar is a quadruple  $G = (N, \Sigma, P, S)$ , where  $N$  and  $\Sigma$  are disjoint finite sets of *nonterminals* and *terminals*, respectively,  $S \in N$  is the *start symbol*, and  $P$  is a finite set of *productions* of the form  $A \rightarrow \alpha$ , where  $A \in N$  and  $\alpha \in (N \cup \Sigma)^*$ . Let  $A$  be a nonterminal, then a production with  $A$  on its left-hand side is called *A-production*. The set of all words of length at most  $k$ , for  $k \geq 0$ , over  $\Sigma$  is denoted by  $\Sigma^{\leq k}$ . We also consider further restrictions of context-free grammars: a context-free grammar is called *linear context-free (LIN)* if all productions in  $G$  are of the form  $A \rightarrow \alpha$ , where  $\alpha \in \Sigma^*(N \cup \{\varepsilon\})\Sigma^*$ ; a context-free grammar is called *right-linear* or *regular (REG)* if all productions in  $G$  are of the form  $A \rightarrow \alpha$ , where  $\alpha \in \Sigma^*(N \cup \{\varepsilon\})$ . Moreover, a context-free grammar is called *strict linear (SLIN)* if all productions are of the form  $A \rightarrow aBb$  or  $A \rightarrow c$ , where  $B \in N$  and  $a, b, c \in \Sigma^{\leq 1}$ . Similarly, a context-free grammar is called *strict regular (SREG)* if all productions are of the form  $A \rightarrow aB$  or  $A \rightarrow b$ , where  $B \in N$  and  $a, b \in \Sigma^{\leq 1}$ . We will also write *SREG*, *REG*, ... for the set of strict regular, regular, ... grammars and define  $\Gamma = \{SREG, REG, SLIN, LIN, CF\}$ . As usual, the *derivation relation of G* is denoted by  $\Rightarrow_G$  and the reflexive and transitive closure of  $\Rightarrow_G$  is written as  $\Rightarrow_G^*$ . If the grammar is clear from the context, we will often omit the subscript  $G$ . The *language of a grammar G* is defined as  $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$ . We say that a context-free grammar  $G$  covers a language  $L$  if  $L(G) \supseteq L$ . The *size* of a context-free grammar  $G = (N, \Sigma, P, S)$  is defined as  $|G| = |P|$ . Let  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  and  $X \in \Gamma$ . Then the  $X$ -complexity of  $L$  is

$$\text{Xc}(L) = \min\{|G| \mid G \in X, L = L(G)\}.$$

Clearly,  $\text{Xc}$  is a complexity measure and induces the cover complexity measure

$$\text{Xcc}(L) = \min\{\text{Xc}(L') \mid L \subseteq L' \in \mathcal{P}_{\text{fin}}(\Sigma^*)\}.$$

Consequently, we say that  $G$  is a *minimal X-grammar* covering (or generating, respectively) the finite language  $L$  if  $L(G)$  is finite,  $L \subseteq L(G)$  (or  $L = L(G)$ , respectively), and  $|G| = \text{Xcc}(L)$  (or  $|G| = \text{Xc}(L)$ , respectively). Note that, in general, there may be more than one minimal  $X$ -grammar for a given language  $L$ . The following result shows the existence of regular-incompressible sequences of finite languages and has been proved in [10, 11].

**Theorem 1.** *For all  $n \geq 1$ , there is a language  $L_n$  with  $|L_n| = n = \text{REGcc}(L_n)$ .*

On the other hand, for every finite language  $L$ , there is a trivial context-free grammar covering  $L$  with a constant number of productions:

**Theorem 2.** *Let  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ , then  $\text{CFcc}(L) \leq |\Sigma| + 2$ .*

*Proof.* Let  $\Sigma = \{a_1, a_2, \dots, a_n\}$ ,  $l = \max\{|w| \mid w \in L\}$ , and consider the grammar  $G$  consisting of the productions  $S \rightarrow A^l, A \rightarrow a_1 \mid a_2 \mid \dots \mid a_n \mid \varepsilon$ . Then  $L(G) = \Sigma^{\leq l} \supseteq L$ .  $\square$

### 3 Unboundedness of Cover Complexity Measures

Motivated by the above Theorems 1 and 2, in this section, we will characterise the situations in which a cover complexity measure collapses to a bounded complexity measure. Before we can give this characterisation, we need some auxiliary results on “almost inverting” functions from  $\mathbb{N}$  to  $\mathbb{N}$ . These will be provided in Lemmas 1 and 2. A function  $f : \mathbb{N} \rightarrow \mathbb{N}$  is called *bounded* if there is a  $k \in \mathbb{N}$  such that  $f(n) \leq k$ , for all  $n \in \mathbb{N}$ , and *unbounded* otherwise. The function  $f$  is called *monotone* if  $n \leq m$  implies  $f(n) \leq f(m)$ .

**Lemma 1.** *Let  $f : \mathbb{N} \rightarrow \mathbb{N}$  be monotone and unbounded, define  $g : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto \min\{i \in \mathbb{N} \mid n \leq f(i)\}$ , then  $g$  is well-defined, monotone, unbounded, and, for all  $x, y \in \mathbb{N}$ :  $g(x) \leq y$  iff  $x \leq f(y)$ .*

**Lemma 2.** *Let  $g : \mathbb{N} \rightarrow \mathbb{N}$  be monotone and unbounded, let  $f : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto \max\{i \in \mathbb{N} \mid g(i) \leq n\}$ . Then  $f$  is well-defined, monotone, unbounded, and, for all  $x, y \in \mathbb{N}$ :  $g(x) \leq y$  iff  $x \leq f(y)$ .*

A complexity measure  $\rho : \mathcal{P}_{\text{fin}}(\Sigma^*) \rightarrow \mathbb{N}$  is called *reference complexity measure* if  $\rho$  is unbounded and  $L_1 \subseteq L_2$  implies  $\rho(L_1) \leq \rho(L_2)$ . For reference complexity measures, what we have in mind are, e.g., the number of words  $|L|$  in a language or their cumulated lengths  $\|L\| = \sum_{w \in L} |w|$ . Let  $\mu$  be a complexity measure, then a reference complexity measure  $\rho$  is called *reference complexity measure for  $\mu$*  if  $\mu(L) \leq \rho(L)$ , for all finite languages  $L$ . Typical examples for the above definition include:  $\mu = \text{REGc}, \text{CFc}, \dots$  and  $\rho(L) = |L|$ , or  $\mu$  is the minimal size, that is, symbolic complexity of a regular, context-free, ... grammar and  $\rho(L) = \|L\|$ . The following theorem provides a characterisation of the unboundedness of a cover complexity measure.

**Theorem 3.** *Let  $\mu$  be an unbounded  $\Sigma$ -complexity measure and  $\rho$  be a reference complexity measure for  $\mu$ , then the following are equivalent:*

1.  $\mu$  is unbounded
2. there is a monotone and unbounded function  $f : \mathbb{N} \rightarrow \mathbb{N}$  s.t.  $\rho(L) \leq f(\mu(L))$ , for all  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ .
3. there is a monotone and unbounded function  $g : \mathbb{N} \rightarrow \mathbb{N}$  s.t.  $g(\rho(L)) \leq \mu(L)$ , for all  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ .

*Proof.* 2.  $\Rightarrow$  3. has been shown in Lemma 1, and 3.  $\Rightarrow$  2. in Lemma 2.

For 3.  $\Rightarrow$  1., let  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ , then there is some  $L' \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  s.t.  $L \subseteq L'$  and  $\mu(L) = \mu(L')$ . Therefore,  $\mu(L) = \mu(L') \geq^3. g(\rho(L')) \geq^{\text{mon.}} g(\rho(L))$ , which shows unboundedness of  $\mu$  based on the unboundedness of  $g$  and  $\rho$ .

For showing 1.  $\Rightarrow$  3., we prove the contrapositive. Assume that every  $g : \mathbb{N} \rightarrow \mathbb{N}$  s.t.  $g(\rho(L)) \leq \mu(L)$ , for all  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ , is bounded or not monotone. Consider  $h : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto \min\{\mu(L) \mid \rho(L) \geq n, L \in \mathcal{P}_{\text{fin}}(\Sigma^*)\}$  and note that, due to the unboundedness of  $\rho$ ,  $h$  is well-defined. Moreover,  $h(\rho(L)) \leq \mu(L)$ . For monotonicity, let  $n \leq m$ . Then we have  $\{L \in \mathcal{P}_{\text{fin}}(\Sigma^*) \mid \rho(L) \geq m \geq n\} \subseteq \{L \in \mathcal{P}_{\text{fin}}(\Sigma^*) \mid \rho(L) \geq n\}$ . Therefore,  $h(n) = \min\{\mu(L) \mid \rho(L) \geq n\} \leq$

$\min\{\mu(L) \mid \rho(L) \geq m\} = h(m)$ . So  $h$  is bounded, i.e., there is a  $k \in \mathbb{N}$  and  $(L_n)_{n \in \mathbb{N}}$  such that  $n \mapsto \rho(L_n)$  is unbounded, but  $\mu(L_n) \leq k$ , for all  $n \in \mathbb{N}$ . Since  $\mu c(L_n) \leq \mu(L_n) \leq k$ ,  $\mu c$  is bounded too.  $\square$

**Theorem 4.** *Let  $\mu$  be a complexity measure and  $\rho$  be a reference complexity measure for  $\mu$ . Then, for every finite language  $L$ , there is some  $b \in \mathbb{N}$  such that*

$$\mu c(L) = \min\{\mu(L') \mid L \subseteq L' \in \mathcal{P}_{\text{fin}}(\Sigma^*) \text{ and } \rho(L') \leq b\}.$$

*Proof.* If  $\mu c$  is bounded by  $k$ , let  $b = k$ . If  $\mu c$  is unbounded, then, by Theorem 3, there is an unbounded and monotone function  $g : \mathbb{N} \rightarrow \mathbb{N}$  s.t.  $g(\rho(K)) \leq \mu(K)$ , for all finite languages  $K$ , and, by Lemma 2, there is an unbounded and monotone function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that  $g(x) > y$  iff  $x > f(y)$ , for all  $x, y \in \mathbb{N}$ . Let  $b = f(\rho(L))$  and  $L'' \supseteq L$  with  $\rho(L'') > f(\rho(L))$ , then  $g(\rho(L'')) > \rho(L)$ , and, since we have  $\mu(L'') \geq g(\rho(L''))$ , we obtain  $\mu(L'') > \rho(L)$ . Moreover, since  $\rho(L) \geq \mu(L) \geq \mu c(L)$ , we have  $\mu(L'') > \mu c(L)$ .  $\square$

The above theorem expresses  $\mu c$  in terms of  $\mu$  and  $\rho$ . Depending on  $\rho$ , the set of covers  $L'$  of  $L$  that is used to determine  $\mu c(L)$  may or may not be a finite set. We will analyse the reduction of  $\mu c(L)$  to the value of  $\mu(\cdot)$  on a finite set more thoroughly in the next section.

## 4 Computing Cover Complexity from Exact Complexity

After dealing with complexity measures in an abstract sense in the previous section, we now come back to applications in the realm of context-free grammars. In particular, we now focus on the number of productions in various types of grammars. Hence, we will fix  $\rho(L) = |L|$  as reference complexity measure.

The subsequent lemma was already shown in [4] and implies that  $\text{Xcc}$ , for  $X \in \{\text{SREG}, \text{REG}, \text{SLIN}, \text{LIN}\}$ , is an unbounded complexity measure.

**Lemma 3.** *Let  $G$  be a linear grammar with  $n$  productions generating a finite language, then  $|\text{L}(G)| \leq 2^{n-1}$ .*

**Corollary 1.** *The measures  $\text{SREGcc}$ ,  $\text{REGcc}$ ,  $\text{SLINcc}$ , and  $\text{LINcc}$  are unbounded.*

*Proof.* Define the function  $f : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto 2^{n-1}$ . Clearly,  $f$  is both monotone and unbounded. By Lemma 3, for all finite languages  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ , we have  $\rho(L) = |L| \leq 2^{\text{LINc}(L)-1} = f(\text{LINc}(L))$ . Hence, by Theorem 3,  $\text{LINcc}$  is unbounded. The unboundedness of  $\text{SREGcc}$ ,  $\text{REGcc}$ , and  $\text{SLINcc}$  follows from the fact that  $\text{LINcc}(L) \leq \text{SLINcc}(L) \leq \text{SREGcc}(L)$  and  $\text{LINcc}(L) \leq \text{REGcc}(L)$ , for all finite languages  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ .  $\square$

**Definition 1.** *A set  $\mathbb{X}$  of context-free grammars is called class of context-free grammars if 1.  $(N, \Sigma, P, S) \in \mathbb{X}$  and  $p \in P$  implies that  $(N, \Sigma, P \setminus \{p\}, S) \in \mathbb{X}$  and 2.  $\mathbb{X}$  is closed under identifying two nonterminals.*

A context-free grammar  $G = (N, \Sigma, P, S)$  is called *self-embedding* if there is some  $A \in N$  such that  $A \Rightarrow_G^+ w_1 A w_2$ , for  $w_1, w_2 \in (N \cup \Sigma)^*$ ; otherwise  $G$  is called *non self-embedding*.

**Lemma 4.** *Let  $\mathsf{X}$  be a class of context-free grammars. If  $G \in \mathsf{X}$  and  $L(G)$  is finite, then there is a non self-embedding  $G' \in \mathsf{X}$  with  $|G'| \leq |G|$  and  $L(G') = L(G)$ .*

The following result shows that Lemma 3 can be generalised from linear to context-free grammars that contain only a bounded number of nonterminals on the right-hand side of each of their productions:

**Lemma 5.** *Let  $G$  be a grammar with  $n$  productions generating a finite language such that every production of  $G$  contains at most  $k$  nonterminals on its right-hand side. Then  $|L(G)| \leq n^{(k+1)^n}$ .*

*Proof Sketch.* Since  $G$  generates a finite language, by Lemma 4, we can assume, without loss of generality, that it is non self-embedding. Thus, there is a non-terminal  $A$  whose productions are  $A \rightarrow w_1 \mid w_2 \mid \dots \mid w_m$  with  $w_i \in \Sigma^*$ , for  $1 \leq i \leq m \leq n$ . Replacing each occurrence of  $A$  by all of the  $w_i$  yields a grammar with less nonterminals. By iterating this operation, one obtains a trivial grammar with the above mentioned bound.  $\square$

**Corollary 2.** *Let  $\mathsf{X}$  be a class of CFGs with a bounded number of nonterminals occurring on the right-hand side of each production. Then  $\mathsf{Xcc}$  is unbounded.*

*Proof.* Let  $G \in \mathsf{X}$  contain  $n$  production rules and let  $k$  be the bound on the number of nonterminals occurring on the right-hand side of each production. Define  $f : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto n^{(k+1)^n}$ . Clearly,  $f$  is both monotone and unbounded. By Lemma 5, for all finite languages  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ , we have  $\rho(L) = |L| \leq \mathsf{Xc}(L)^{(k+1)^{\mathsf{Xc}(L)}} = f(\mathsf{Xc}(L))$ . Hence, by Theorem 3,  $\mathsf{Xcc}$  is unbounded.  $\square$

An immediate consequence of Corollary 2 is that for the class CNF of grammars in Chomsky normal form<sup>1</sup>, CNFcc is an unbounded complexity measure. Moreover, by Lemma 5, the number of words generated by a grammar  $G$  in CNF with  $n$  productions is bounded above by  $n^{3^n}$ , i.e.,  $|L(G)| \leq n^{3^n}$ .

Now, we show that the right-hand side of each production in a minimal context-free grammar covering a language whose longest word has length  $\ell$  contains at most  $\ell$  terminals.

**Lemma 6.** *Let  $\mathsf{X}$  be a class of CFGs,  $L$  be a finite language,  $\ell := \max\{|w| \mid w \in L\}$ , and  $G$  be a minimal  $\mathsf{X}$ -grammar with  $L(G) \supseteq L$ . Then for all productions of the form  $A \rightarrow u_0 B_1 u_1 B_2 \dots B_n u_n$  of  $G$  with  $u_0, u_1, \dots, u_n \in \Sigma^*$ , we have  $|u_0 u_1 \dots u_n| \leq \ell$ .*

<sup>1</sup> A context-free grammar  $G = (N, \Sigma, P, S)$  is said to be in *Chomsky normal form* if all productions are of the form  $A \rightarrow BC$ ,  $A \rightarrow a$ , or  $A \rightarrow \varepsilon$ , where  $A, B, C \in N$  and  $a \in \Sigma$ .

**Lemma 7.** *Let  $L$  be a finite language,  $\ell := \max\{|w| \mid w \in L\}$ , and  $G$  be a minimal LIN-grammar with  $L(G) \supseteq L$ . Then  $\max\{|w| \mid w \in L(G)\} \leq |L| \cdot \ell$ .*

**Lemma 8.** *Let  $\mathsf{X}$  be a class of CFGs such that every production in an  $\mathsf{X}$ -grammar contains at most  $k \geq 2$  nonterminals on its right-hand side, let  $L$  be a finite language,  $\ell := \max\{|w| \mid w \in L\}$ , and  $G$  be a minimal  $\mathsf{X}$ -grammar with  $L(G) \supseteq L$ . Then  $\max\{|w| \mid w \in L(G)\} \leq \ell \cdot k^{|L|}$ .*

*Proof Sketch.* Since  $G$  generates a finite language, by Lemma 4, we can assume, without loss of generality, that it is non self-embedding. Thus, the nonterminals  $A_1, A_2, \dots, A_p$  can be ordered such that every production with left-hand side  $A_i$  only contains nonterminals  $A_j$  with  $i > j$ . Thus, we show by induction that every derivation consists of at most  $\sum_{i=0}^{p-1} k^i \leq k^p$  steps. Since  $p \leq |G| \leq |L|$  and each derivation step can add at most  $\ell$  new terminals, any word derivable in  $G$  has length at most  $\ell \cdot k^{|L|}$ .  $\square$

**Theorem 5.** *Let  $\mathsf{X}$  be a class of CFGs such that every production in an  $\mathsf{X}$ -grammar contains at most  $k$  nonterminals on its right-hand side. Then, for every finite language  $L$ , there is a finite set  $\mathcal{S}_L$  of finite languages such that  $\mathsf{Xcc}(L) = \min\{\mathsf{Xc}(L') \mid L' \in \mathcal{S}_L\}$ .*

*Proof.* Let  $G$  be an arbitrary minimal  $\mathsf{X}$ -grammar with  $n$  productions covering a finite language  $L$ , i.e.,  $\mathsf{Xcc}(L) = n$ , and let  $\ell = \max\{|w| \mid w \in L\}$ . Clearly,  $n \leq |L|$ . We distinguish two cases. The case  $k = 1$  follows from Lemmas 3 and 7, since every  $\mathsf{X}$ -grammar covering  $L$  is an  $\mathsf{X}$ -grammar generating a finite language  $L' \supseteq L$  that satisfies  $\mathsf{Xc}(L') \leq |L'| \leq 2^{|L|-1}$  and  $\max\{|w| \mid w \in L'\} \leq \ell \cdot |L|$ . Similarly, the case  $k \geq 2$  follows from Lemmas 5 and 8. Hence, the sets

$$\mathcal{S}_{L,1} = \{L' \in \mathcal{P}_{\text{fin}}(\Sigma^*) \mid L \subseteq L', |L'| \leq 2^{|L|-1}, \max\{|w| \mid w \in L'\} \leq \ell \cdot |L|\}$$

and, for  $k \geq 2$ ,

$$\mathcal{S}_{L,k} = \{L' \in \mathcal{P}_{\text{fin}}(\Sigma^*) \mid L \subseteq L', |L'| \leq |L|^{(k+1)^{|L|}}, \max\{|w| \mid w \in L'\} \leq \ell \cdot k^{|L|}\}$$

are finite.  $\square$

So, for a class of CFGs as in Theorem 5, determining the cover complexity of  $L$  boils down to computing the exact complexity on the finite set  $\mathcal{S}_L$ .

## 5 Bounds on Language Operations

In this section, we will prove upper and lower bounds on the cover complexity of the operations *intersection*, *union*, and *concatenation*. Since the lower bounds are hard to show in the cover formulation, we have not yet been able to obtain lower bounds on union and concatenation for fixed alphabets. The only exceptions are union in the cases of strict regular and strict linear grammars as well as concatenation in the case of strict regular grammars. The results of this section are summarised in Fig. 1, where **bold font** means that we have matching upper and lower bounds w.r.t. a fixed alphabet and non-bold means that the bounds are matching w.r.t. a growing alphabet. For the remainder of this section, let  $\Delta = \Gamma \setminus \{CF\}$ .

	$\text{Xcc}(L_1 \cap L_2)$	$\text{Xcc}(L_1 \cup L_2)$	$\text{Xcc}(L_1 L_2)$
<i>LIN</i>	$\min\{c_1, c_2\}$	$c_1 + c_2$	$\min\{d_1 + c_2, c_1 + d_2\}$
<i>SLIN</i>	$\min\{c_1, c_2\}$	$c_1 + c_2$	$\min\{d_1 + c_2, c_1 + d_2\}$
<i>REG</i>	$\min\{c_1, c_2\}$	$c_1 + c_2$	$c_1 + c_2$
<i>SREG</i>	$\min\{c_1, c_2\}$	$c_1 + c_2$	$c_1 + c_2$

Fig. 1. Summary of results,  $c_i = \text{Xcc}(L_i)$  and  $d_i = (\text{S})\text{REGcc}(L_i)$ .

### 5.1 Intersection

**Theorem 6.** *Let  $X \in \Delta$  and  $L_1$  and  $L_2$  be finite languages. Then*

$$\text{Xcc}(L_1 \cap L_2) \leq \min\{\text{Xcc}(L_1), \text{Xcc}(L_2)\}.$$

*Proof.* Let  $G_i$  be a minimal  $X$ -grammar with  $L(G_i) \supseteq L_i$ , for  $i \in \{1, 2\}$ ; then  $L(G_i) \supseteq L_1 \cap L_2$ . Simply choose  $G = G_i$  with  $|G_i| = \min\{|G_1|, |G_2|\}$ .  $\square$

**Theorem 7.** *Let  $X \in \Delta$ . Then there exists a finite alphabet  $\Sigma$  such that for all  $m, n \geq 1$ , there are  $L_1, L_2 \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  with  $\text{Xcc}(L_1) \geq m$  and  $\text{Xcc}(L_2) \geq n$  such that*

$$\text{Xcc}(L_1 \cap L_2) \geq \min\{\text{Xcc}(L_1), \text{Xcc}(L_2)\}.$$

*Proof.* Let  $\Sigma$  be an arbitrary finite alphabet,  $m, n \geq 1$ . From Corollary 1, it follows that there are  $L_1, L_2 \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  with  $\text{Xcc}(L_1) \geq m$  and  $\text{Xcc}(L_2) \geq n$ . Assume, w.l.o.g.,  $\text{Xcc}(L_1) \leq \text{Xcc}(L_2)$ . Define  $L'_2 = L_1 \cup L_2$ . Then  $\text{Xcc}(L'_2) \geq \text{Xcc}(L_2) \geq \text{Xcc}(L_1)$ , for otherwise there would be a grammar generating  $L'_2 \supseteq L_2$  with less than  $\text{Xcc}(L_2)$  productions. Thus, we clearly have  $\text{Xcc}(L_1 \cap L'_2) = \text{Xcc}(L_1) = \min\{\text{Xcc}(L_1), \text{Xcc}(L'_2)\}$ .  $\square$

### 5.2 Union

**Theorem 8.** *Let  $X \in \Delta$  and  $L_1$  and  $L_2$  be finite languages. Then*

$$\text{Xcc}(L_1 \cup L_2) \leq \text{Xcc}(L_1) + \text{Xcc}(L_2).$$

*Proof.* Let  $X \in \Delta$  and, for  $i \in \{1, 2\}$ ,  $G_i = (N_i, \Sigma_i, P_i, S_i)$  be a minimal  $X$ -grammar with  $L(G_i) \supseteq L_i$  and  $|G_i| = \text{Xcc}(L_i)$  s.t.  $N_1 \cap N_2 = \emptyset$ . Since  $G_i$  is minimal and non self-embedding,  $S_i$  does not occur on the right-hand side of a production in  $P_i$ . Let  $S \notin N_1 \cup N_2$  and  $G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P, S)$  where

$$P = \{S \rightarrow \alpha \mid S_1 \rightarrow \alpha \in P_1 \text{ or } S_2 \rightarrow \alpha \in P_2\} \\ \cup \{A \rightarrow \alpha \in P_1 \mid A \neq S_1\} \cup \{A \rightarrow \alpha \in P_2 \mid A \neq S_2\}.$$

Clearly, we have  $L(G) = L(G_1) \cup L(G_2) \supseteq L_1 \cup L_2$  and  $|G| = |G_1| + |G_2|$ , that is,  $\text{Xcc}(L_1 \cup L_2) \leq \text{Xcc}(L_1) + \text{Xcc}(L_2)$ . Moreover,  $G_1, G_2 \in X$  implies  $G \in X$ .  $\square$



If we consider growing alphabets, then we can show that the above upper bound on the cover complexity of union is tight for all considered grammar types.

**Theorem 9.** *Let  $X \in \Delta$ . Then, for all  $m, n \geq 1$ , there exists a finite alphabet  $\Sigma$  and finite languages  $L_1$  and  $L_2$  with  $\text{Xcc}(L_1) = m$  and  $\text{Xcc}(L_2) = n$  such that*

$$\text{Xcc}(L_1 \cup L_2) \geq \text{Xcc}(L_1) + \text{Xcc}(L_2).$$

*Proof.* Let  $m, n \geq 1$ . Then define  $\Sigma = \{a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n\}$ ,  $L_1 = \{a_1, a_2, \dots, a_m\}$ , and  $L_2 = \{b_1, b_2, \dots, b_n\}$ . Consequently,  $L_1 \cup L_2 = \Sigma$  and, clearly,  $\text{Xcc}(L_1) = m$ ,  $\text{Xcc}(L_2) = n$ , and the language  $L_1 \cup L_2$  can only be covered by a trivial grammar. Therefore,  $\text{Xcc}(L_1 \cup L_2) = m + n = \text{Xcc}(L_1) + \text{Xcc}(L_2)$ .  $\square$

Now, we prove—with respect to a fixed alphabet—a lower bound on the strict regular and strict linear cover complexity of union that matches the upper bound. To do so, we use the fact that in the case of strict regular and strict linear grammars, there is a connection between the number of productions and the length of a longest word in the generated finite language.

**Lemma 9.** *Let  $L \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  and  $\ell = \max\{|w| \mid w \in L\}$ . Then*

$$\text{SREGcc}(L) \geq \ell \quad \text{and} \quad \text{SLINcc}(L) \geq \left\lfloor \frac{\ell}{2} + 1 \right\rfloor.$$

*Proof Sketch.* First, show by induction on the length  $k$  of a derivation of  $v \in \Sigma^*$  that  $k \geq \left\lfloor \frac{|v|}{2} + 1 \right\rfloor$ . Since in a strict linear grammar all right-hand sides of productions contain at most one nonterminal, no production can occur twice in such a derivation, for otherwise the generated language would be infinite. As a consequence, such a derivation uses  $k$  distinct productions in order to derive  $v$ . Thus, for some  $w \in \Sigma^*$  with  $|w| = \ell$ , we have  $k \geq \left\lfloor \frac{\ell}{2} + 1 \right\rfloor$ . The SREG-case can be shown using similar arguments.  $\square$

**Theorem 10.** *Let  $X \in \{\text{SREG}, \text{SLIN}\}$ . Then there exists a finite alphabet  $\Sigma$  such that for all  $m, n \geq 1$ , there are  $L_1, L_2 \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  with  $\text{Xcc}(L_1) = m$  and  $\text{Xcc}(L_2) = n$  such that*

$$\text{Xcc}(L_1 \cup L_2) \geq \text{Xcc}(L_1) + \text{Xcc}(L_2).$$

*Proof.* For  $X = \text{SREG}$ , let  $\Sigma = \{a, b\}$  and, for  $m, n \geq 1$ , we define the finite languages  $L_1 = \{a^m\}$  and  $L_2 = \{b^n\}$ . Moreover, let  $L = L_1 \cup L_2$ . Then, from Lemma 9, we get that  $\text{SREGcc}(L_1) \geq m$  and  $\text{SREGcc}(L_2) \geq n$ . It is easy to see that also  $\text{SREGcc}(L_1) \leq m$  and  $\text{SREGcc}(L_2) \leq n$ . Since the words in  $L_1$  and  $L_2$  do not share a common letter, there can be no production that is used to derive words from both  $L_1$  and  $L_2$ . Thus, we must have that  $\text{SREGcc}(L) = \text{SREGcc}(L_1 \cup L_2) \geq \text{SREGcc}(L_1) + \text{SREGcc}(L_2)$ .

For  $X = \text{SLIN}$ , let  $L_1 = \{a^{2^m-1}\}$  and  $L_2 = \{b^{2^n-1}\}$  and define  $L = L_1 \cup L_2$ . Then proceed analogous to the SREG-case using Lemma 9.  $\square$

### 5.3 Concatenation

**Theorem 11.** *Let  $X \in \{SREG, REG\}$  and  $L_1, L_2 \in \mathcal{P}_{\text{fin}}(\Sigma^*)$ . Then*

1.  $\text{Xcc}(L_1L_2) \leq \text{Xcc}(L_1) + \text{Xcc}(L_2)$ ,
2.  $\text{LINcc}(L_1L_2) \leq \min\{\text{REGcc}(L_1) + \text{LINcc}(L_2), \text{LINcc}(L_1) + \text{REGcc}(L_2)\}$ ,
3.  $\text{SLINcc}(L_1L_2) \leq \min\{\text{SREGcc}(L_1) + \text{SLINcc}(L_2), \text{SLINcc}(L_1) + \text{SREGcc}(L_2)\}$ .

*Proof Sketch.* Let  $G_i = (N_i, \Sigma_i, P_i, S_i)$  be a minimal  $X$ -grammar with  $L(G_i) \supseteq L_i$  and  $|G_i| = \text{Xcc}(L_i)$ , for  $i \in \{1, 2\}$ . Assume, without loss of generality, that  $N_1 \cap N_2 = \emptyset$ . First, note that in a right-linear and left-linear grammar all productions of the form  $A \rightarrow w$  with  $w \in \Sigma^*$  are used to derive the postfixes and prefixes of words, respectively.

For  $X \in \{SREG, REG\}$ , we construct an  $X$ -grammar covering  $L_1L_2$  by taking the union  $P_1 \cup P_2$  and replacing all productions of the form  $A \rightarrow w \in P_1$  with  $w \in \Sigma^*$  by  $A \rightarrow wS_2$ . Consequently,  $\text{Xcc}(L_1L_2) \leq \text{Xcc}(L_1) + \text{Xcc}(L_2)$ .

For  $X \in \{SLIN, LIN\}$ , let  $G_{(S)REG,i}$  and  $G_{(S)LIN,i}$  be minimal (S)REG- and (S)LIN-grammars covering  $L_i$ , for  $i \in \{1, 2\}$ . Assume that these grammars have pairwise disjoint sets of nonterminals. We define two (S)LIN-grammars  $G_1$  and  $G_2$  covering  $L_1L_2$  as follows:  $G_1$  is obtained by taking the union  $P_{(S)REG,1} \cup P_{(S)LIN,2}$  and replacing all productions of the form  $A \rightarrow w \in P_{(S)REG,1}$  with  $w \in \Sigma^*$  by  $A \rightarrow wS_{(S)LIN,2}$ . Similarly,  $G_2$  is obtained by taking the union  $P_{(S)LIN,1} \cup P_{(S)REG,2}$  and replacing all productions of the form  $A \rightarrow w \in P_{(S)REG,2}$  with  $w \in \Sigma^*$  by  $A \rightarrow S_{(S)LIN,1}w$ . Then simply take the grammar with the fewest number of productions out of  $G_1$  and  $G_2$ . Thus,  $\text{Xcc}(L_1L_2) \leq \min\{(\text{S})\text{REGcc}(L_1) + (\text{S})\text{LINcc}(L_2), (\text{S})\text{LINcc}(L_1) + (\text{S})\text{REGcc}(L_2)\}$ .  $\square$

The following lemma shows that a grammar covering the concatenation of two disjoint alphabets (where each contains at least two letters) needs at least as many productions as there are elements in their (disjoint) union. This lemma will play an important role in the proof of Theorem 12.

**Lemma 10.** *Let  $\Sigma = \Sigma_1 \uplus \Sigma_2$  with  $|\Sigma_1|, |\Sigma_2| \geq 2$ . Then for all CFGs  $G$  with  $L(G) \supseteq \Sigma_1\Sigma_2$ , we have  $|G| \geq |\Sigma_1| + |\Sigma_2|$ .*

*Proof Sketch.* Proceed by induction on  $|\Sigma|$ , making a case distinction in the base case  $|\Sigma| = 4$  and reducing the step case to the induction hypothesis by deleting productions that contain the new letter.  $\square$

**Theorem 12.** *Let  $X \in \{SREG, REG\}$ . Then, for all  $m, n \geq 2$ , there is a finite alphabet  $\Sigma$  and  $L_1, L_2 \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  with  $\text{Xcc}(L_1) = m$  and  $\text{Xcc}(L_2) = n$  s.t.*

1.  $\text{Xcc}(L_1L_2) \geq \text{Xcc}(L_1) + \text{Xcc}(L_2)$ ,
2.  $\text{LINcc}(L_1L_2) \geq \min\{\text{REGcc}(L_1) + \text{LINcc}(L_2), \text{LINcc}(L_1) + \text{REGcc}(L_2)\}$ ,
3.  $\text{SLINcc}(L_1L_2) \geq \min\{\text{SREGcc}(L_1) + \text{SLINcc}(L_2), \text{SLINcc}(L_1) + \text{SREGcc}(L_2)\}$ .

*Proof.* Let  $m, n \geq 2$  and define the alphabet  $\Sigma = \{a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n\}$  as well as the languages  $L_1 = \{a_1, a_2, \dots, a_m\}$ ,  $L_2 = \{b_1, b_2, \dots, b_n\}$ , and let  $X \in \{SREG, REG\}$ . Then clearly we have  $\text{Xcc}(L_1) = m$  and  $\text{Xcc}(L_2) = n$ .

Thus, since every  $X$ -grammar is context-free, we have by Lemma 10 that  $X_{\text{cc}}(\Sigma) = X_{\text{cc}}(L_1L_2) \geq m + n = X_{\text{cc}}(L_1) + X_{\text{cc}}(L_2)$  and  $(\text{S})\text{LIN}_{\text{cc}}(L_1L_2) \geq m + n = \min\{(\text{S})\text{REG}_{\text{cc}}(L_1) + (\text{S})\text{LIN}_{\text{cc}}(L_2), (\text{S})\text{LIN}_{\text{cc}}(L_1) + (\text{S})\text{REG}_{\text{cc}}(L_2)\}$ .  $\square$

**Theorem 13.** *There exists a finite alphabet  $\Sigma$  such that for all  $m, n \geq 1$ , there exist  $L_1, L_2 \in \mathcal{P}_{\text{fin}}(\Sigma^*)$  with  $\text{SREG}_{\text{cc}}(L_1) = m$  and  $\text{SREG}_{\text{cc}}(L_2) = n$  such that*

$$\text{SREG}_{\text{cc}}(L_1L_2) \geq \text{SREG}_{\text{cc}}(L_1) + \text{SREG}_{\text{cc}}(L_2).$$

*Proof.* Let  $\Sigma = \{a\}$  and, for  $m, n \geq 1$ , define  $L_1 = \{a^m\}$  and  $L_2 = \{a^n\}$ . From Lemma 9, we get  $\text{SREG}_{\text{cc}}(L_1) \geq m$  and  $\text{SREG}_{\text{cc}}(L_2) \geq n$ . It is easy to see that also  $\text{SREG}_{\text{cc}}(L_1) \leq m$  and  $\text{SREG}_{\text{cc}}(L_2) \leq n$ . Again, by Lemma 9, it follows that  $\text{SREG}_{\text{cc}}(L_1L_2) \geq m + n = \text{SREG}_{\text{cc}}(L_1) + \text{SREG}_{\text{cc}}(L_2)$ .  $\square$

## 6 Conclusion

In this paper, we have investigated cover complexity measures for finite languages on three different levels and shown that every complexity measure on finite languages naturally induces a corresponding cover complexity measure. We have characterised in which situations arbitrary complexity measures thus obtained are unbounded. Based on these rather abstract results, we have shown that every class of context-free grammars that allows only a bounded number of nonterminals on the right-hand side of each production induces an unbounded production cover complexity measure. This, in turn, entails that the production cover complexity of a finite language  $L$  can be obtained as the minimum of the exact production complexities of a finite number of supersets  $L'$  of  $L$ . Moreover, we have investigated upper and lower bounds on the production cover complexity of the language operations intersection, union, and concatenation (see Fig. 1). Generalising the incompressibility result of [10, 11] in a suitable fashion seems to be a promising starting point for improving the lower bounds from growing to fixed alphabets. In summary, we believe that the study of the complexity of finite languages is a fruitful research area with strong ties to both proof theory and more classical questions of descriptonal complexity.

**Acknowledgements.** The authors would like to thank Markus Holzer and the anonymous reviewers for several useful comments and suggestions concerning the results in this paper.

## References

1. Alspach, B., Eades, P., Rose, G.: A lower-bound for the number of productions required for a certain class of languages. *Discrete Appl. Math.* **6**(2), 109–115 (1983)
2. Bucher, W.: A note on a problem in the theory of grammatical complexity. *Theor. Comput. Sci.* **14**, 337–344 (1981)
3. Bucher, W., Maurer, H.A., Culik II, K.: Context-free complexity of finite languages. *Theor. Comput. Sci.* **28**, 277–285 (1984)

4. Bucher, W., Maurer, H.A., Culik II, K., Wotschke, D.: Concise description of finite languages. *Theor. Comput. Sci.* **14**, 227–246 (1981)
5. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. In: Champarnaud, J.-M., Ziadi, D., Maurel, D. (eds.) WIA 1998. LNCS, vol. 1660, pp. 43–56. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48057-9\\_4](https://doi.org/10.1007/3-540-48057-9_4)
6. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. *Theor. Comput. Sci.* **267**(1–2), 3–16 (2001)
7. Černý, A.: Complexity and minimality of context-free grammars and languages. In: Gruska, J. (ed.) MFCS 1977. LNCS, vol. 53, pp. 263–271. Springer, Heidelberg (1977). [https://doi.org/10.1007/3-540-08353-7\\_144](https://doi.org/10.1007/3-540-08353-7_144)
8. Dassow, J.: Descriptive complexity and operations – two non-classical cases. In: Pighizzini, G., Câmpeanu, C. (eds.) DCFS 2017. LNCS, vol. 10316, pp. 33–44. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-60252-3\\_3](https://doi.org/10.1007/978-3-319-60252-3_3)
9. Dassow, J., Harbich, R.: Production complexity of some operations on context-free languages. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 141–154. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31623-4\\_11](https://doi.org/10.1007/978-3-642-31623-4_11)
10. Eberhard, S., Hetzl, S.: Compressibility of finite languages by grammars. In: Shallit, J., Okhotin, A. (eds.) DCFS 2015. LNCS, vol. 9118, pp. 93–104. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-19225-3\\_8](https://doi.org/10.1007/978-3-319-19225-3_8)
11. Eberhard, S., Hetzl, S.: On the compressibility of finite languages and formal proofs. *Inf. Comput.* **259**, 191–213 (2018)
12. Gruska, J.: On a classification of context-free languages. *Kybernetika* **3**(1), 22–29 (1967)
13. Gruska, J.: Some classifications of context-free languages. *Inf. Control* **14**(2), 152–179 (1969)
14. Gruska, J.: Complexity and unambiguity of context-free grammars and languages. *Inf. Control* **18**(5), 502–519 (1971)
15. Gruska, J.: On the size of context-free grammars. *Kybernetika* **8**(3), 213–218 (1972)
16. Hetzl, S.: Applying tree languages in proof theory. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 301–312. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-28332-1\\_26](https://doi.org/10.1007/978-3-642-28332-1_26)
17. Hetzl, S., Leitsch, A., Reis, G., Tapolczai, J., Weller, D.: Introducing quantified cuts in logic with equality. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 240–254. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-08587-6\\_17](https://doi.org/10.1007/978-3-319-08587-6_17)
18. Hetzl, S., Leitsch, A., Reis, G., Weller, D.: Algorithmic introduction of quantified cuts. *Theor. Comput. Sci.* **549**, 1–16 (2014)
19. Hetzl, S., Leitsch, A., Weller, D.: Towards algorithmic cut-introduction. In: Bjørner, N., Voronkov, A. (eds.) LPAR 2012. LNCS, vol. 7180, pp. 228–242. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-28717-6\\_19](https://doi.org/10.1007/978-3-642-28717-6_19)
20. Pudlák, P.: Twelve problems in proof complexity. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) CSR 2008. LNCS, vol. 5010, pp. 13–27. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-79709-8\\_4](https://doi.org/10.1007/978-3-540-79709-8_4)
21. Tuza, Z.: On the context-free production complexity of finite languages. *Discrete Appl. Math.* **18**(3), 293–304 (1987)



# On the Grammatical Complexity of Finite Languages

Markus Holzer<sup>1</sup>(✉) and Simon Wolfsteiner<sup>2</sup>

<sup>1</sup> Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany  
holzer@informatik.uni-giessen.de

<sup>2</sup> Institut für Diskrete Mathematik und Geometrie, TU Wien,  
Wiedner Hauptstr. 8–10, 1040 Wien, Austria  
simon.wolfsteiner@tuwien.ac.at

**Abstract.** We study the grammatical production complexity of *finite* languages w.r.t. (i) different interpretations of approximation, i.e., equivalence, cover, and scattered cover, and (ii) whether the underlying grammar generates a finite or infinite language. In case the generated language is infinite, the intersection with all words up to a certain length has to be considered in order to obtain the finite language under consideration. In this way, we obtain six different measures for regular, linear context-free, and context-free grammars. We compare these measures according to the taxonomy introduced in [J. DASSOW, GH. PÄUN: *Regulated Rewriting in Formal Language Theory*, 1989] with each other by fixing the grammar type and varying the complexity measure and the other way around, that is, by fixing the complexity measure and varying the grammar type. In both of these cases, we develop an almost complete picture, which gives new and interesting insights into the old topic of grammatical production complexity.

## 1 Introduction

Measures of descriptonal complexity or cost functions have a long and fruitful history. Most approaches to defining descriptonal complexity measures are based on quantifying the ability of a device—automaton or grammar—to approximate languages (by finite subsets). The interesting quantities in the case of automata and grammars are, e.g., the number of states or transitions and the number of nonterminals or productions, respectively. For instance, finite languages can be represented by ordinary deterministic finite automata (DFAs) or by cover finite automata (CFAs)—roughly speaking a CFA is a DFA  $A$  and a natural number  $\ell$  whose accepted language is defined as  $L(A) \cap \Sigma^{\leq \ell}$ . For a precise definition of

---

This research was completed while the author was on leave at the Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany, in 2017 and is supported by the Vienna Science Fund (WWTF) project VRG12-004.

CFA, we refer the reader to [5]. This gives rise to two different complexity measures: for a finite language  $L \subseteq \Sigma^{\leq \ell}$ , one defines the DFA state and the CFA state complexity as<sup>1</sup>

$$\text{DFAc}(L) = \min\{|A| \mid A \text{ is a DFA and } L = L(A)\} \quad (1)$$

and

$$\text{DFAc}_\infty(L) = \min\{|A| \mid A \text{ is a DFA and } L = L(A) \cap \Sigma^{\leq \ell}\},$$

respectively, where  $|A|$  refers to the number of states of the automaton  $A$ . Obviously,  $\text{DFAc}_\infty(L) \leq \text{DFAc}(L)$ , for every finite language  $L$ . Note that equality is possible for certain finite languages. It is worth mentioning that although these measures look very similar in their definitions, they can differ quite immensely when applied to the same language. For instance, there is a finite language  $L$  such that  $\text{DFAc}_\infty(L) = 1$ , but  $\text{DFAc}(L) = n$ , for every  $n \geq 1$ . Hence, the gap between both measures can be arbitrarily large. In general, complexity gaps can be classified according to the different growth rates of the complexity measures. To this end, a notion was introduced in [6], which defines three categories  $\leq^1$ ,  $\leq^2$ , and  $\leq^3$  of increasing complexity gaps; a precise definition is given later. The aforementioned complexity gap of arbitrary size between DFAs and CFAs is a gap of highest type, which is simply written as  $c_\infty \leq_{\text{DFA}}^3 c$ . This is one example of a complexity measure, but one can find legions of other automata-based descriptive complexity measures in the literature.

In this paper, we study the *grammatical* production complexity of finite languages. This topic is not new, and already in [2–4], measures similar in definition to (1), for regular (REG), linear context-free (LIN), and context-free grammars (CFG) have been investigated; they are named  $Xc$ , for  $X \in \{\text{REG}, \text{LIN}, \text{CF}\}$ . For instance, there it has been shown that there are incompressible finite languages for each grammar type mentioned above, i.e., languages that need at least as many productions of a certain type as there are words in that language. To the best of our knowledge, a classification of these grammatical measures in the sense of [6] has not been done yet. We close this gap and, moreover, also consider natural variants of  $Xc$  by varying the equivalence condition  $L = L(A)$  in (1) to  $L \subseteq L(A)$  and  $L(A)$  finite (cover) or even  $L \leq L(A)$  and  $L(A)$  finite (scattered cover), where  $\leq$  refers to the scattered subword relation—similarly this can be done for  $L = L(A) \cap \Sigma^{\leq \ell}$  in  $Xc_\infty$ , too. This leads to the additional grammatical measures (i)  $Xcc$  and  $Xcc_\infty$  (cover) and (ii)  $Xsc$  and  $Xsc_\infty$  (scattered cover), for  $X \in \{\text{REG}, \text{LIN}, \text{CF}\}$ . The variation  $Xcc$  is inspired by recent results on proof complexity in first-order logic, a research topic, which, from a first glance, seems completely unrelated to grammatical complexity, that connects the number of certain inference rules used in a specific logical calculus with the number of productions needed to *cover* a certain finite language [7]. For further results on the cover complexity of finite languages, see also [8].

<sup>1</sup> Observe that it is common in the literature to refer to the DFA and the CFA state complexity as  $sc$  and  $csc$ , respectively. We adapted the notation in order to be consistent with the notation used throughout this paper.

We compare these measures according to the taxonomy introduced in [6] with each other by (i) fixing the grammar type and varying the complexity measure and (ii) by fixing the complexity measure and varying the grammar type. In both of these cases, we develop an almost complete picture. As a byproduct, we also show that there are finite languages with large complexity. More precisely, the language of even length palindromes  $P_n = \{ w\$w^R \mid w \in \{a, b\}^{\leq n} \}$  requires at least  $\Omega(2^n)$  productions to be generated by a regular grammar. Moreover, the triple language  $T_n = \{ w\$w\#w \mid w \in \{a, b\}^n \}$  can only be generated by grammars of type  $X$ , for  $X \in \{\text{REG}, \text{LIN}, \text{CF}\}$ , by simply listing all words in  $T_n$ , i.e.,  $\text{Xc}(T_n) = \Omega(2^n)$ .

## 2 Preliminaries

We assume that the reader is familiar with the basic notions of formal language theory as contained in [10]. Nevertheless, to fix notation and terminology, we introduce the basic notions and results relevant to this paper in this section.

Let  $\Sigma$  be a finite alphabet. Then  $\Sigma^*$  denotes the set of all words over the finite alphabet  $\Sigma$  including the *empty word*  $\varepsilon$  and we write  $\Sigma^+$  for  $\Sigma^* \setminus \{\varepsilon\}$ . The *length* of a word  $w$  in  $\Sigma^*$  is denoted by  $|w|$ . In particular, the length of the empty word  $\varepsilon$  is 0, i.e.,  $|\varepsilon| = 0$ . The *reversal* of a word is defined as follows:  $\varepsilon^R = \varepsilon$  and  $(wa)^R = aw^R$ , for  $w \in \Sigma^*$  and  $a \in \Sigma \cup \{\varepsilon\}$ . Let  $\ell \geq 0$ . Then  $\Sigma^\ell$  and  $\Sigma^{\leq \ell}$  refers to the set of all words over  $\Sigma$  of length exactly  $\ell$  and at most  $\ell$ , respectively. A subset  $L$  of  $\Sigma^*$  is called a *language*. Any language  $L \subseteq \Sigma^{\leq \ell}$ , for  $\ell \geq 0$ , is called *finite* and, unless stated otherwise, we always assume  $\ell = \max\{|w| \mid w \in L\}$ . If  $L$  is a subset of  $\Sigma^\ell$ , for  $\ell \geq 0$ , then  $L$  is called a *uniform language*. This means that in a uniform language all words have the same length.

A *context-free grammar* (CFG) is a quadruple  $G = (N, \Sigma, P, S)$ , where  $N$  and  $\Sigma$  are disjoint alphabets of *nonterminals* and *terminals*, respectively,  $S \in N$  is the *start symbol*, and  $P$  is a finite set of *productions* of the form  $A \rightarrow \alpha$ , where  $A \in N$  and  $\alpha \in (N \cup \Sigma)^*$ . As usual, the derivation relation of  $G$  is denoted by  $\Rightarrow_G$  and the reflexive and transitive closure of  $\Rightarrow_G$  is written as  $\Rightarrow_G^*$ . The *language generated* by  $G$  is defined as  $L(G) = \{ w \in \Sigma^* \mid S \Rightarrow_G^* w \}$ . We also consider the following restrictions of context-free grammars: (i) a context-free grammar is said to be *linear context-free* (LIN) if the productions are of the form  $A \rightarrow \alpha$ , where  $A \in N$  and  $\alpha \in \Sigma^*(N \cup \{\varepsilon\})\Sigma^*$ , and (ii) a context-free grammar is said to be *right-linear* or *regular* (REG) if the productions are of the form  $A \rightarrow \alpha$ , where  $A \in N$  and  $\alpha \in \Sigma^*(N \cup \{\varepsilon\})$ . Furthermore,  $\Gamma$  will denote the set of grammar types in the sequel, that is,  $\Gamma = \{\text{REG}, \text{LIN}, \text{CF}\}$ .

Let  $G = (N, \Sigma, P, S)$  be a context-free grammar. By  $|G|$ , we denote the number of productions of  $G$ , i.e., the cardinality of  $P$ . Then the (*exact*) *X-complexity* (or *X-complexity* for short) of a finite language  $L$  w.r.t.  $X$ -grammars, for  $X \in \Gamma$ , is defined as

$$\text{Xc}(L) = \min\{|G| \mid G \text{ is an } X\text{-grammar with } L = L(G) \text{ and } L(G) \text{ finite}\}.$$

The additional condition that  $L(G)$  is finite is redundant, but becomes important whenever we replace  $L = L(G)$  by  $L \subseteq L(G)$  or some other language-relating

property. Similarly, the *infinite X-complexity* of a finite language  $L \subseteq \Sigma^{\leq \ell}$  is defined as

$$Xc_{\infty}(L) = \min\{|G| \mid G \text{ is an } X\text{-grammar with } L = L(G) \cap \Sigma^{\leq \ell}\}.$$

Note that in the definition of  $Xc_{\infty}$ , the grammar  $G$  is allowed to generate an infinite language. If we replace  $L = L(G)$  and  $L = L(G) \cap \Sigma^{\leq \ell}$  in the definitions of  $Xc(L)$  and  $Xc_{\infty}(L)$ , respectively, by  $L \subseteq L(G)$  and  $L \subseteq L(G) \cap \Sigma^{\leq \ell}$ , respectively, then we get the definitions for the *X-cover-complexity*  $Xcc(L)$  and the *infinite X-cover-complexity*  $Xcc_{\infty}(L)$ , respectively. The *scattered subword relation*  $\leq$  is defined as follows: let  $w = w_1u_1w_2u_2 \dots u_{n-1}w_n$  be a word with  $w_i, u_j \in \Sigma^*$ , for  $1 \leq i \leq n$  and  $1 \leq j \leq n-1$ . Then the word  $w' = w_1w_2 \dots w_n$  is called a *scattered subword* of  $w$  and we write  $w' \leq w$  in this case. We extend the relation  $\leq$  from words to languages  $L_1$  and  $L_2$  as follows:  $L_1 \leq L_2$  if for all words  $w_1 \in L_1$ , there is a word  $w_2 \in L_2$  such that  $w_1 \leq w_2$ . If  $L_1 \leq L_2$  holds, we say that  $L_1$  is a *scattered sublanguage* of  $L_2$ . We, obtain the definitions for  $Xsc(L)$  and  $Xsc_{\infty}(L)$  if we replace  $L = L(G)$  and  $L = L(G) \cap \Sigma^{\leq \ell}$  by  $L \leq L(G)$  and  $L \leq L(G) \cap \Sigma^{\leq \ell}$ , respectively, in the definitions of  $Xc(L)$  and  $Xc_{\infty}(L)$ , respectively. This results in the *X-scattered-complexity* and the *infinite X-scattered-complexity*, respectively.

Note that the definitions of  $Xc$ ,  $Xcc$ , and  $Xsc$  also have the additional requirement that  $L(G)$  is a finite language. In the following,  $\mathcal{M}$  will denote the set of measure types, i.e.,  $\mathcal{M} = \{c, cc, sc, c_{\infty}, cc_{\infty}, sc_{\infty}\}$ . By definition, for  $\tau \in \mathcal{M}$ , the following relations hold:

$$CFG \leq_{\tau} LIN \leq_{\tau} REG, \tag{2}$$

where, for  $X, Y \in \Gamma$ , we define  $X \leq_{\tau} Y$  if and only if  $X_{\tau}(L) \leq Y_{\tau}(L)$ , for all finite languages  $L$ . In case that  $X \leq_{\tau} Y$ , we say that  $X$  is *more succinct than*  $Y$  (w.r.t. the complexity measure  $\tau$ ).

We say that  $G$  is a *minimal X-grammar*, for  $X \in \Gamma$ , w.r.t. the measure  $X_{\tau}$  with  $\tau \in \mathcal{M}$ , if  $|G| = X_{\tau}(L)$ . In the case of the measure  $Xc$ , we speak of a *minimal X-grammar* generating a finite language.

Finally, we show that grammars that generate non-trivial uniform languages do not contain  $\varepsilon$ -productions. To this end, we first need the following result on the length of words generated by some nonterminal from a grammar that describes a finite uniform language.

**Lemma 1.** *Let  $X \in \Gamma$  and  $G = (N, \Sigma, P, S)$  be a minimal X-grammar generating a finite and uniform language. Then, for all  $A \in N$ , all words occurring in the set  $\{w \in \Sigma^* \mid A \Rightarrow_{\mathcal{C}}^* w\}$  have the same length.*

An easy consequence of the previous lemma is the next theorem.

**Theorem 2.** *Let  $X \in \Gamma$  and  $G = (N, \Sigma, P, S)$  be a minimal X-grammar generating a finite and uniform language satisfying  $L(G) \neq \{\varepsilon\}$ . Then  $G$  is  $\varepsilon$ -free, i.e.,  $P$  does not contain any rule of the form  $A \rightarrow \varepsilon$ , for all  $A \in N$ .*



### 3 Some Bounds on the Various X-Complexities

First, we prove some upper bounds for the finite variants of the introduced grammatical measures. We obtain the following results:

**Theorem 3.** *Let  $L \subseteq \Sigma^{\leq \ell}$  be a finite language over the alphabet  $\Sigma$ . Then, for  $X \in \Gamma$  and  $Y \in \{\text{REG}, \text{LIN}\}$ , we have*

1.  $\text{Xc}(L) \leq \ell + 1$  if  $|\Sigma| = 1$ , and  $\text{Xc}(L) \leq (|\Sigma|^{\ell+1} - 1)/(|\Sigma| - 1)$ , otherwise,
2.  $\text{CFcc}(L) \leq |\Sigma| + 2$  and  $\text{Ycc}(L) \leq \ell + 1$  if  $|\Sigma| = 1$ , and  $\text{Ycc}(L) \leq (|\Sigma| + 1) \cdot \ell$ , otherwise, and
3.  $\text{Xsc}(L) = 1$  if  $L$  is non-empty, and  $\text{Xsc}(L) = 0$ , otherwise.

*Proof.* We argue as follows:

1. Every finite language  $L$  can be generated by a grammar of type  $X \in \Gamma$  by simply listing all words from  $L$ . Since there are at most  $\sum_{i=0}^{\ell} |\Sigma|^i$  words of length at most  $\ell$  in  $L$ , the upper bounds of  $\ell + 1$  and  $(|\Sigma|^{\ell+1} - 1)/(|\Sigma| - 1)$  follow for the cases  $|\Sigma| = 1$  and  $|\Sigma| \geq 2$ , respectively.
2. Consider the context-free grammar  $G' = (\{A, S'\}, \Sigma, P', S')$ , where  $P'$  consists of the productions  $S' \rightarrow A^\ell$  and  $A \rightarrow a$ , for  $a \in \Sigma \cup \{\varepsilon\}$ . Clearly, we have  $L(G') = \Sigma^{\leq \ell}$  and  $|G'| = |\Sigma| + 2$ , i.e.,  $\text{CFcc}(\Sigma^{\leq \ell}) \leq |\Sigma| + 2$ .

By assumption,  $L \subseteq \Sigma^{\leq \ell}$ . Thus, every grammar generating  $\Sigma^{\leq \ell}$  automatically covers the language  $L$ . For  $|\Sigma| \geq 2$ , consider the regular grammar  $G = (N, \Sigma, P, S)$ , where  $N = \{A_1, A_2, \dots, A_\ell\}$  with  $S = A_1$  and

$$P = \{A_i \rightarrow aA_{i+1} \mid a \in \Sigma \text{ and } 1 \leq i \leq \ell - 1\} \cup \{A_\ell \rightarrow a \mid a \in \Sigma\} \\ \cup \{A_i \rightarrow \varepsilon \mid 1 \leq i \leq \ell\}.$$

Obviously,  $L(G) = \Sigma^{\leq \ell}$  and  $|G| = (|\Sigma| + 1) \cdot \ell$ . In the case that  $|\Sigma| = 1$ , we simply list all  $\ell + 1$  words occurring in  $\Sigma^{\leq \ell}$ .

3. Assume that  $\Sigma = \{a_1, a_2, \dots, a_n\}$  and consider the language  $\{(a_1 a_2 \dots a_n)^\ell\}$ , which is generated by  $G = (\{S\}, \Sigma, \{S \rightarrow (a_1 a_2 \dots a_n)^\ell\}, S)$ , a regular grammar with a single production rule. Clearly, we have  $L \subseteq \{(a_1 a_2 \dots a_n)^\ell\}$ , for all nonempty languages  $L \subseteq \Sigma^{\leq \ell}$ . Thus,  $\text{Xsc}(L) \leq 1$ . Since any grammar with empty production set can only generate the empty language, we also have  $\text{Xsc}(L) \geq 1$ . In case  $L = \emptyset$ , we obviously have  $\text{Xsc}(L) = 0$ .  $\square$

What about lower bounds for these measures? Observe that  $\text{Xsc}(L) = 1$  is already a lower bound result. In the seminal paper [4] on concise description of finite languages by different types of grammars, certain languages have been identified that require at least a *polynomial* number of productions. The proofs of these results are based on [4, Lemma 2.1] which states some easy facts about *minimal* context-free grammars:

**Lemma 4.** *Let  $G = (N, \Sigma, P, S)$  be a minimal context-free grammar for the finite language  $L$ . Then, for every nonterminal  $A \in N \setminus \{S\}$ , there are strings  $\alpha_1$  and  $\alpha_2$  with  $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$  and  $\alpha_1 \neq \alpha_2$  such that  $A \rightarrow \alpha_1$  and  $A \rightarrow \alpha_2$  are*

in  $P$ . Moreover, for every  $A \in N \setminus \{S\}$ , the set  $L_A(G) = \{w \in \Sigma^* \mid A \Rightarrow_G^* w\}$  contains at least two words and there is no derivation of the form  $A \Rightarrow_G^+ \alpha A \beta$  with  $\alpha, \beta \in (N \cup \Sigma)^*$ . Finally, for every  $A \in N \setminus \{S\}$ , there are  $u_1, u_2, v_1, v_2 \in \Sigma^*$  such that  $u_1 A u_2 \neq v_1 A v_2$  as well as  $S \Rightarrow_G^* u_1 A u_2$  and  $S \Rightarrow_G^* v_1 A v_2$ .

Our first candidate with a large  $X$ -complexity is the language of all even palindromes (with middle marker)  $P_n = \{w\$w^R \mid w \in \{a, b\}^{\leq n}\}$ . We show that any regular grammar generating this language needs at least an exponential number of productions.

**Theorem 5.** *Let  $n \geq 1$ . Then  $\text{REGc}(P_n) \geq 2^n$ .*

*Proof.* In the proof, we will use the following result from [4, Lemma 2.2]: let  $G = (N, \Sigma, P, S)$  be a context-free grammar generating a finite language. Then there is a context-free grammar  $G_{\max} = (N_{\max}, \Sigma, P_{\max}, S)$  such that  $N_{\max} \subseteq N$ ,  $P_{\max} \subseteq P$ , and  $L(G_{\max}) = L_{\max}$ , where  $L_{\max}$  is the subset of  $L(G)$  consisting of the words of maximal length. In light of this result, it suffices to show that  $\text{REGc}(P'_n) \geq 2^n$ , for the language  $P'_n = \{w\$w^R \mid w \in \{a, b\}^n\}$ . To this end, assume that  $\Sigma = \{a, b, \$\}$  and  $G = (N, \Sigma, P, S)$  is a minimal regular grammar generating  $P'_n$  that contains a nonterminal  $A \in N \setminus \{S\}$ . By Lemma 4, there are derivations  $S \Rightarrow_G^* u_1 A$  and  $S \Rightarrow_G^* u_2 A$  with  $u_1, u_2 \in \Sigma^*$  and  $u_1 \neq u_2$  as well as  $v_1, v_2 \in \Sigma^*$  with  $A \Rightarrow_G^* v_1$ ,  $A \Rightarrow_G^* v_2$ , and  $v_1 \neq v_2$ . Note that we must have both  $|u_1| = |u_2|$  and  $|v_1| = |v_2|$ , for otherwise we would be able to derive words  $w_1$  and  $w_2$  with  $|w_1| \neq |w_2|$ , but  $P'_n$  only contains words of the same length. Since  $v_1 \neq v_2$  and  $|v_1| = |v_2|$ , it follows that both  $v_1 \neq \varepsilon$  and  $v_2 \neq \varepsilon$ . Let  $w \in \{a, b\}^n$  be arbitrary. We distinguish the following cases:

1. Suppose  $u_1 \in \{w\$\}\{a, b\}^*$ . Then we must have  $u_1 = w_1 w_2 \$w_2^R$ , where  $w = w_1 w_2$  and both  $v_1 \in \{a, b\}^*$  and  $v_2 \in \{a, b\}^*$  holds. Assume, w.l.o.g., that  $v_1 = w_1^R$ . Since  $v_1 \neq v_2$ , it follows that  $v_2^R \neq w_1$ . Thus,  $u_1 v_2 \notin P'_n$ . Contradiction.
2. Suppose  $u_1 \in \{a, b\}^*$ . Then we must have  $v_1, v_2 \in \{a, b\}^* \{ \$w^R \}$ . Assume  $w = u_1 w_2$ , for some  $w_2 \in \{a, b\}^*$  and, w.l.o.g.,  $v_1 = w_2 \$w_2^R u_1^R$ . Since  $v_1 \neq v_2$ , it follows that  $v_2 = w'_2 \$w_2^R u_1^R$  with  $w'_2 \neq w_2$ . Thus,  $u_1 v_2 \notin P'_n$ . Contradiction.

Consequently, we have  $N = \{S\}$  and so the only way to generate the language  $P'_n$  minimally with a regular grammar is to list all of its words using  $S$ .  $\square$

For linear context-free and context-free grammars, one observes that both measures  $\text{LINc}(P_n)$  and  $\text{Cfc}(P_n)$  are at most linear, as witnessed by the linear context-free grammar  $G = (N, \Sigma, P, S)$  with  $N = \{S_0, S_1, \dots, S_n\}$ ,  $\Sigma = \{a, b, \$\}$ , start symbol  $S = S_0$ , and the productions

$$P = \{S_i \rightarrow aS_{i+1}a, S_i \rightarrow bS_{i+1}b, S_i \rightarrow S_{i+1} \mid 0 \leq i \leq n - 1\} \cup \{S_n \rightarrow \$\},$$

satisfying  $L(G) = P_n$ , for  $n \geq 1$ .

Using similar arguments as in the proof of Theorem 5, one can show that the triple language  $T_n = \{w\$w\#w \mid w \in \{a, b\}^n\}$  has large  $X$ -complexity, for all grammar types  $X$  with  $X \in \Gamma$ . A detailed proof of this fact can be found in [9].

**Theorem 6.** *Let  $X \in \Gamma$  and  $n \geq 1$ . Then  $\text{Xc}(T_n) = 2^n$ .*

## 4 Relating Finite and Infinite Complexity Measures

In this section, we will consider several different complexity measures on finite languages and relate them according to a group of relations that vary in strength. By the very nature of these relations, one can distinguish two main categories: the first category fixes the measure type  $\tau \in \mathcal{M}$  and then compares the different grammar types in  $\Gamma$  with each other w.r.t. the measure type  $\tau$ —see, e.g., (2). The second category swaps the roles of measure and grammar type, i.e., some grammar type  $X \in \Gamma$  is fixed and then the different measure types in  $\mathcal{M}$  are compared with each other w.r.t. the grammar type  $X$ .

### 4.1 Relating Grammar Types

Now, we define several different relations on the grammar types in  $\Gamma$  w.r.t. some fixed measure type from  $\mathcal{M}$ . In this way, we classify the difference between different grammar types w.r.t. the same measure type. This is similar to the notion introduced in [6] for the nonterminal complexity and thus leads to a certain kind of production complexity hierarchy.

Let  $X, Y \in \Gamma$  and  $\tau \in \mathcal{M}$ . Then we write

- $X \leq_\tau Y$  if and only if  $X_\tau(L) \leq Y_\tau(L)$ , for all finite languages  $L$ ;
- $X \leq_\tau^1 Y$  if and only if there is a constant  $c$  such that  $X_\tau(L) \leq Y_\tau(L) + c$ , for all finite languages  $L$ , and there is a sequence of finite languages  $(L_i)_{i \geq 0}$  such that  $Y_\tau(L_i) - X_\tau(L_i) \geq i$ ;
- $X \leq_\tau^2 Y$  if and only if there is a constant  $c$  such that  $X_\tau(L) \leq Y_\tau(L) + c$ , for all finite languages  $L$ , and there is a sequence of finite languages  $(L_i)_{i \geq 0}$  such that

$$\lim_{i \rightarrow \infty} \frac{X_\tau(L_i)}{Y_\tau(L_i)} = 0;$$

- $X \leq_\tau^3 Y$  if and only if there is a constant  $c$  such that  $X_\tau(L) \leq Y_\tau(L) + c$ , for all finite languages  $L$ , and there is no function  $f: \mathbb{N} \rightarrow \mathbb{N}$  such that  $Y_\tau(L) \leq f(X_\tau(L))$ , for all finite languages  $L$ .

Clearly,  $X \leq_\tau^3 Y$  implies  $X \leq_\tau^2 Y$ , which, in turn, implies  $X \leq_\tau^1 Y$ . Moreover,  $X \leq_\tau^3 Y$  holds if the first condition of its definition is satisfied and there is a sequence  $(L_i)_{i \geq 0}$  of finite languages such that  $X_\tau(L_i) \leq k$ , for some constant  $k$  and  $Y_\tau(L_i) \geq i$ . We write  $X =_\tau Y$  if both  $X \leq_\tau Y$  and  $Y \leq_\tau X$  hold.

Now, we are going to relate the different grammar types in  $\Gamma$  w.r.t. the finite complexity measure types under investigation. As already mentioned earlier, the relation  $\text{CF} \leq_\tau \text{LIN} \leq_\tau \text{REG}$  holds by definition, for all  $\tau \in \mathcal{M}$ . The following result was shown in [4]:

**Theorem 7.** *It holds that  $\text{CF} \leq_c^2 \text{LIN} \leq_c^2 \text{REG}$ .*

Next, we show that we can only obtain  $\text{CF} \leq_\tau^3 \text{LIN}$  and  $\text{CF} \leq_\tau^3 \text{REG}$ , for  $\tau \in \{\text{c}, \text{cc}\}$ . As a prerequisite, we need the following result from [4].

**Lemma 8.** *Let  $G$  be a linear grammar generating a finite language with  $|G| \geq 1$ . Then  $|L(G)| \leq 2^{|G|-1}$  and  $|G| \geq \log |L(G)| + 1$ .*

Now, we are ready for the following theorem.

**Theorem 9.** *Let  $\tau \in \{\text{c}, \text{cc}\}$ . Then*

1.  $\text{CF} \leq_{\tau}^3 X$ , for  $X \in \{\text{REG}, \text{LIN}\}$ , and
2.  $\text{LIN} \not\leq_{\tau}^3 \text{REG}$  and  $\text{REG} \not\leq_{\tau}^3 \text{LIN}$ .

*Proof.* 1. Let  $L_n = \{a, b\}^{\leq n}$ , for  $n \geq 0$ . It was shown in [4], that  $\text{CFc}(L_n) \leq 4$  and  $\text{LINc}(L_n) \geq n + 1$ . From Lemma 8, it follows that also  $\text{LINcc}(L_n) \geq n + 1$ . Moreover, it holds that  $\text{CFcc}(L_n) \leq \text{CFc}(L_n) \leq 4$ . Consequently,  $\text{CF} \leq_{\tau}^3 \text{LIN}$ . Since  $\text{REG}\tau(L_n) \geq \text{LIN}\tau(L_n) \geq n + 1$ , we immediately get  $\text{CF} \leq_{\tau}^3 \text{REG}$ .

2. Let  $L$  be an arbitrary finite language and  $G$  be a minimal linear grammar with  $L(G) = L$ . Clearly,  $\text{REG}\tau(L) \leq |L| = |L(G)|$ . From Lemma 8, it follows that  $|L(G)| \leq 2^{\text{LIN}\tau(L)-1}$ . Thus,  $\text{REG}\tau(L) \leq |L(G)| \leq 2^{\text{LIN}\tau(L)-1}$ . The function  $f: \mathbb{N} \rightarrow \mathbb{N}$  with  $x \mapsto 2^{x-1}$  fulfills  $\text{REG}\tau(L') \leq f(\text{LIN}\tau(L'))$ , for all finite languages  $L'$ , i.e.,  $\text{LIN} \not\leq_{\tau}^3 \text{REG}$ . Since  $\text{LIN}\tau(L') \leq \text{REG}\tau(L')$ , setting  $f = \text{id}_{\mathbb{N}}$  yields  $\text{REG} \not\leq_{\tau}^3 \text{LIN}$ .  $\square$

For the  $X$ -scattered-complexity, we have the following situation:

**Theorem 10.** *Let  $X, Y \in \Gamma$ . Then  $\text{REG} =_{\text{sc}} \text{LIN} =_{\text{sc}} \text{CF}$ , but  $X \not\leq_{\text{sc}}^i Y$ , for  $i \in \{1, 2, 3\}$ .*

It remains to relate the different grammar types in  $\Gamma$  w.r.t. the infinite complexity measure types  $\text{c}_{\infty}, \text{cc}_{\infty}, \text{sc}_{\infty}$ . For the infinite  $X$ -complexity we have:

**Theorem 11.** *Let  $n \geq 1$ . Then  $\text{REGc}_{\infty}(P_n) = \Omega(2^n)$ .*

*Proof.* The idea of the proof is as follows: let  $G = (N, \Sigma, P, S)$  be a regular grammar that is a witness for  $\text{REGc}_{\infty}(P_n)$ . Then we construct a regular grammar generating the finite set  $L(G) \cap \Sigma^{\leq 2n+1}$ . Since this language is equal to  $P_n$ , we can apply Theorem 5 in order to obtain a lower bound on  $|G|$ . Although  $G$  may contain  $\varepsilon$ -productions, we can safely assume that  $G$  does not contain productions with a right hand-side longer than  $2n + 2$ , since none of these productions generates a word of length less than or equal to  $2n + 1$  even with erasing rules.

To keep the presentation simple, assume for a moment that the grammar  $G$  is in 2-normal form.<sup>2</sup> Then we apply the triple construction—see, e.g., [10]—to the grammar  $G$  and the *nondeterministic* finite automaton  $A = (Q, \Sigma, \delta, q_0, F)$  with  $Q = \{0, 1, \dots, 2n + 1\}$ , initial state  $q_0 = 0$ , final state set  $F = \{2n + 1\}$ , and the transition function  $\delta(i, a) = \{i + 1, 2n + 1\}$ , for  $0 \leq i < 2n + 1$  and  $a \in \Sigma$ . Then the regular grammar  $G' = (N', \Sigma, P', S')$  with  $N' = Q \times N \times Q$ , start symbol  $S' = [0, S, 2n + 1]$ , and the productions

$$P' = \{ [i, A, j] \rightarrow a \mid A \rightarrow a \in P, a \in \Sigma \cup \{\varepsilon\}, \text{ and } j \in \delta(i, a) \} \\ \cup \{ [i, A, k] \rightarrow a[j, B, k] \mid A \rightarrow aB \in P, a \in \Sigma \cup \{\varepsilon\}, \text{ and } j \in \delta(i, a) \}$$

<sup>2</sup> A regular grammar  $G = (N, \Sigma, P, S)$  is in 2-normal form if all productions in  $P$  are of the form  $A \rightarrow a$  and  $A \rightarrow aB$ , where  $A, B \in N$  and  $a \in \Sigma \cup \{\varepsilon\}$ .

generates the finite language  $L(G) \cap \Sigma^{\leq 2n+1}$ . Observe that  $|G'| \leq |G| \cdot (2n+2)^3$ , because the productions of the form  $A \rightarrow aB$  from  $P$  increase  $|G'|$  the most.

Now, we are ready to prove the stated claim. Assume to the contrary that we have  $|G| = \text{REG}_{c_\infty}(P_n) = o(2^n)$ , for a regular grammar  $G$ . Then we transform  $G$  into an equivalent regular grammar in 2-normal form and apply the above described intersection construction. Let  $G'$  refer to the result of these construction steps. The transformation into 2-normal form is done by simply splitting the right hand-side of each production into a sequence of productions of the appropriate form. This increases  $|G|$  by at most a factor of  $2n+2$ . Together with the increase of productions by the triple construction by a factor of at most  $(2n+2)^3$ , we conclude that  $|G'| = o(2^n)$ , because  $o(2^n) \cdot (2n+2)^4 = o(2^n)$ . Since the regular grammar  $G'$  with  $|G'| = o(2^n)$  generates  $P_n$ , we get a contradiction to Theorem 5. Thus, we obtain  $\text{REG}_{c_\infty}(P_n) = \Omega(2^n)$ .  $\square$

The taxonomy of the next theorem applies to the infinite exact  $X$ -complexity.

**Theorem 12.** *It holds that  $\text{CF} \leq_{c_\infty}^i \text{REG}$  and  $\text{LIN} \leq_{c_\infty}^i \text{REG}$ , for all  $i \in \{1, 2, 3\}$ .*

Finally, both the infinite  $X$ -cover and the infinite  $X$ -scattered-complexity do not classify according to the used taxonomy.

**Theorem 13.** *Let  $X, Y \in \Gamma$  and  $\tau \in \{\text{cc}_\infty, \text{sc}_\infty\}$ . Then we have  $X \not\leq_\tau^i Y$ , for all  $i \in \{1, 2, 3\}$ .*

We have to leave open the question whether  $\text{LIN} \leq_{\text{cc}}^i \text{REG}$  and  $\text{CF} \leq_{c_\infty}^j \text{LIN}$ , for  $i \in \{1, 2\}$  and  $j \in \{1, 2, 3\}$ , holds.

## 4.2 Relating Complexity Measure Types

Now, we introduce relations for measuring the difference between different measure types w.r.t. some fixed grammar type that we consider in this paper. Therefore, for  $\tau, \sigma \in \mathcal{M}$  and  $X \in \Gamma$ , we similarly define the relations  $\tau \leq_X \sigma$ ,  $\tau \leq_X^1 \sigma$ ,  $\tau \leq_X^2 \sigma$ , and  $\tau \leq_X^3 \sigma$  as in the beginning of the previous subsection. For instance,  $\tau \leq_X^1 \sigma$  if and only if there is a constant  $c$  such that  $X\tau(L) \leq X\sigma(L) + c$ , for all finite languages  $L$ , and there is a sequence of finite languages  $(L_i)_{i \geq 0}$  such that  $X\sigma(L_i) - X\tau(L_i) \geq i$ .

Clearly, the following chain of implications holds:  $\tau \leq_X^3 \sigma$  implies  $\tau \leq_X^2 \sigma$ , which, in turn, implies  $\tau \leq_X^1 \sigma$ . Moreover,  $\tau \leq_X^3 \sigma$  holds if the first condition of its definition is satisfied and there is a sequence  $(L_i)_{i \geq 0}$  of finite languages such that  $X\tau(L_i) \leq c$ , for some constant  $c$ , and  $X\sigma(L_i) \geq i$ .

We start with comparing the finite  $X$ - with the infinite  $X$ -measures. Except for the  $X$ -scattered-complexity, the infinite versions are more succinct than their finite counterparts.

**Lemma 14.** *Let  $X \in \Gamma$ . Then (i)  $c_\infty \leq_X c$  and (ii)  $\text{cc}_\infty \leq \text{cc}$ , but we have (iii)  $\text{sc} \leq_X \text{sc}_\infty$ .*

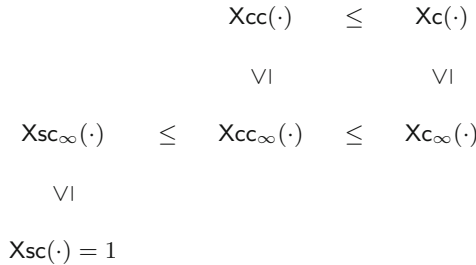
*Proof.* The first relation follows by definition. Let  $L \subseteq \Sigma^{\leq \ell}$ . Assume that  $G$  is a witness for  $\mathsf{Xc}(L)$ , i.e.,  $G$  generates a finite language,  $L = L(G)$ , and  $\mathsf{Xc}(L) = |G|$ . But then also  $L = L(G) \cap \Sigma^{\leq \ell}$ , which implies  $\mathsf{Xc}_\infty(L) \leq \mathsf{Xc}(L)$ . A similar argumentation applies to the second relation. For the third relation, we argue as follows: in Theorem 3, it was shown that  $\mathsf{Xsc}(L) = 1$  if  $L$  is non-empty and  $\mathsf{Xsc}(L) = 0$  if  $L = \emptyset$ . In the latter case, we also have  $\mathsf{Xsc}_\infty(L) = 0$ . Thus, we conclude that  $\mathsf{Xsc}_\infty(L) \leq \mathsf{Xsc}(L)$ , for every finite language  $L$ .  $\square$

It is worth mentioning that in the previous lemma, the argumentation used in the proof of the first two relations does not apply to the third one. This is seen as follows: consider the uniform finite language  $L = \{a, b\}$ . Then the regular grammar  $G = (\{S\}, \{a, b\}, \{S \rightarrow ab\}, S)$  witnesses  $\mathsf{Xsc}(L) = 1$ , because  $L$  is a scattered sublanguage of  $\{ab\}$ , i.e.,  $L \leq \{ab\} = L(G)$ . But  $L(G) \cap \{a, b\}^{\leq 1} = \emptyset$ . Thus,  $|G|$  cannot be used as an upper bound for  $\mathsf{Xsc}_\infty(L)$  and hence  $\mathsf{sc}_\infty \leq_X \mathsf{sc}$  does not hold in general. Thus, we conclude:

**Corollary 15.** *Let  $X \in \Gamma$ . Then  $\mathsf{sc}_\infty \not\leq_X \mathsf{sc}$ .*

Next, we compare the remaining  $X$ -complexities. Observe that  $L_1 = L_2$  implies  $L_1 \subseteq L_2$ , which, in turn, implies  $L_1 \leq L_2$ . As an easy consequence, we deduce that the (infinite)  $X$ -scattered-complexity is more succinct than the (infinite)  $X$ -cover-complexity and it is also easy to see that the (infinite)  $X$ -cover-complexity is more succinct than the (infinite)  $X$ -complexity. We summarize:

**Lemma 16.** *Let  $X \in \Gamma$ . Then (i)  $\mathsf{sc} \leq_X \mathsf{cc} \leq_X \mathsf{c}$  and (ii)  $\mathsf{sc}_\infty \leq_X \mathsf{cc}_\infty \leq_X \mathsf{c}_\infty$ .*



**Fig. 1.** Relations between the different grammatical complexity measures on finite languages. In this figure, a  $\leq$ -relation is of type  $(i, j, k)$ , for  $i, j, k \in \{1, 2, 3\}$ , if (i)  $\leq_{\text{CFG}}^i$ , (ii)  $\leq_{\text{LIN}}^j$ , and (iii)  $\leq_{\text{REG}}^k$  hold for the appropriate  $X$ -measures. All  $\leq$ -relations are of type  $(3, 3, 3)$ , except  $\mathsf{Xcc} \leq \mathsf{Xc}$ ,  $\mathsf{Xcc}_\infty \leq \mathsf{Xcc}$ ,  $\mathsf{Xsc} \leq \mathsf{Xsc}_\infty$ , and  $\mathsf{Xsc}_\infty \leq \mathsf{Xcc}_\infty$ , which are of types  $(3, 2, 2)$ ,  $(-, 3, 3)$ ,  $(-, -, -)$ , and  $(-, -, -)$ , respectively; the  $-$  sign means that it cannot be classified by the taxonomy.

The obtained  $\leq$ -relations are visualized in Fig. 1. For the measures  $\mathsf{cc}$  and  $\mathsf{c}_\infty$  we show incomparability w.r.t. the  $\leq$ -relation. Before we can prove this, we need two prerequisites. The first one is a lower bound on  $T_n$  w.r.t. the infinite

$X$ -complexity, which reads as follows—the proof is similar to the proof of Theorem 11 with the slight modification that we use 2-Greibach normal form<sup>3</sup> instead of 2-normal form, since we cannot apply the reasoning regarding the absence of  $\varepsilon$ -productions to context-free grammars in case of the  $c_\infty$ -measure. The change to 2-Greibach normal form induces the fourth root in the lower bound:

**Theorem 17.** *Let  $X \in \Gamma$ . Then  $Xc_\infty(T_n) = \Omega(2^{n/4})$ .*

The second prerequisite is an exact complexity bound for the language  $\Sigma^{\leq \ell}$  w.r.t. the finite  $X$ -cover-complexity.

**Lemma 18.** *Let  $X \in \Gamma$  and  $\ell \geq 2$ . Then  $CFcc(\Sigma^{\leq \ell}) = |\Sigma| + 2$ .*

Now, we are ready for the incomparability results:

**Theorem 19.** *Let  $X \in \Gamma$ . Then (i)  $cc \not\leq_X c_\infty$  and (ii)  $c_\infty \not\leq_X cc$ .*

*Proof.* For the proof of (i), observe that the grammar  $G = (N, \Sigma, P, S)$  with the productions  $P = \{S \rightarrow aS, S \rightarrow bS, S \rightarrow \varepsilon\}$  shows that  $Xc_\infty(\{a, b\}^{\leq n}) \leq 3$ . By Lemma 8, we have  $n + 1 \leq Ycc(\{a, b\}^{\leq n})$ , for  $n \geq 3$  and  $Y \in \{\text{REG}, \text{LIN}\}$ . As a consequence,  $Yc_\infty(\{a, b\}^{\leq n}) < Ycc(\{a, b\}^{\leq n})$ . From Lemma 18, it follows that  $CFcc(\{a, b\}^{\leq n}) \geq 4$ , i.e.,  $CFc_\infty(\{a, b\}^{\leq n}) < CFcc(\{a, b\}^{\leq n})$ . Next, we prove (ii). By Theorem 17,  $Xc_\infty(T_n) = \Omega(2^{n/4})$  and, by Theorem 3, we have  $XccT_n \leq 15n + 10$ . Thus,  $XccT_n < Xc_\infty(T_n)$ , for large enough  $n$ .  $\square$

In the remainder of this subsection, we classify the relations between the  $X$ -measures under consideration according to the taxonomy from [6].

**Theorem 20.** *Let  $X \in \Gamma$  and  $Y \in \{\text{REG}, \text{LIN}\}$ . Then*

1.  $cc \leq_{CFG}^3 c$  and  $cc \leq_Y^2 c$ , but  $\tau \not\leq_Y^3 \sigma$ , for all  $\tau, \sigma \in \{c, cc\}$  with  $\tau \neq \sigma$ ,
2.  $sc \leq_X^3 c$ ,
3.  $sc \leq_Y^3 cc$ , but  $\tau \not\leq_{CFG}^i \sigma$ , for all  $\tau, \sigma \in \{sc, cc\}$  with  $\tau \neq \sigma$  and all  $i \in \{1, 2, 3\}$ ,
4.  $c_\infty \leq_X^3 c$ ,
5.  $\tau \leq_X^3 c$ , for all  $\tau \in \{cc_\infty, sc_\infty\}$ ,
6.  $cc_\infty \leq_Y^3 cc$ , but  $\tau \not\leq_{CFG}^i \sigma$ , for  $\tau, \sigma \in \{cc, cc_\infty\}$  with  $\tau \neq \sigma$  and  $i \in \{1, 2, 3\}$ ,
7.  $cc \leq_{CFG}^3 c_\infty$ , and
8.  $\tau \leq_X^3 c_\infty$ , for all  $\tau \in \{sc, sc_\infty, cc_\infty\}$ .

*Proof.* We only prove the first statement. The remaining results can be shown with similar arguments. Let  $L_n = \{a^j b^j c^j \mid 1 \leq j \leq n\}$ . In [2], it was shown that  $CFc(L_n) = n$ . On the other hand, by Theorem 3, we have  $CFcc(L_n) \leq 5$ .

Let  $Y \in \{\text{REG}, \text{LIN}\}$  and  $T_n = \{w\$w\#w \mid w \in \{a, b\}^n\}$ . By Theorem 6, we have  $Yc(T_n) = 2^n$ . On the other hand, we have  $Ycc(T_n) \leq 15n + 10$  by Theorem 3.

<sup>3</sup> A context-free grammar  $G = (N, \Sigma, P, S)$  is in 2-Greibach normal form if all productions in  $P$  are of the form  $A \rightarrow a$ ,  $A \rightarrow aB$ ,  $A \rightarrow aBC$ , or  $S \rightarrow \varepsilon$ , where  $A \in N$ ,  $a \in \Sigma$ , and  $B, C \in N \setminus \{S\}$ . The transformation increases the number of productions by at most a polynomial of fourth degree [1].

Hence,  $\text{cc} \leq_Y^2 c$ . Finally, let  $L$  be an arbitrary finite language and  $f: \mathbb{N} \rightarrow \mathbb{N}$  a function defined by  $x \mapsto 2^{x-1}$ . Moreover, let  $G$  be a minimal  $Y$ -grammar with  $L(G) \supseteq L$ . By Lemma 8, we know that  $|L| \leq |L(G)| \leq 2^{\text{Ycc}(L)-1}$ . It holds that  $\text{Yc}(L) \leq |L|$ . Thus,

$$\text{Yc}(L) \leq |L| \leq |L(G)| \leq 2^{\text{Ycc}(L)-1} = f(\text{Ycc}(L)).$$

Hence,  $\text{cc} \not\leq_Y^3 c$ . If we set  $f = \text{id}_{\mathbb{N}}$ , it follows that  $c \not\leq_Y^3 \text{cc}$ .  $\square$

Finally, we list some incomparability results.

**Theorem 21.** *Let  $X \in \Gamma$  and  $Y \in \{\text{REG}, \text{LIN}\}$ . Then, for  $i \in \{1, 2, 3\}$ , we have*

1.  $\tau \not\leq_X^i \sigma$ , for every  $\tau, \sigma \in \{\text{cc}_\infty, \text{sc}_\infty\}$  with  $\tau \neq \sigma$ ,
2.  $\tau \not\leq_X^i \sigma$ , for every  $\tau, \sigma \in \{\text{cc}, \text{sc}_\infty\}$  with  $\tau \neq \sigma$ ,
3.  $\tau \not\leq_X^i \sigma$ , for every  $\tau, \sigma \in \{\text{sc}, \text{sc}_\infty\}$  with  $\tau \neq \sigma$ ,
4.  $\tau \not\leq_X^i \sigma$ , for every  $\tau, \sigma \in \{\text{sc}, \text{cc}_\infty\}$  with  $\tau \neq \sigma$ , and
5.  $\tau \not\leq_Y^i \sigma$ , for every  $\tau, \sigma \in \{\text{cc}, \text{c}_\infty\}$  with  $\tau \neq \sigma$ .

*Proof.* We only prove the first statement. The remaining results can be shown with similar arguments. First note that  $L \subseteq \Sigma^* \cap \Sigma^{\leq \ell}$  as well as  $L \leq \Sigma^* \cap \Sigma^{\leq \ell}$ , for all finite languages  $L$  over  $\Sigma$  with  $\ell = \max\{|w| \mid w \in L\}$ . The universal language  $\Sigma^*$ , for  $\Sigma = \{a_1, a_2, \dots, a_n\}$ , can be produced with the following regular productions:  $S \rightarrow a_1 S \mid a_2 S \mid \dots \mid a_n S \mid \varepsilon$ . Thus,  $\text{Xcc}_\infty(L) \leq |\Sigma| + 1$  and  $\text{Xsc}_\infty(L) \leq |\Sigma| + 1$ , for all finite languages  $L$  over  $\Sigma$ . This, however, means that  $\tau \leq_X^i \sigma$  does *not* hold for  $\tau, \sigma \in \{\text{cc}_\infty, \text{sc}_\infty\}$  and all  $i \in \{1, 2, 3\}$ .  $\square$

## References

1. Blum, N., Koch, R.: Greibach normal form transformation revisited. *Inform. Comput.* **150**(1), 112–118 (1999)
2. Bucher, W.: A note on a problem in the theory of grammatical complexity. *Theoret. Comput. Sci.* **14**(3), 337–344 (1981)
3. Bucher, W., Maurer, H.A., Culik II, K.: Context-free complexity of finite languages. *Theoret. Comput. Sci.* **28**(3), 277–285 (1983)
4. Bucher, W., Maurer, H.A., Culik II, K., Wotschke, D.: Concise description of finite languages. *Theoret. Comput. Sci.* **14**(3), 227–246 (1981)
5. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. *Theoret. Comput. Sci.* **267**(1–2), 3–16 (2001)
6. Dassow, J., Păun, Gh.: *Regulated Rewriting in Formal Language Theory*. EATCS Monographs in Theoretical Computer Science, vol. 18. Springer, Heidelberg (1989)
7. Eberhard, S., Hetzl, S.: On the compressibility of finite languages and formal proofs. *Inform. Comput.* **259**(2), 191–213 (2018)
8. Hetzl, S., Wolfsteiner, S.: Cover complexity of finite languages. In: Konstantinidis, S., Pighizzini, G. (eds.) *DCFS 2018*. LNCS, vol. 10952, pp. 139–150. Springer, Cham (2018)
9. Gruber, H., Holzer, M., Wolfsteiner, S.: *On Minimal Grammar Problems for Finite Languages* (2018, Submitted for publication)
10. Wood, D.: *Theory of Computation*. Wiley, New York (1987)





# State Grammars with Stores

Oscar H. Ibarra<sup>1</sup> and Ian McQuillan<sup>2</sup>(✉)

<sup>1</sup> Department of Computer Science, University of California,  
Santa Barbara, CA 93106, USA  
ibarra@cs.ucsb.edu  
<sup>2</sup> Department of Computer Science,  
University of Saskatchewan, Saskatoon, SK S7N 5A9, Canada  
mcquillan@cs.usask.ca

**Abstract.** State grammars are context-free grammars where the productions have states associated with them, and can only be applied to a nonterminal if the current state matches the state in the production. Once states are added to grammars, it is natural to add various stores, similar to machine models. With such extensions, productions can only be applied if both the state and the value read from each store matches between the current sentential form and the production. Here, generative capacity results are presented for different derivation modes, with and without additional stores. In particular, with the standard derivation relation, it is shown that adding reversal-bounded counters does not increase the capacity, and states are enough. Also, state grammars with reversal-bounded counters that operate using leftmost derivations are shown to coincide with languages accepted by one-way machines with a pushdown and reversal-bounded counters, and these are surprisingly shown to be strictly weaker than state grammars with the standard derivation relation (and no counters). Complexity results of some decision problems involving state grammars with counters are also studied.

**Keywords:** Grammars · Reversal-bounded counters  
Automata models · Matrix grammars

## 1 Introduction

State grammars were created by Kasai [11], and have context-free grammar rules with additional state components. As originally defined, they consist of a set of nonterminals  $V$ , a set of terminals  $\Sigma$ , an initial nonterminal  $S \in V$ , a set of states  $Q$ , an initial state  $q_0 \in Q$ , and a set of productions  $P$ . Instead of normal context-free productions of the form  $A \rightarrow w$ , where  $A \in V, w \in (V \cup \Sigma)^*$ , now productions

---

The research of O. H. Ibarra was supported, in part, by NSF Grant CCF-1117708. The research of I. McQuillan was supported, in part, by Natural Sciences and Engineering Research Council of Canada Grant 2016-06172.

are of the form  $(q, A) \rightarrow (p, w)$ , where  $q, p \in Q$ , and  $w$  was forced to be non-empty in Kasai's original formulation. Sentential forms are of the form  $(q, \alpha)$  where  $q \in Q, \alpha \in (V \cup \Sigma)^*$ . A production is only applicable to a sentential form if the state of the production matches the state of the sentential form. The original derivation relation considered by Kasai (later called the *leftish* derivation relation in [13] which we will call it here as well), was as follows:  $(q, uAv) \Rightarrow_{\text{lt}} (p, uww)$  if  $(q, A) \rightarrow (p, w) \in P$ , and  $A$  is the leftmost nonterminal in the sentential form that has a production that is applicable from the current state. A word is generated if there is some leftish derivation starting at the initial state and initial nonterminal that produces a word over  $\Sigma^*$ . The family of languages generated by such systems with  $\lambda$ -free rules, denoted by  $\mathcal{L}_{\text{lt}}(\lambda\text{-free-CFG-S})$ , was shown to be equal to the family of context-sensitive languages [11]. Later, it was shown that when including  $\lambda$  rules, the family produced,  $\mathcal{L}_{\text{lt}}(\text{CFG-S})$ , is equal to the family of recursively enumerable languages [15].

The definition of state grammars was extended shortly afterwards by Moriya [12] to also include a final state set  $F$ . Furthermore, he defined another derivation relation called the *free interpretation*, whereby any nonterminal can be rewritten that has a production defined on the current state, rather than the leftmost. With this derivation relation, the family of languages generated by state grammars,  $\mathcal{L}(\text{CFG-S})$ , was proven to equal the languages generated by matrix grammars (or  $\lambda$ -free matrix grammars for  $\lambda$ -free state grammars) [1].

The notion of combining grammars with states is a powerful one. It is then easy to add various stores to grammars that operate like machine models. It can also enable the study of trade-offs between numbers of states, nonterminals, productions, and stores, relevant to the area of descriptonal complexity. Changing the derivation relation and the rules allowed can also significantly change the families generated, obtaining many important language families as special cases.

In this paper, we will collate some of the existing generative capacity results on state grammars. In doing so, we provide a shorter alternative proof that state grammars (with the free interpretation) generate the same family as matrix grammars by using context-free grammars with regular control. A new derivation mode is defined where all nonterminals are rewritten from left-to-right until the last nonterminal, then this repeats starting again at the first nonterminal. State grammars with this mode are found to generate the recursively enumerable languages (or context-sensitive languages for  $\lambda$ -free grammars). We will then consider adding multiple reversal-bounded counters to state grammars (with the free interpretation) and find that this does not change the capacity beyond only having states. However, this system provides quite an easy way of describing languages. Furthermore, it is shown that leftmost derivations for state grammars are strictly weaker than leftmost derivations for state grammars with counters, which are then strictly weaker than state grammars with no counters using the free interpretation. Lastly, some complexity results are presented on state grammars with counters. Many proofs are omitted due to space constraints.

## 2 Preliminaries

Here, some notation used in the paper is presented; we refer to [7] for an introductory treatment of automata and formal languages. We assume knowledge of deterministic and nondeterministic finite automata, context-free grammars, context-sensitive languages, and the recursively enumerable languages.

An *alphabet*  $\Sigma$  is a finite set of symbols, a *word* over  $\Sigma$  is a finite sequence of symbols  $a_1 \cdots a_n$ ,  $n \geq 0, a_i \in \Sigma, 1 \leq i \leq n$ , and  $\Sigma^*$  (respectively  $\Sigma^+$ ) is the set of all words (non-empty words) over  $\Sigma$ . Then,  $\Sigma^*$  contains the empty word, denoted by  $\lambda$ . Given a word  $w \in \Sigma^*$ , the length of  $w$  is denoted by  $|w|$ , for  $a \in \Sigma$ ,  $|w|_a$  is the number of  $a$ 's in  $w$ , and for subsets  $X$  of  $\Sigma$ ,  $|w|_X = \sum_{a \in X} |w|_a$ . The set of letters occurring in  $w$ ,  $alph(w) = \{a \in \Sigma \mid |w|_a > 0\}$ . Given  $\Sigma = \{a_1, \dots, a_k\}$ , the Parikh map of  $w$  is  $\psi(w) = (|w|_{a_1}, \dots, |w|_{a_k})$ , extended to languages  $L$ ,  $\psi(L) = \{\psi(w) \mid w \in L\}$ . The commutative closure of  $L$ ,  $comm(L) = \{v \in \Sigma^* \mid \psi(v) = \psi(w) \text{ for some } w \in L\}$ . We will not define the notion of semilinear sets and languages here, but an equivalent definition is that a language  $L$  is semilinear if and only if it has the same commutative closure as some regular language [5]. Given  $u, v \in \Sigma^*$ , the shuffle of  $u$  and  $v$ , denoted by  $u \sqcup v$  is  $\{u_1 v_1 \cdots u_n v_n \mid u = u_1 u_2 \cdots u_n, v = v_1 v_2 \cdots v_n, u_i, v_i \in \Sigma^*, 1 \leq i \leq n\}$ .

The context-free languages are denoted by  $\mathcal{L}(\text{CFG})$ , the linear languages are denoted by  $\mathcal{L}(\text{LG})$ , the context-sensitive languages by  $\mathcal{L}(\text{CS})$ , and the right linear (regular languages) are denoted by  $\mathcal{L}(\text{REG})$ .

Moreover, we will discuss other families and grammars systems summarized in [1], such as matrix grammars. The languages generated by matrix grammars are denoted by  $\mathcal{L}(\text{M})$ , and the languages generated by  $\lambda$ -free matrix grammars are denoted by  $\mathcal{L}(\lambda\text{-free-M})$ .

## 3 Grammars with States

To start, we will formally define state grammars, following the notation of [12] with final states.

**Definition 1.** A state grammar (CFG-S), is a 7-tuple  $G = (V, \Sigma, P, S, Q, q_0, F)$ , where  $V$  is the finite nonterminal alphabet,  $\Sigma$  is the finite terminal alphabet,  $S \in V$  is the initial nonterminal,  $Q$  is the finite set of states ( $V, \Sigma, Q$  are disjoint),  $q_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of final states, and  $P$  is a finite set of productions of the form:

$$(q, A) \rightarrow (p, w),$$

where  $A \in V, w \in (V \cup \Sigma)^*, q, p \in Q$ . The grammar is said to be linear (and is an LG-S) if, for all productions  $(q, A) \rightarrow (p, w), w \in \Sigma^*(V \cup \{\lambda\})\Sigma^*$ . The grammar is said to be right linear (and is a RLG-S) if, for all productions  $(q, A) \rightarrow (p, w), w \in \Sigma^*(V \cup \{\lambda\})$ . In all cases,  $G$  is  $\lambda$ -free if all productions are to some  $(p, w)$  where  $w \in (V \cup \Sigma)^+$ .

A sentential form of  $G$  is any element of  $Q \times (V \cup \Sigma)^*$ . Four different methods of derivation will be defined. They are as follows:

1. The free interpretation derivation relation is defined such that  $(q, uAv) \Rightarrow (p, uxv)$  if,  $(q, A) \rightarrow (p, x) \in P$ , and  $u, v \in (V \cup \Sigma)^*$ . This is extended to the reflexive, transitive closure  $\Rightarrow^*$ . The language generated by  $G$  is

$$L(G) = \{w \mid (q_0, S) \Rightarrow^* (f, w), f \in F, w \in \Sigma^*\}.$$

2. The leftmost derivation relation is defined such that  $(q, uAv) \Rightarrow_{\text{lm}} (p, uxv)$  if,  $(q, A) \rightarrow (p, x) \in P$ ,  $(q, uA) \Rightarrow (p, ux)$ , and  $u \in \Sigma^*$ . This is extended to the reflexive, transitive closure  $\Rightarrow_{\text{lm}}^*$ . The leftmost language generated by  $G$  is

$$L_{\text{lm}}(G) = \{w \mid (q_0, S) \Rightarrow_{\text{lm}}^* (f, w), f \in F, w \in \Sigma^*\}.$$

3. The leftish derivation relation is defined such that  $(q, uAv) \Rightarrow_{\text{lt}} (p, uxv)$  if,  $(q, A) \rightarrow (p, x) \in P$ ,  $(q, uA) \Rightarrow (p, ux)$ , and for all  $B \in \text{alph}(u)$  with  $B \in V$ , then there is no production from  $(q, B)$ . This is extended to the reflexive, transitive closure  $\Rightarrow_{\text{lt}}^*$ . The leftish language generated by  $G$  is

$$L_{\text{lt}}(G) = \{w \mid (q_0, S) \Rightarrow_{\text{lt}}^* (f, w), f \in F, w \in \Sigma^*\}.$$

4. The circular derivation relation is, for  $v_0A_1v_1 \cdots A_nv_n, A_i \in V, v_i \in \Sigma^*, 0 \leq i \leq n$ ,

$$\begin{aligned} (p_0, v_0A_1v_1A_2 \cdots A_nv_n) &\Rightarrow_{\circ} (p_1, v_0x_1v_1A_2 \cdots A_nv_n) \Rightarrow_{\circ} \\ (p_2, v_0x_1v_1x_2v_2A_3 \cdots A_nv_n) &\Rightarrow_{\circ} \cdots \Rightarrow_{\circ} (p_n, v_0x_1v_1x_2v_2 \cdots x_nv_n), \end{aligned}$$

where  $(p_i, A_{i+1}) \rightarrow (p_{i+1}, x_{i+1}) \in P$  for all  $i, 0 \leq i < n$ . In this case, it is written

$$(p_0, v_0A_1v_1A_2 \cdots A_nv_n) \Rightarrow_{\bullet} (p_n, v_0x_1v_1x_2v_2 \cdots x_nv_n).$$

This is extended to  $\Rightarrow_{\bullet}^*$ , the reflexive, transitive closure of  $\Rightarrow_{\bullet}$ . Therefore, this relation rewrites all nonterminals from left-to-right, then repeats in a circular fashion. The circular language generated by  $G$  is

$$L_{\bullet}(G) = \{w \mid (q_0, S) \Rightarrow_{\bullet}^* (f, w), f \in F, w \in \Sigma^*\}.$$

We also sometimes associate labels  $P_{\Sigma}$  in bijective correspondence with  $P$ , and write  $u \xRightarrow[p]{\quad} v$ , if production  $p$  was applied from  $u$  to  $v$  (and similarly for the other derivation relations).

The family of languages generated by CFG-S grammars with the free interpretation (respectively the leftmost, leftish, and circular) derivation relation is denoted by  $\mathcal{L}(\text{CFG-S})$  (respectively  $\mathcal{L}_{\text{lm}}(\text{CFG-S})$ ,  $\mathcal{L}_{\text{lt}}(\text{CFG-S})$ ,  $\mathcal{L}_{\bullet}(\text{CFG-S})$ ). For each of these families, we precede the family with  $\lambda$ -free to represent those language generated by  $\lambda$ -free systems; e.g.  $\mathcal{L}_{\bullet}(\lambda\text{-free-CFG-S})$ . Similarly, replacing CFG-S with LG-S in these (or RLG-S) restricts the families to grammars where the rules are linear (or right linear).

*Example 2.* Let  $k \geq 2$ ,  $\Sigma = \{a_1, b_1, \dots, a_k, b_k\}$ , and  $G_k = (V, \Sigma, P, S, Q, q_0, F)$  where  $Q = \{q_0, \dots, q_{k-1}\}$ ,  $F = \{q_0\}$ , and  $P$  contains:

- $(q_0, S) \rightarrow (q_0, A_1 A_2 \cdots A_k)$ ,
- $(q_{i-1}, A_i) \rightarrow (q_i, a_i A_i b_i) \mid (q_i, a_i b_i)$ , for  $1 \leq i < k$ ,
- $(q_{k-1}, A_k) \rightarrow (q_0, a_k A_k b_k) \mid (q_0, a_k b_k)$ .

In any successful derivation using the free interpretation, states must follow a pattern in  $q_0(q_0 \cdots q_{k-1})^+ q_0$ , and from  $q_{i-1}$ , only productions on  $A_i$  can be applied, and they all must terminate on the last pass. Hence,  $L(G_k) = \{a_1^n b_1^n \cdots a_k^n b_k^n \mid n > 0\}$ .

For the first comparison, we see that the different derivation relations for linear and right linear grammars with states are the same.

**Proposition 3.** *The following are true:*

- $\mathcal{L}(\text{LG}) = \mathcal{L}(\text{LG-S}) = \mathcal{L}_{\text{lm}}(\text{LG-S}) = \mathcal{L}_{\text{lt}}(\text{LG-S}) = \mathcal{L}_{\bullet}(\text{LG-S})$ ,
- $\mathcal{L}(\text{REG}) = \mathcal{L}(\text{RLG-S}) = \mathcal{L}_{\text{lm}}(\text{RLG-S}) = \mathcal{L}_{\text{lt}}(\text{RLG-S}) = \mathcal{L}_{\bullet}(\text{RLG-S})$ .

*Proof.* It is obvious that the method of derivation does not matter for linear and right linear grammars.

A linear grammar (resp., a right linear grammar) can easily be simulated by such a grammar with one state. The converse follows by creating nonterminals in  $V \times Q$ . Then, for all productions of the form  $(q, A) \rightarrow (p, uBv)$ ,  $q, p \in Q$ ,  $A, B \in V$ ,  $u, v \in \Sigma^*$ , create a normal production  $(q, A) \rightarrow u(p, B)v$  (ie. the state stays on the nonterminal) and for all terminating productions of the form  $(q, A) \rightarrow (p, u)$ ,  $q, p \in Q$ ,  $A \in V$ ,  $u \in \Sigma^*$ , create  $(q, A) \rightarrow u$  if and only if  $p \in F$ . It is clear that the languages generated are the same.  $\square$

The following was mentioned in [13], and it follows by considering the standard simulation of context-free grammars with pushdown automata, but using the state of the pushdown to simulate the state of the state grammar.

**Proposition 4.**  $\mathcal{L}_{\text{lm}}(\text{CFG-S}) = \mathcal{L}(\text{CFG})$ .

As proven in [12], the family of languages generated by matrix grammars (respectively  $\lambda$ -free matrix grammars) is equal to the family generated by state grammars (respectively  $\lambda$ -free state grammars) with the free interpretation. An alternate, shorter proof can also be demonstrated by showing equivalence of state grammars to context-free grammars with regular control [1]. It is known that such grammars are equivalent to matrix grammars [1].

**Proposition 5.**  $\mathcal{L}(\text{CFG-S}) = \mathcal{L}(\text{M})$ , and  $\mathcal{L}(\lambda\text{-free-CFG-S}) = \mathcal{L}(\lambda\text{-free-M})$ .

When circular derivations are used, then it will be seen next that CFG-S grammars already generate all recursively enumerable languages. We use the notion of a complete derivation tree  $t$  of a context-free grammar [7], which is a tree where all nodes are labelled by either a nonterminal, a terminal, or  $\lambda$ , the root is labelled by the initial nonterminal, if a parent is labelled by  $A$  and its children are labelled by  $A_1, \dots, A_k$  from left to right, then  $A \rightarrow A_1 \cdots A_k$  is a production, if a node is labelled by  $\lambda$ , then it is the only child of its parent, and all leaves are labelled by terminals. The yield of a derivation tree,  $\text{yd}(t)$ , is the

sequence of terminals obtained via a preorder traversal. Given such a tree  $t$ , level  $i$  is all nodes at distance  $i$  from the root, and the level- $i$  word is the sequence of labels on the nodes of level  $i$  concatenated together from left to right. It is known that the set of yields of complete derivation trees of a grammar is exactly the language generated by the grammar [7].

State grammars with the circular derivation will be shown equivalent to *tree controlled grammars* which are defined as follows. A tree controlled grammar is a tuple  $G = (V, \Sigma, P, S, R)$ , where  $G' = (V, \Sigma, P, S)$  is a CFG, and  $R$  is a regular language over  $V \cup \Sigma$ . When considering context-free derivation trees in  $G'$ , a restriction on the trees is used as follows: the language generated by  $G$ ,  $L(G)$ , is equal to

$$\{\text{yd}(t) \mid t \text{ is a complete derivation tree of } G', \text{ for all levels } i \text{ but the last,} \\ \text{the level-}i \text{ word is in } R\}.$$

Let  $\mathcal{L}(\text{TREE})$  (respectively  $\mathcal{L}(\lambda\text{-free-TREE})$ ) be the family of languages generated by (respectively  $\lambda$ -free) tree controlled grammars. It is known that tree controlled grammars generate all recursively enumerable languages, and  $\lambda$ -free tree controlled grammars generate the context-sensitive languages [1]. Equivalence to tree controlled grammars therefore implies:

**Proposition 6.** *The following are true:*

- $\mathcal{L}_\bullet(\text{CFG-S}) = \mathcal{L}_{\text{lt}}(\text{CFG-S}) = \mathcal{L}(\text{TREE}) = \mathcal{L}(\text{RE})$ ,
- $\mathcal{L}_\bullet(\lambda\text{-free-CFG-S}) = \mathcal{L}_{\text{lt}}(\lambda\text{-free-CFG-S}) = \mathcal{L}(\lambda\text{-free-TREE}) = \mathcal{L}(\text{CS})$ .

Hence, the following hierarchies are obtained:

**Corollary 7.** *The following are true:*

- $\mathcal{L}(\text{CFG}) = \mathcal{L}_{\text{lm}}(\text{CFG-S}) \subsetneq \mathcal{L}(\text{CFG-S}) = \mathcal{L}(\text{M}) \subsetneq \mathcal{L}_\bullet(\text{CFG-S}) = \mathcal{L}_{\text{lt}}(\text{CFG-S}) = \mathcal{L}(\text{RE})$ ,
- $\mathcal{L}(\lambda\text{-free-CFG}) = \mathcal{L}_{\text{lm}}(\lambda\text{-free-CFG-S}) \subsetneq \mathcal{L}(\lambda\text{-free-CFG-S}) = \mathcal{L}(\lambda\text{-free-M}) \subsetneq \mathcal{L}_\bullet(\lambda\text{-free-CFG-S}) = \mathcal{L}_{\text{lt}}(\lambda\text{-free-CFG-S}) = \mathcal{L}(\text{CS})$ .

## 4 State Grammars with Stores

Now that states are attached to grammars, it is quite natural to attach one or more stores as well, just like machine models. Then, store contents can be part of sentential forms just as states are with state grammars. For example, one could define context-free grammars with states plus a pushdown store. This would be represented with a tuple just like a CFG-S but with an additional word over the pushdown alphabet  $\Gamma$  and a bottom-of-pushdown marker  $Z_0$ . In particular, the productions would be of the form  $(q, X, A) \rightarrow (p, \alpha, w)$ , where  $q, p$  are states,  $A$  is a nonterminal,  $w$  is over the nonterminal and terminal alphabets,  $X$  is the topmost symbol of the pushdown, and  $\alpha$  is the string to replace the topmost symbol of the pushdown. Sentential forms are therefore in  $Q \times \Gamma^+ \times (V \cup \Sigma)^*$ , and the derivation relation is defined in the obvious way. Here, we

will attach multiple reversal-bounded counters as stores as they are defined with reversal-bounded counter machines [9]. Explained briefly, a one-way  $k$ -counter machine is an NFA with  $k$  counters, each containing some non-negative integer, and the transition function can detect whether each counter is empty or not, and can increment, keep the same, or decrement each counter by one. Such a machine is  $r$ -reversal-bounded if the number of times each counter switches between non-decreasing and non-increasing is at most  $r$ . Then  $\mathcal{L}(\text{NCM})$  is the family of languages accepted by machines that are  $r$ -reversal-bounded  $k$ -counter machines, for some  $k, r \geq 1$ .

Since grammars with states using either circular or leftish derivations already generate all recursively enumerable languages, we will not consider those derivation relations with stores.

Denote the set of all context-free grammars with states and some number of reversal-bounded counters by CFG-SC, and the languages they generate with the free interpretation and the leftmost derivation modes by  $\mathcal{L}(\text{CFG-SC})$  and  $\mathcal{L}_{\text{lm}}(\text{CFG-SC})$  respectively. For each such grammar  $G = (V, \Sigma, P, S, Q, q_0, F)$  with  $k$  counters, productions are of the form  $(q, i_1, \dots, i_k, A) \rightarrow (p, l_1, \dots, l_k, w)$ , where  $p, q \in Q, i_j \in \{0, 1\}$  (a production with  $i_j = 0$  is applied if and only if counter  $j$  is 0),  $l_j \in \{-1, 0, +1\}$  (which changes the counter),  $A \in V, w \in (V \cup \Sigma)^*$ .

*Example 8.* Let  $G = (V, \{a, b\}, P, S, Q, q_0, \{q_f\})$  be a CFG-SC with 2 counters accepting  $\{w\$w \mid |w|_a = |w|_b \geq 0\}$ , where  $P$  is as follows:

- $(q_0, 0, 0, S) \rightarrow (q_0, 0, 0, A_1 A_2)$ ,
- $(q_0, i, j, A_1) \rightarrow (q_a, 1, 0, aA_1) \mid (q_b, 0, 1, bA_1)$ , for  $i, j \in \{0, 1\}$ ,
- $(q_a, i, j, A_2) \rightarrow (q_0, 0, 0, aA_2)$ ,  $(q_b, i, j, A_2) \rightarrow (q_0, 0, 0, bA_2)$ , for  $i, j \in \{0, 1\}$ ,
- $(q_0, i, i, A_1) \rightarrow (q_1, 0, 0, A_1)$ , for  $i \in \{0, 1\}$ ,
- $(q_1, 1, 1, A_1) \rightarrow (q_1, -1, -1, A_1)$ ,
- $(q_1, 0, 0, A_2) \rightarrow (q_1, 0, 0, \lambda)$ ,  $(q_1, 0, 0, A_1) \rightarrow (q_f, 0, 0, \$)$ .

To start,  $G$  switches to  $(q_0, 0, 0, A_1 A_2)$ . Then the derivation repeatedly guesses either that  $A_1$  derives an  $a$  or a  $b$ ; if it guesses it derives an  $a$ , it switches to  $q_a$  and increases the first counter, and then from  $q_a$ , only  $A_2$  can be rewritten and it must derive an  $a$  (similarly with the  $b$  case using  $q_b$  and the second counter). Therefore,  $A_1$  derives some sequence of terminals  $w$  and  $A_2$  must derive the same sequence, and the first counter contains  $|w|_a$  and the second contains  $|w|_b$ . At any point while in state  $q_0$ ,  $G$  can switch to  $q_1$  which repeatedly decreases both counters in parallel until both are verified to be zero at the same time, at which point both  $A_2$  and  $A_1$  are erased.

Before studying the generative capacity of CFG-SC, a definition is required. A CFG-SC  $G$  is in normal form if each counter makes exactly 1 reversal (once they decrease, they can no longer increase), and a terminal string is successfully generated when  $G$  enters a unique accepting state  $f$  and all the counters are zero. Moreover, at each step at most one counter is changed (i.e.,  $+1$  or  $-1$ ). We also assume that the state remembers when a counter enters a decreasing mode.

So, e.g., when counter  $i$  enters the decreasing mode, the state remembers that from that point on, counter  $i$  can no longer increase. When another counter  $j$  enters the decreasing mode, the state now remembers that counters  $i$  and  $j$  can no longer increase, etc.

**Lemma 9.** *Let  $G$  be an CFG-SC. We can effectively construct a CFG-SC  $G'$  in normal form such that  $L(G) = L(G')$ .*

Hence, we may assume that a CFG-SC is in normal form.

Next, it will be seen that reversal-bounded counters do not increase the generative capacity.

**Proposition 10.**  $\mathcal{L}(\text{CFG-S}) = \mathcal{L}(\text{CFG-SC}) = \mathcal{L}(\text{M})$ .

*Proof.* By [12] (and Proposition 5),  $\mathcal{L}(\text{CFG-S}) = \mathcal{L}(\text{M})$ , and clearly  $\mathcal{L}(\text{CFG-S}) \subseteq \mathcal{L}(\text{CFG-SC})$ .

Let  $G$  be such a CFG-SC. Assume without loss of generality that  $G$  is in normal form, and it therefore has  $k$  1-reversal bounded counters. Make a state grammar  $G'$  (without counters) over  $\Sigma \cup \Delta$  where  $\Delta = \{c_1, d_1, \dots, c_k, d_k\}$  are new symbols. Then,  $G'$  simulates  $G$  but, whenever it adds from counter  $i$ , it instead outputs terminal symbol  $c_i$ , and whenever it decreases from counter  $i$ , it outputs  $d_i$ . The states of  $G'$  also verify that  $G'$  starts by, for each counter  $i$ , simulating only productions associated with counter  $i$  being empty until it adds to the counter for the first time, then it simulates productions defined on counter  $i$  being positive (while outputting  $c_i$ 's), then simulates productions on counter  $i$  being positive (while outputting  $d_i$ 's), until some nondeterministically guessed spot after outputting some  $d_i$ , where it guesses that the counter is now empty, and then it only simulates productions on counter  $i$  being empty while not outputting any more  $c_i$ 's and  $d_i$ 's.  $G'$  operates in this fashion, as states were specifically marked in the normal form. Therefore,  $G'$  operates just like  $G$ , where it simulates all of the counters, making sure that for each counter, all additions occur before any subtractions, but it does not do any of the counting. If one then restricted the derivations of  $G'$  to those where the number of increases is the same as the number of decreases for each counter, then after erasing the letters of  $\Delta$ , it would give  $L(G)$ . But, consider the following regular language  $R = (c_1 d_1)^* \cdots (c_k d_k)^* \Sigma^*$ , and the commutative closure of  $R$ ,  $\text{comm}(R)$ . Let  $h$  be a homomorphism that erases all letters of  $\Delta$  and fixes all letters of  $\Sigma$ . Then,  $h(L(G') \cap \text{comm}(R))$  is exactly this language, where  $L(G') \cap \text{comm}(R)$  restricts words to only those that have the same number of  $c_i$ 's as  $d_i$ 's (and hence the same number of increases as decreases for each counter), and  $h$  erases the letters of  $\Delta$ . Hence,  $L(G) = h(L(G') \cap \text{comm}(R))$ .

Since  $G'$  is a normal grammar with states, it can be converted to a matrix grammar  $G''$  by Proposition 5. It is known that matrix grammars are closed under intersection with  $\mathcal{L}(\text{NCM})$  [16] (there, they used closure under the BLIND multicounter languages which is equivalent to  $\mathcal{L}(\text{NCM})$  [3]). Also, the commutative closure of every regular language is in  $\mathcal{L}(\text{NCM})$  [8]. So  $L'' = L(G'') \cap \text{comm}(R)$  is a language generated by a matrix grammar. Lastly, erasing all  $c_i$ 's



and  $d_i$ 's with  $h$  gives  $L(G)$ , and matrix grammars are closed under homomorphism [1]. Since this is a matrix grammar, it can be converted back to a normal state grammar by Proposition 5 generating the same language as  $L(G)$ .  $\square$

To note, in the proof above, in  $G$ , despite the counters being 1-reversal-bounded, productions can be applied to any nonterminal in the sentential form. Thus, some counter additions could occur when rewriting a nonterminal to the right of other nonterminals that get rewritten with a production that decreases. Hence, when intersecting with an  $\mathcal{L}(\text{NCM})$  language, it must not enforce that all  $c_i$ 's occur before any  $d_i$ 's.

The following corollary is true, since the results are known to be true for matrix grammars [6].

**Corollary 11.** *The following are true:*

1. *Every unary language generated by a CFG-SC is regular.*
2. *The emptiness problem for CFG-SC is decidable.*

Next, we will study leftmost derivations of CFG-SC's. We will show that  $\mathcal{L}_{\text{lm}}(\text{CFG-SC}) = \mathcal{L}(\text{NPCM}) \subsetneq \mathcal{L}(\text{CFG-SC})$ , where NPCMs are one-way nondeterministic pushdown automata augmented by reversal-bounded counters [9]. To help, we need the notion of a CFG with monotonic counters introduced in [10]. This is a simpler model of grammars with counters that do not have states. At each step in the derivation, the counters can be incremented by 0 or +1, but not decremented. A derivation in this grammar starts with the counters having value zero. A terminal string  $w$  is in the language of the grammar if there is a derivation of  $w$  that ends with all counters having the same value.

A CFG with *monotonic counters* (CFG-MC) is a 5-tuple,  $G = (\Sigma, V, S, k, P)$ , where  $\Sigma$  is the set of terminals,  $V$  is the set of nonterminals,  $S \in V$  is the initial nonterminal,  $k$  is the number of monotonic counters, all are initially set to 0, and  $P$  is the set of rules of the form:  $A \rightarrow (z, c_1, \dots, c_k)$ , where  $A \in V$ ,  $z \in (V \cup \Sigma)^*$  and  $c_i = 0$  or  $+1$ .

The language defined is  $L(G) = \{w \mid w \in \Sigma^*, (S, 0, \dots, 0) \Rightarrow^* (w, n, \dots, n)$  for some  $n \geq 0\}$ .

The following result was shown in [10]:

**Proposition 12.**  *$\mathcal{L}(\text{NPCM})$  is equal to the family of languages generated by CFG-MCs (using either the leftmost derivation relation or the normal derivation relation).*

From this, the following can be shown:

**Lemma 13.**  $\mathcal{L}(\text{NPCM}) = \mathcal{L}_{\text{lm}}(\text{CFG-SC})$ .

*Proof.* Every CFG-MC  $G$  with a leftmost derivation can be simulated by a CFG-SC with a leftmost derivation. It starts by simulating with one state. Then, before terminating, it guesses all counters are equal, decreases them all to verify, then terminates. Thus,  $\mathcal{L}(\text{NPCM}) \subseteq \mathcal{L}_{\text{lm}}(\text{CFG-SC})$ . For the reverse containment, consider the standard simulation on a CFG with an NPDA with a leftmost

derivation. This same construction can work with states while the counters of the CFG-SC can be simulated by the counters of the NPCM.  $\square$

**Lemma 14.**  $\mathcal{L}(\text{NPCM}) \subsetneq \mathcal{L}(\text{M})$ .

*Proof.* It is clear that  $\mathcal{L}(\text{CFL}) \subseteq \mathcal{L}(\text{M})$ , it is known that  $\mathcal{L}(\text{M})$  is closed under intersection with  $\mathcal{L}(\text{NCM})$  [16], and homomorphism [1]. Recently, a Chomsky-Schützenberger-like theorem was shown that demonstrates that every language in NPCM can be obtained by some Dyck language (which is context-free) intersected with an  $\mathcal{L}(\text{NCM})$  language, then mapped via a homomorphism [10]. Therefore,  $\mathcal{L}(\text{NPCM}) \subseteq \mathcal{L}(\text{M})$ .

It is known that every NPCM language is semilinear [9]. Now  $\mathcal{L}(\text{CFG-SC}) = \mathcal{L}(\text{M})$  by Proposition 5. It known that matrix grammars can generate non-semilinear languages, e.g., a matrix grammar can generate the non-semilinear language [1],  $L = \{a^n b^m \mid 1 \leq n < m \leq 2^n\}$ . It follows that  $\mathcal{L}(\text{NPCM}) \subsetneq \mathcal{L}(\text{M})$ .  $\square$

Therefore, the following hierarchy exists by Propositions 4, 5, 6, 10 and Lemmas 13 and 14.

**Proposition 15.** *The following is true:*

$$\mathcal{L}(\text{CFG}) = \mathcal{L}_{\text{lm}}(\text{CFG-S}) \subsetneq \mathcal{L}_{\text{lm}}(\text{CFG-SC}) = \mathcal{L}(\text{NPCM}) \subsetneq \mathcal{L}(\text{CFG-S}) = \mathcal{L}(\text{CFG-SC}) = \mathcal{L}(\text{M}) \subsetneq \mathcal{L}_{\text{lt}}(\text{CFG-S}) = \mathcal{L}_{\bullet}(\text{CFG-S}) = \mathcal{L}(\text{RE}).$$

Lastly, we study the emptiness problem for a restriction of CFG-SCs. A CFG-S  $G = (V, \Sigma, P, S, Q, q_0, F)$  is of index  $m$ , ( $m \geq 1$ ) if, for every  $w \in L(G)$ , there exists some derivation  $(p_0, \alpha_0) \Rightarrow (p_1, \alpha_1) \Rightarrow \dots \Rightarrow (p_n, \alpha_n), p_0 = q_0, \alpha_0 = S, p_n \in F, \alpha_n = w$  with  $|\alpha_i|_V \leq m$ , for all  $0 \leq i \leq n$ . If it is index  $m$  for some  $m$ , then it is finite-index. This property is more general than requiring that every derivation of a word in the language has at most  $m$  nonterminals, a notion that is called uncontrolled index  $m$ , or uncontrolled finite-index. For context-free grammars, the languages generated by uncontrolled finite-index grammars corresponds to pushdown automata with a reversal-bounded pushdown [2], which cannot accept languages such as  $\{a^n b^n \mid n > 0\}^*$  that can be generated by an index 2 grammar [14]. Hence, finite-index CFG-S are quite general. Clearly, the definitions easily extend to  $m$ -index CFG-SC (finite-index CFG-SC).

**Proposition 16.** *Let  $m, k \geq 1$  be fixed. The emptiness problem for  $m$ -index CFG-SC with at most  $k$  1-reversal counters is decidable in polynomial time.*

*Proof.* Let  $G$  be an  $m$ -index CFG-SC with at most  $k$  1-reversal counters. We first construct from  $G$  a grammar  $G'$  where all terminal symbols are mapped to  $\lambda$ . Clearly,  $L(G')$  is empty if and only if  $L(G)$  is empty. Then all rules in  $G'$  are of the form:

$$(q, i_1, \dots, i_k, A) \rightarrow (p, l_1, \dots, l_k, u),$$

where  $i_j \in \{0, 1\}, l_j \in \{-1, 0, 1\}$  for  $1 \leq j \leq k$ ,  $u$  is string of nonterminals of length at most  $m$  (possibly  $\lambda$ ; there are no terminals used). Now from  $G'$ , we

construct an NCM  $M$  with  $k$  1-reversal counters as follows: Its initial state is  $[q_0, S]$ , where  $q_0$  is the initial state of  $G'$  and  $S$  is the start nonterminal of  $G'$ . The other states of  $M$  are of the form  $[q, w]$ , where  $q$  is a state of  $G'$ , and  $w$  is a string of at most  $m$  nonterminals (possibly  $\lambda$ ).

Then  $M$  starts in state  $[q_0, S]$  with all its  $k$  counters zero. A move of  $M$  is defined by: if  $G'$  has a rule

$$(q, i_1, \dots, i_k, A) \rightarrow (p, l_1, \dots, l_k, v),$$

$i_j \in \{0, 1\}, l_j \in \{-1, 0, 1\}$  for  $1 \leq j \leq k$ , then in  $M$ , for all strings  $xAy$  with  $|xAy| \leq m$  and  $|xvy| \leq m$ , create transitions from state  $[q, xAy]$  and counter status  $i_1, \dots, i_k$  on  $\lambda$ , that goes to state  $[p, xvy]$  and updates the counters by  $l_1, \dots, l_k$ . The accepting states of  $M$  are of the form  $[p, \lambda]$ , where  $p$  is an accepting state of  $G'$ .

Clearly, since  $G$  (and, hence,  $G'$ ) is  $m$ -index and  $m$  is fixed, the size of  $M$  is polynomial in the size of  $G'$  (hence, of  $G$ ). Since  $M$  is an NCM with a fixed ( $k$ ) number of 1-reversal counters and it is known that the emptiness problem for NCM with a fixed number of 1-reversal counters is decidable in polynomial time [4], the result follows.  $\square$

Note that if the index of  $G$  is not fixed, the size of  $M$  is no longer polynomial in the size of  $G'$  (and, hence, of  $G$ ). We also note that Proposition 16 can be generalized to hold for many  $m$ -index grammar systems with  $k$  1-reversal counters (for fixed  $m$  and  $k$ ).

## 5 Conclusions and Future Directions

State grammars were studied, and it was shown that with a new circular derivation relation, they generate all recursively enumerable languages. Then, using the free interpretation and the leftmost derivation relations, additional stores were added to state grammars; in particular, some number of reversal-bounded counters. When using the free interpretation derivation relation, the counters do not add any generative capacity, and only states are needed. When using leftmost derivations, the class coincides with the machine model NPCM (pushdown automata with reversal-bounded counters). This leads to the result that state grammars with counters and leftmost derivations are strictly weaker than state grammars with no counters and the free interpretation derivation relation.

There are many interesting problems of descriptiveness that are open. For example, do state grammars form an infinite hierarchy with the number of states? We conjecture that in Example 2, for each  $k \geq 2$ , it is impossible to generate  $L_k$  with a state grammar with fewer than  $k$  states, which would form such a hierarchy.

## References

1. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. EATCS Monographs on Theoretical Computer Science. Springer, Heidelberg (1989)
2. Ginsburg, S., Spanier, E.: Finite turn pushdown automata. *SIAM J. Control* **4**(3), 429–453 (1966)
3. Greibach, S.: Remarks on blind and partially blind one-way multicounter machines. *Theoret. Comput. Sci.* **7**, 311–324 (1978)
4. Gurari, E.M., Ibarra, O.H.: The complexity of decision problems for finite-turn multicounter machines. *J. Comput. Syst. Sci.* **22**(2), 220–229 (1981)
5. Harrison, M.: Introduction to Formal Language Theory. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Co., Boston (1978)
6. Hauschildt, D., Jantzen, M.: Petri net algorithms in the theory of matrix grammars. *Acta Informatica* **31**(8), 719–728 (1994)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
8. Ibarra, O., McQuillan, I.: The effect of end-markers on counter machines and commutativity. *Theoret. Comput. Sci.* **627**, 71–81 (2016)
9. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. *J. ACM* **25**(1), 116–133 (1978)
10. Ibarra, O.H.: Grammatical characterizations of NPDAs and VPDAs with counters. In: Han, Y.S., Salomaa, K. (eds.) 21st International Conference on Implementation and Application of Automata, CIAA 2016, Seoul, South Korea. Lecture Notes in Computer Science, vol. 9705, p. 11 (2016). Invited abstract, journal version submitted
11. Kasai, T.: An hierarchy between context-free and context-sensitive languages. *J. Comput. Syst. Sci.* **4**(5), 492–508 (1970)
12. Moriya, E.: Some remarks on state grammars and matrix grammars. *Inf. Control* **23**, 48–57 (1973)
13. Moriya, E., Hofbauer, D., Huber, M., Otto, F.: On state-alternating context-free grammars. *Theoret. Comput. Sci.* **337**(1), 183–216 (2005)
14. Rozenberg, G., Vermeir, D.: On the effect of the finite index restriction on several families of grammars. *Inf. Control* **39**, 284–302 (1978)
15. Salomaa, A.: Matrix grammars with a leftmost restriction. *Inf. Control* **20**(2), 143–149 (1972)
16. Stiebe, R.: Slender matrix languages, pp. 375–385. World Scientific (2000)



# Error-Free Affine, Unitary, and Probabilistic OBDDs

Rishat Ibrahimov<sup>1,3</sup>, Kamil Khadiev<sup>1,2,3(✉)</sup>, Krišjānis Prūsis<sup>2</sup>,  
and Abuzer Yakaryılmaz<sup>2</sup>

<sup>1</sup> Kazan Federal University, Kazan, Russia

rishat.ibrahimov@yandex.ru, kamilhadi@gmail.com

<sup>2</sup> Center for Quantum Computer Science, University of Latvia, Rīga, Latvia  
{krisjanis.prusis, abuzer}@lu.lv

<sup>3</sup> Smart Quantum Technologies Ltd., Kazan, Russia

**Abstract.** We introduce the affine OBDD model and show that zero-error affine OBDDs can be exponentially narrower than bounded-error unitary and probabilistic OBDDs on certain problems. Moreover, we show that Las Vegas unitary and probabilistic OBDDs can be quadratically narrower than deterministic OBDDs. We also obtain the same results for the automata versions of these models.

**Keywords:** OBDDs · Affine models  
Quantum and probabilistic computation · Zero-error  
Las Vegas computation · Succinctness

## 1 Introduction

Using negative transition values (allowing interference between states and configurations) is unarguably the main distinguishing property of quantum computational models. In order to define a quantum-like classical system, one can also introduce “negative probabilities” but the system is no longer linear. In this direction, affine systems were introduced as an almost linear<sup>1</sup> generalization of probabilistic systems that can use negative transitions [10], and, due to their simplicity, affine finite automata (AfAs) have been examined in a series of papers by comparing them with classical and quantum finite automata [9, 10, 14, 26, 27, 29]. Both bounded- and unbounded-error AfAs have been shown to be more powerful than their probabilistic and quantum counterparts and they are equivalent to quantum models in nondeterministic acceptance mode [10]. AfAs can also be very succinct on languages and promise problems [9, 27]. In this paper, we

---

Part of the research work was done while Ibrahimov was visiting University of Latvia in February 2017.

<sup>1</sup> It evolves linearly but a non-linear operator is applied when we retrieve information from the state vector.

compare the computational power of error-free affine OBDDs with their probabilistic and unitary counterparts on solving total functions and recognizing languages.

Ordered Binary Decision Diagrams (OBDD), also known as oblivious read once branching programs [28], are an important restriction of branching programs. It is a good model of data streaming algorithms. Since the length of an OBDD is fixed (linear), the main complexity measure is “width”, analogous to the number of states for automata models (see [28]). OBDDs can also be seen as a variant of nonuniform automata that allow accessing the input in a predetermined order and using possibly different sets of instructions in each step (e.g. [1]). It is known that [3, 4, 6, 18, 23] the gap between the width of the following OBDD models can be at most exponential on total functions: deterministic and bounded-error probabilistic, deterministic and bounded-error unitary, and bounded-error probabilistic and bounded-error unitary. Each exponential gap has also been shown to be tight by presenting witness functions. On partial functions, on the other hand, the gap (width) between deterministic and exact unitary OBDDs was shown not to be bounded [2, 5, 12].

In this paper, we introduce affine OBDDs and then compare error-free (zero-error or Las Vegas) affine, unitary, probabilistic, and deterministic OBDD models. Zero-error probabilistic OBDDs are identical to deterministic OBDDs. Zero-error unitary OBDDs cannot be narrower than deterministic OBDDs. On the other hand, zero-error affine OBDDs can be exponentially narrower than not only deterministic OBDDs but also bounded-error probabilistic and unitary OBDDs. With success probability  $\frac{1}{2}$ , Las Vegas probabilistic and unitary OBDDs can be quadratically narrower than deterministic OBDDs. We give an example function that achieves this bound up to a logarithmic factor. Finally, we examine the automata counterpart of these models and obtain similar results.

The preliminaries and definitions are given in the next section. The generic lower bounds are given in Sect. 3. The results on zero-error affine OBDD are given in Sect. 4 and the results on Las Vegas OBDDs are given in Sect. 5. We close the paper with automata results in Sect. 6. The extended version of the paper can be accessible on arXiv (1703.07184).

## 2 Preliminaries

We assume the reader familiar with the basics of branching programs. An Ordered Binary Decision Diagram (OBDD) can be defined as a non-uniform automaton that can read the input symbols in a predetermined order (see [28]).

An OBDD  $P$  reads the variables in a specific order  $\pi = (j_1, \dots, j_n)$ , called the order of  $P$ , and it can trace its computation on a finite set of states  $S = \{s_1, \dots, s_m\}$  such that (i)  $m$  is the width of OBDD, (ii) the initial state is the initial node, and (iii) the accepting states are the accepting sink nodes. Thus, each node in any level can be easily associated with a state in  $S$ . An OBDD can also have a different transition function at each level.

A probabilistic OBDD (POBDD)  $P_n$  of width  $m$  is formally defined as a 5-tuple:  $P_n = (S, T, v_0, S_a, \pi)$ , where  $S = \{s_1, \dots, s_m\}$  is the set of states corresponding to at most one node in each level<sup>2</sup>,  $v_0$  is the initial probabilistic state, which is a stochastic column vector of size  $m$ ,  $S_a$  is the set of accepting states corresponding to the accepting sink nodes in the last level,  $\pi$  is a permutation of  $\{1, \dots, n\}$  defining the order of the input variables, and,  $T = \{T_i^0, T_i^1 \mid 1 \leq i \leq n\}$  is the set of (left) stochastic transition matrices.

Let  $x \in \{0, 1\}^n$  be the given input. At the beginning of the computation  $P_n$  is in  $v_0$ . Then the input bits are read in the order  $\pi(1), \pi(2), \dots, \pi(n)$  and the corresponding stochastic operators are applied. That is, at the  $j$ -th step, we have  $v_j = T_j^{x_{\pi(j)}} v_{j-1}$ , where  $v_{j-1}$  and  $v_j$  are the probabilistic states before and after the transition, respectively,  $\pi(j)$  represents the input bit read in this step,  $x_{\pi(j)}$  is the value of this bit, and  $T_j^{x_{\pi(j)}}$  is the stochastic operator applied in this step. We represent the final state as  $v_f = v_n$ . The input  $x$  is accepted (the value of 1 is returned) with probability  $f_{P_n}(x) = \sum_{s_i \in S_a} v_f[i]$ .

If all stochastic elements of transition matrices of a POBDD are composed of only 0s and 1s, then it is a deterministic OBDD.

Quantum OBDDs (QOBDD) using superoperators are non-trivial generalizations of POBDDs [2]. In this paper, we use the most restricted version of quantum OBDDs called unitary OBDDs (UOBDDs) [13]. Note that UOBDDs and POBDDs are incomparable [23]. (We refer the reader to [25] for a basic introduction to the basics of quantum computation.)

A UOBDD with width  $m$ , say  $M_n$ , is a 5-tuple  $M_n = (Q, T, |v_0\rangle, Q_a, \pi)$ , where  $Q = \{q_1, \dots, q_m\}$  is the set of states,  $|v_0\rangle$  is the initial quantum state,  $Q_a$  is the set of accepting states,  $\pi$  is a permutation of  $\{1, \dots, n\}$  defining the order of the variables, and  $T = \{T_i^0, T_i^1 \mid 1 \leq i \leq n\}$  is the set of unitary transition function matrices such that at the  $i$ -th step  $T_i^0$  ( $T_i^1$ ) is applied if the corresponding input bit is 0 (1).

Let  $x \in \{0, 1\}^n$  be the given input. At the beginning of the computation  $M_n$  is in  $|v_0\rangle$ . Then the input bits are read in the order  $\pi(1), \pi(2), \dots, \pi(n)$  and the corresponding unitary operators are applied:  $|v_j\rangle = T_j^{x_{\pi(j)}} |v_{j-1}\rangle$ . This represents the transition at the  $j$ -th step, where  $|v_{j-1}\rangle$  and  $|v_j\rangle$  are the quantum states before and after the transition, respectively,  $\pi(j)$  represents the input bit read in this step,  $x_{\pi(j)}$  is the value of this bit, and  $T_j^{x_{\pi(j)}}$  is the unitary operator applied in this step. We represent the final state as  $|v_f\rangle = |v_n\rangle$ . At the end of the computation, the final state is measured in the computational basis and the input is accepted if the observed state is an accepting one. Thus, the input  $x$  is accepted with probability  $f_{M_n}(x) = \sum_{q_i \in Q_a} |\langle q_i | v_f \rangle|^2$ , where  $|q_i\rangle$  represents the basis state corresponding to state  $q_i$  and  $\langle q_i | v_f \rangle$  gives the amplitude of  $q_i$  in the final state.

An  $m$ -state affine system [10] can be represented by the space  $\mathbb{R}^m$ , where  $\mathbb{R}$  is the set of real numbers. The set of (classical) states is denoted  $E = \{e_1, \dots, e_m\}$ . Any affine state (similar to a probabilistic state) is represented as a column vector

<sup>2</sup> Suppose we have  $w_i \leq m$  nodes on a level  $i$ ; then the node  $u_j$  of this level corresponds to the state  $s_j$ , for  $j \in \{1, \dots, w_i\}$ .

$v = (\alpha_1, \alpha_2, \dots, \alpha_m)^T \in \mathbb{R}^m$  such that  $\sum_{i=1}^m \alpha_i = 1$ . Each  $e_i$  also corresponds to a standard basis vector of  $\mathbb{R}^m$  having value 1 in its  $i$ -th entry. An affine operator is an  $m \times m$  matrix, each column of which is an affine state, where the  $(j, i)$ -th entry represents the transition from state  $e_i$  to state  $e_j$ . If we apply an affine operator  $A$  to the affine state  $v$ , we obtain the new affine state  $v' = Av$ . To get information from the affine state, a non-linear operator called weighting is applied, which returns any state with probability equal to the weight of the corresponding vector element in the  $l_1$  norm of the affine state. If it is applied to  $v$ , the state  $e_i$  is observed with probability  $\frac{|\alpha_i|}{\sum_{j=1}^m |\alpha_j|} = \frac{|\alpha_i|}{|v|}$ , where  $|v|$  is the  $l_1$  norm of  $v$ .

Here we define affine OBDDs (AfOBDDs) as a model with both classical and affine states, which is similar to the quantum model having quantum and classical states [7]. This addition does not change the computational power of the model, but helps in algorithm construction.

An AfOBDD, say  $M_n$ , having  $m_1$  classical and  $m_2$  affine states is an 9-tuple  $M_n = (S, E, \delta, T, s_I, v_0, S_a, E_a, \pi)$ , where  $S = \{s_1, \dots, s_{m_1}\}$  is the set of classical states,  $s_I \in S$  is the initial classical state,  $S_a \subseteq S$  is the set of classical accepting states,  $E = \{e_1, \dots, e_{m_2}\}$  is the set of affine states,  $v_0 \in \mathbb{R}^{m_2}$  is the initial affine state,  $E_a \subseteq E$  is the set of affine accepting states,  $\pi$  is a permutation of  $\{1, \dots, n\}$  defining the order of the variables,  $\delta = \{\delta_i : S \times \{0, 1\} \rightarrow S \mid 1 \leq i \leq n\}$  is the classical transition function such that at the  $i$ -th step the classical state is set to  $\delta_i(s, x_{\pi(i)})$  when in state  $s \in S$  and corresponding input bit is  $x_{\pi(i)}$ , and,  $T = \{T_i^{s,0}, T_i^{s,1} \mid s \in S \text{ and } 1 \leq i \leq n\}$  is the set of affine transition matrices such that at the  $i$ -th step  $T_i^{s,0}$  is applied if the corresponding input bit is 0 (or  $T_i^{s,1}$  if it is 1) and the current classical state is  $s$ . The width of  $M_n$  is equal to  $m_1 \cdot m_2$ .

Let  $x \in \{0, 1\}^n$  be the given input. At the beginning of the computation  $M_n$  is  $(s_I, v_0)$ . Then the input bits are read in the order  $\pi(1), \pi(2), \dots, \pi(n)$ . In each step, depending on the current input bit and classical state, the affine state is updated and then the classical state is updated based on the current input bit. Let  $(s, v_{j-1})$  be the classical-affine state pair at the beginning of the  $j$ -th step. Then the new affine state is updated as  $v_j = T_j^{s, x_{\pi(j)}} v_{j-1}$ . After that the new classical state is updated by  $\delta_j(s, x_{\pi(j)})$ . At the end of the computation we have  $(s_F, v_f)$ . If  $s_F \notin S_a$ , then the input is rejected. Otherwise, the weighting operator is applied to  $v_f$ , and the input is accepted with probability  $f_{M_n}(x) = \sum_{e_i \in E_a} \frac{|v_f[i]|}{|v_f|}$ . Note that if we use only non-negative numbers for an AfOBDD, then we obtain a POBDD.

Any OBDD with  $\pi = (1, \dots, n)$  is called an id-OBDD. If we use the same transitions at each level of an id-OBDD, then we obtain a finite automaton (FA). A FA can also read an additional symbol after reading the whole input called the right end-marker (\$) for the post-processing. We abbreviate the FA versions of OBDD, POBDD, QOBDD, UOBDD, and AfOBDD as DFA, PFA, QFA, UFA, and AfA, respectively. Remark that UFAs are also known as Measure-Once or Moore-Crutchfield quantum finite automata [8, 21].



A Las Vegas automaton can make three decisions: “accept”, “reject”, and “don’t know”. Therefore, its set of states is split into three disjoint sets, the set of accepting, rejecting, and neutral states in which the aforementioned decisions are given, respectively. To be a well-defined Las Vegas algorithm, each member (resp., non-member) is rejected (resp., member) with zero probability, i.e. the algorithm never makes false classification, and the correct decision is given with probability at least  $p > 0$ .

We assume the reader is familiar with the basic terminology of computing functions and recognizing languages. Here we revise some necessary notations.

A function  $f : \{0, 1\}^n \rightarrow \{1, 0\}$  is computed by a bounded-error machine if each member of  $f^{-1}(1)$  is accepted with probability at least  $1 - \varepsilon$  and each member of  $f^{-1}(0)$  is accepted with probability no more than  $\varepsilon$  for some non-negative  $\varepsilon < \frac{1}{2}$ . If  $\varepsilon = 0$ , then the computation (and the machine) is called zero-error or exact.

In the case of FAs, languages are considered instead of functions and the term “language recognition” is used instead of “computing a function”.

A Las Vegas FA can recognize a language  $L$  with success probability  $p < 1$  (with error bound  $1 - p$ ) if each member is accepted and each non-member is rejected with probability at least  $p$ .

For a given language  $L$ ,  $\text{DFA}(L)$ ,  $\text{LV}_\varepsilon(L)$ , and  $\text{ULV}_\varepsilon(L)$  denote the number of states of a minimal DFA, a minimal LV-PFA and a minimal LV-QFA recognizing language  $L$ , respectively, where the error bound is  $\varepsilon$  for the probabilistic and quantum models. For a given Boolean function  $f$ ,  $\text{OBDD}(f)$ ,  $\text{LV-OBDD}_\varepsilon(f)$ , and  $\text{ULV-OBDD}_\varepsilon(L)$  denote the widths of a narrowest OBDD, a narrowest LV-POBDD and a narrowest LV-UOBDDs computing  $f$ , respectively, where the error bound is  $\varepsilon$  for the probabilistic and quantum models. Remark that in the case of zero-error we set  $\varepsilon = 0$ .

### 3 Lower Bounds

Let  $X = \{X_1, \dots, X_n\}$  be the set of variables. Let  $\theta = (X_A, X_B)$  be a partition of the set  $X$  into two parts  $X_A$  and  $X_B = X \setminus X_A$ . Let  $f|_\rho$  be a “subfunction” of  $f$ , where  $\rho$  is a mapping  $\rho : X_A \rightarrow \{0, 1\}^{|X_A|}$ . The function  $f|_\rho(X_B)$  is obtained from  $f$  by fixing each variable from  $X_A$  to its value under  $\rho$ . The concept of subfunction can be seen as a counterpart of Myhill-Neroda equivalence classes. Let  $N^\theta(f)$  be the number of different subfunctions with respect to the partition  $\theta$ . Let  $\Pi(n)$  be the set of all permutations of  $\{1, \dots, n\}$ . Let  $\theta(\pi, u) = (X_A, X_B) = (\{X_{j_1}, \dots, X_{j_u}\}, \{X_{j_{u+1}}, \dots, X_{j_n}\})$ , for the permutation  $\pi = (j_1, \dots, j_n) \in \Pi(n)$ ,  $1 < u < n$ . We denote  $\Theta(\pi) = \{\theta(\pi, u) : 1 < u < n\}$ . Let  $N^\pi(f) = \max_{\theta \in \Theta(\pi)} N^\theta(f)$ ,  $N(f) = \min_{\pi \in \Pi(n)} N^\pi(f)$ .

Based on techniques from communication complexity theory, it has been shown that exact quantum and probabilistic protocols have at least the same complexity as deterministic ones [15, 19]. The followings are also known:

**Fact 1** [11, 15, 16, 19]. *For any regular language  $L$  and error bound  $\varepsilon < 1$ , we have the following lower bounds for PFAs and QFAs:  $(\text{DFA}(L))^{1-\varepsilon} \leq \text{LV}_\varepsilon(L)$  and  $(\text{DFA}(L))^{1-\varepsilon} \leq \text{ULV}_\varepsilon(L)$ .*

**Fact 2** [23]. *For any Boolean function  $f$  over  $X = (X_1, \dots, X_n)$  and error bound  $\varepsilon < 1$ :  $(\text{OBDD}(f))^{1-\varepsilon} \leq \text{ULV-OBDD}_\varepsilon(f)$ .*

These results are followed from certain facts from communication complexity. We briefly remind the notion of communication complexity (see [20]). Let  $h : \{0, 1\}^q \times \{0, 1\}^m \rightarrow \{0, 1\}$  be a Boolean function. We have two players called Alice and Bob, who compute  $h(x, y)$ . The function  $h$  is known by both of them, but Alice can see only  $x$  and Bob can see only  $y$ . First, Alice sends message(s) to Bob and then Bob returns the answer. Alice's aim is trying to minimize the number of sending bits. The protocol is called Las Vegas if Alice chooses her messages randomly, and then Bob returns the correct answer with probability at least  $1 - \varepsilon$  and gives-up (has error) with the remaining probability.

The communication complexity of function  $h(x, y)$  equals to  $c$  if and only if the minimum number of bits sent during the communication is equal to  $c$ . It is denoted as  $C(h)$  if the protocol is deterministic and  $\text{LV} - C_\varepsilon(h)$  if the protocol is Las Vegas and the probability of giving-up is bounded by  $\varepsilon$ . The relation between these two complexity measures is presented in the following fact:

**Fact 3** [11, 15, 16, 19]. *For any Boolean function  $h$  and an error bound  $\varepsilon < 1$ , we have the following lower bound:  $(1 - \varepsilon)C(h) \leq \text{LV} - C_\varepsilon(h)$ .*

Facts 1 and 2 follows from Fact 3 and a simulation of automata and OBDD by communication protocols (see [16, 17]).

We can easily extend the result from Fact 2 for the probabilistic OBDD model as well.

**Theorem 1.** *For any Boolean function  $f$  over  $X = \{X_1, \dots, X_n\}$  and error bound  $\varepsilon < 1$ :  $(\text{OBDD}(f))^{1-\varepsilon} \leq \text{LV-OBDD}_\varepsilon(f)$ .*

*Proof.* Let  $d = \text{OBDD}(f)$ . Due to [28], we have  $N(f) = d$ . Assume that there is a Las Vegas OBDD  $P$  of width  $w$ , i.e.  $w < d^{1-\varepsilon}$ . Let  $u = \text{argmax}_t N^{\theta(\pi(P), t)}(f)$  and  $\theta = \theta(\pi(P), u)$ . Then  $P$  can be simulated by a Las Vegas probabilistic protocol (see [17]) with  $\log_2 w < (1 - \varepsilon) \log_2 d$  bits. By the definition of the number of subfunctions, we have  $N^\theta(f) \geq d$ . Moreover, it is known that the deterministic communication complexity of a function is  $\log_2(N^\theta(f)) = \log_2 d$ . But we also have a Las Vegas communication protocol which uses  $\log_2 w < (1 - \varepsilon) \log_2 d$  communication bits, which contradicts with Fact 3.  $\square$

It is trivial that these results imply the equality for exact (zero-error) computation, where  $\varepsilon = 0$ .

### 4 Zero-Error Affine OBDDs

We show that exact AfOBDDs can be exponentially narrower than classical and unitary quantum OBDD models. For this purpose we use two different functions.

The hidden weighted bit function [28]  $\text{HWB}_n : \{0, 1\}^n \rightarrow \{0, 1\}$  returns the value of  $x_z$  on the input  $x = (x_1, \dots, x_n)$  where  $z = x_1 + \dots + x_n$ , taking  $x_0 = 0$ . It is known [28] that any OBDD solving  $\text{HWB}_n$  has a minimum width of  $2^{n/5}/n$ . Due to Fact 2 and Theorem 1, the same bound is also valid for exact POBDDs and UOBDDs.

**Theorem 2.** *An exact id-AfOBDD  $M$  with  $n$  classical and  $n$  affine states can solve  $\text{HWB}_n$ .*

*Proof.* The classical states are  $s_0, \dots, s_{n-1}$  where  $s_0$  is the initial and only accepting state. The affine states are  $e_0, \dots, e_{n-1}$  where  $e_0$  is the only accepting affine state and  $v_0$  is  $e_0$ .

Until reading the last input bit ( $x_n$ ), for each value 1, the index of the classical state is increased by 1. Meanwhile, the value of  $x_i$  is written to the value of  $e_i$  in the affine state: the affine state after the  $(i - 1)$ -th step becomes  $v_{i-1} = (\bar{1} \ x_1 \ \dots \ x_{i-1} \ 0 \ \dots \ 0)^T$ , where  $\bar{1} = 1 - \sum_{j=1}^{i-1} x_j$ . If  $x_i = 0$ , then the identity operator is applied. If  $x_i = 1$ , then the following affine transformation is applied

$$\left( \begin{array}{cccc|ccc} 0 & -1 & \dots & -1 & 0 & & \\ 0 & & & & & & 0 \\ \vdots & \mathbf{I}_{(i-1) \times (i-1)} & & & \vdots & & \mathbf{0} \\ 0 & & & & & & 0 \\ \hline 1 & 1 & \dots & 1 & 1 & & \\ \hline & & & \mathbf{0} & & & \mathbf{I} \end{array} \right),$$

which updates the values of  $e_0$  and  $e_i$  as  $-\sum_{j=1}^{i-1} x_j$  and 1, respectively, and does not change the other entries. Remark that the first entry can also be written as  $1 - \sum_{j=1}^i x_j$ . Thus, before reading  $x_n$ , the classical state is  $s_t$  for  $t = x_1 + \dots + x_{n-1}$ , and, the affine state is  $v_{n-1} = (\bar{1} \ x_1 \ x_2 \ \dots \ x_{n-1})^T$ , where  $\bar{1}$  is  $1 - \sum_{j=1}^{n-1} x_j$ .

After reading  $x_n$ , the classical state is always set to  $s_0$  and so the decision is made based on the affine state. Each pair of  $s_t$  and  $x_n$  determine an appropriate last affine transformation that sets the final affine state to  $v_f = (x_z \ 1 - x_z \ 0 \ \dots \ 0)^T$ , where  $x_z$  is set to either 0 or 1. The details of each last transition are as follows: (i) if  $t = 0$  and  $x_n = 0$ , then the corresponding last transitions sets  $x_z = 0$ , (ii) if  $t = n - 1 > 0$ , then regardless the value of  $x_n$ , the corresponding last transition sets  $x_z = 1$ , and (iii) in any other case,  $z = t + x_n$  and the corresponding last transition sets  $x_z$  to the  $(z + 1)$ -th entry of  $v_{n-1}$ . Based on  $x_z$ , the final decision is given: if  $x_z = 0$  (resp.,  $x_z = 1$ ), then the input is accepted with probability 0 (resp., 1).  $\square$

For a positive integer  $n$ , let  $x, y$  be the inputs of size  $n$ , let  $p(n)$  be the smallest prime number greater than  $n$ , and let  $s_n(x) = (\sum_{i=1}^n i \cdot x_i) \bmod p(n)$ . The mixed weighted sum function [22]  $MWS_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  is defined as  $MWS_n(x, y) = x_i \oplus y_i$  if  $i = s_n(x) = s_n(y) \in \{1, \dots, n\}$ , and 0 otherwise. Any bounded-error POBDD or UOBDD solving  $MWS_n$  has a width of at least  $2^{\Omega(n)}/n$  [22, 24]. The same bound is also valid for OBDDs, and so for exact POBDDs and UOBDDs as well.

Before presenting our affine algorithm for  $MWS_n$ , we consider its simpler version: the weighted sum function [22]  $WS_n(x) : \{0, 1\}^n \rightarrow \{0, 1\}$ , defined as  $WS_n(x) = x_{s_n(x)}$  if  $s_n(x) \in \{1, \dots, n\}$ , and 0 otherwise.

**Theorem 3.** *An exact id-AfOBDD can solve  $WS_n$  with  $p(n)$  classical states and  $n$  affine states.*

*Proof.* We use almost the same algorithm given in the previous proof. The affine part is the same. The new classical states are  $s_0, \dots, s_{p(n)-1}$  with the same initial and single accepting state  $s_0$ . Until reading  $x_n$ , the same affine transitions are applied:  $v_{n-1} = (\bar{1} \ x_1 \ x_2 \ \dots \ x_{n-1})^T$ , where  $\bar{1}$  is  $1 - \sum_{j=1}^{n-1} x_j$ . The classical transitions are modified as follows: The classical state before reading  $x_i$  ( $i < n$ ), say  $s_j$ , is set to  $s_{j+i \cdot x_i \bmod p(n)}$ . Thus, before reading  $x_n$ , the classical state is  $s_t$  where  $t = (\sum_{i=1}^{n-1} i \cdot x_i) \bmod p(n)$ . At this point, the pair  $s_t$  and  $x_n$  sets the affine state to  $v_f = (x_{s_n(x)} \ 1 - x_{s_n(x)} \ 0 \ \dots \ 0)^T$  and then the classical state is set to  $s_0$ . Here  $x_{s_n(x)} = 0$  if  $t + n \cdot x_n \bmod p(n) \notin \{1, \dots, n\}$ . Otherwise, depending on the value of  $s_n(x)$ ,  $x_{s_n(x)}$  is set to the corresponding value from  $v_{n-1}$  or directly to  $x_n$ . □

**Theorem 4.** *An exact id-AfOBDD can solve  $MWS_n$  with  $p^2(n)$  classical states and  $(n + 1)$  affine states.*

*Proof.* The classical states are  $\{s_{i,j} \mid 0 \leq i, j < p(n)\}$  where  $s_{0,0}$  is the initial state and the accepting states are  $\{s_{i,i} \mid 1 \leq i \leq n\}$ . While reading all bits, the values of  $s_n(x)$  and  $s_n(y)$  are calculated and stored as the values of the first and second index of the classical states, respectively. Let  $s_{i,j}$  be the final state. It is clear that if  $i \neq j$  or  $i = j$  but not in  $\{1, \dots, n\}$ , the input is rejected classically. In the remaining part, we assume that the final classical state is  $s_{i,i}$  with  $i \in \{1, \dots, n\}$ . Thus, the final decision is given based on the final affine state.

The affine states are  $\{e_0, \dots, e_n\}$  where  $e_0$  is the single accepting state. The initial affine state is  $v_0 = e_0$ .

While reading the first part of the input, all  $x$  values are encoded in the affine state as  $(\bar{1} \ (-1)^{x_1} \ (-1)^{x_2} \ \dots \ (-1)^{x_n})^T$ , where  $\bar{1} = 1 - \sum_{i=1}^n (-1)^{x_i}$ . Then, for each  $y_j$  ( $1 \leq j \leq n$ ), we multiply the  $j$ -th entry of the affine state by  $(-1)^{y_j}$  (and update the first entry accordingly). Thus, the  $j$ -th entry becomes  $(-1)^{x_j + y_j}$ , which is equal to 1 if  $x_j = y_j$  and  $-1$  if  $x_j \neq y_j$ .

The last affine transformation is a composition of three affine transformations. The first transformation is the one explained above for  $y_n$ . By using the second one, the affine state is set to  $((-1)^{x_i + y_i} \ 1 - (-1)^{x_i + y_i} \ 0 \ \dots \ 0)^T$ .

Here the first two entries are  $(1 \ 0)^T$  for members and  $(-1 \ 2)^T$  for non-members. By using the third transformation, we can add half of the second entry to the first entry and so get respectively  $(1 \ 0)^T$  and  $(0 \ 1)^T$ . Thus the AfOBDD can separate members from non-members with zero error.  $\square$

## 5 Las Vegas POBDDs and UOBDDs

For OBDDs with  $\varepsilon = \frac{1}{2}$ , the lower bounds given in Sect. 3 can be at most quadratic. Up to a logarithmic factor, this quadratic gap was achieved by using the  $\text{SA}_d$  function in [23] for id-OBDDs. Here we give the same result for OBDDs (for any order) using the  $\text{SSA}_n$  function and also provide an LV-UOBDD algorithm with the same size as the LV-OBDD.

We start with the well-known *Storage Access Boolean Function*  $\text{SA}_d(x, y) = x_y$ , where the input is split into the *storage*  $x = (x_1, \dots, x_{2^d})$  and the *address*  $y = (y_1, \dots, y_d)$ .

By using the idea of “Shuffling” from [2, 3, 5, 6], we define the *Shuffled Storage Access Boolean function*  $\text{SSA}_n : \{0, 1\}^n \rightarrow \{0, 1\}$  for even  $n$ . Let  $x \in \{0, 1\}^n$  be an input. We form two disjoint sorted lists of even indexes of bits  $I_0 = (2i_1, \dots, 2i_m)$  and  $I_1 = (2j_1, \dots, 2j_k)$  with the following properties: (i) if  $x_{2i-1} = 0$  then  $2i \in I_0(x)$ , (ii) if  $x_{2i-1} = 1$  then  $2i \in I_1(x)$ , (iii)  $i_r < i_{r+1}$ , for  $r \in \{1, \dots, m-1\}$  and  $j_r < j_{r+1}$ , for  $r \in \{1, \dots, k-1\}$ .

Let  $d$  be such that  $2^d + d = n/2$ . We can construct two binary strings by the following procedure: initially  $\alpha(x) := 0^{2^d}$ , then for  $r$  from 1 to  $m$  we do  $\alpha(x) := \text{ShiftX}(\alpha(x), x_{2i_r})$ , where  $\text{ShiftX}((a_1, \dots, a_m), b) = (a_2, \dots, a_m, a_1 \oplus b)$ . And initially  $\beta(x) := 0^d$ , then for  $r$  from 1 to  $k$ :  $\beta(x) := \text{ShiftX}(\beta(x), x_{2j_r})$ . Then,  $\text{SSA}_n(x) = \text{SA}_d(\alpha(x), \beta(x))$ . Firstly, we provide a lower bound for OBDDs.

**Theorem 5.**  $\text{OBDD}(\text{SSA}_n) \geq 2^{2^d}$ , for  $2^d + d = n/2$ .

Now, we provide an upper bound for LV-OBDDs.

**Theorem 6.**  $\text{LV-OBDD}_{0.5}(\text{SSA}_n) \leq 2^{2^d/2+d+3}$ , for  $2^d + d = n/2$ .

**Theorem 7.**  $\text{ULV-OBDD}_{0.5}(\text{SSA}_n) \leq 2^{2^d/2+d+3}$ , for  $2^d + d = n/2$ .

Affine OBDDs can be exponentially narrower also for  $\text{SSA}_n$ .

**Theorem 8.** *An exact AfOBDD A can solve  $\text{SSA}_n$  with  $2^{d+1}$  classical states and  $2^d + 1$  affine states, for  $2^d + d = n/2$ .*

*Proof.* The set of classical states is  $\{(p, s) \mid p \in \{p_0, p_1\} \text{ and } s \in \{s_0, \dots, s_{2^d-1}\}\}$  having  $2^{d+1}$  states. The states  $p_j$  denote whether the next even bit to read is a storage or address bit. The state  $s_i$  corresponds to the current address being  $i$ . Every classical state is accepting, and so the decision is made based on the final affine state. The AfOBDD also has  $2^d + 1$  affine states,  $\{e_1, \dots, e_{2^d+1}\}$ . The initial state is  $e_{2^d+1}$ , i.e.  $v_0 = (0 \ \dots \ 0 \ 1)^T$ , and the only accepting state is  $e_1$ .

During the computation, it keeps the value of the storage in the first  $2^d$  states. Specifically, if (i) the next position to read is odd or (ii) even but an address bit (we are in a state  $(p_1, s_i)$ ), then we perform the identity transformation on the affine state and change only the classical state. If we are reading a storage bit, the classical state remains unchanged and we implement the **ShiftX** operation on the storage: first, the  $2^d$  entries are shifted to the left by one and the first entry becomes the  $2^d$ -th entry. Then, depending on the scanned symbol, the value of the  $2^d$ -th entry is updated:

- If the scanned symbol is 0, then, for calculating the *XOR* value, the  $2^d$ -th entry is multiplied by 1, i.e.,  $0 \rightarrow 0$  and  $1 \rightarrow 1$ .
- If the scanned symbol is 1, then, for calculating the *XOR* value, the  $2^d$ -th entry is multiplied by  $-1$  and then 1 is added to this result, i.e.,  $0 \rightarrow 0 \rightarrow 1$  and  $1 \rightarrow -1 \rightarrow 0$ .

The last entry in the affine state is used to make the state vector well-formed. For example, if the state vector has  $0 \leq t \leq 2^d$  1s in its first  $2^d$  entries, then the last entry is  $1 - t$ .

After reading the whole input, the first  $2^d$  entries keep the storage. We know the address  $i$  from our classical state  $s_i$  – we move the corresponding storage value from  $e_i$  to  $e_1$  and sum all other entries in  $e_2$ . Then, if the first entry is 1, the rest of the vector contains only 0. If the first entry is 0, the second entry is 1 and the rest of the vector contains 0. Therefore, any member (resp., non-member) of  $\text{SSA}_n$  is accepted by  $A$  with probability 1 (resp., 0).  $\square$

## 6 Las Vegas Automata and Zero-Error AfAs

Similar to OBDDs, for  $\varepsilon = \frac{1}{2}$ , the lower bound for finite automata (Sect. 3) is at most quadratic. Up to a constant, this quadratic gap is achieved by  $\text{END}_k = \{u1v \mid u, v \in \{0, 1\}^* \text{ and } |v| = k-1\}$ :  $\text{DFA}(\text{END}_k) = 2^k$  and  $\text{LV}_{0.5}(\text{END}_k) \leq 4 \cdot 2^{k/2}$ .

Here, we propose a new language  $\text{MODXOR}_k$  based on which we improve the above constant for LV-PFAs and provide a LV-UFA algorithm with the same size as LV-PFAs. Then, we show that an AfA can recognize it with exponentially fewer states with zero error. The language  $\text{MODXOR}_k$  for  $k > 0$  is formed by the strings  $\{0, 1\}^{<2k} x_1 \{0, 1\}^{2k-1} x_2 \{0, 1\}^{2k-1} \dots x_m \{0, 1\}^{2k-1}$  where  $m > 0$ , each  $x_i \in \{0, 1\}$  for  $1 \leq i \leq m$ , and  $\bigoplus_{i=0}^m x_i = 1$ , taking  $x_0 = 0$ . First, we give a lower bound for DFAs.

**Theorem 9.**  $\text{DFA}(\text{MODXOR}_k) \geq 2^{2k}$  for each  $k > 0$ .

**Theorem 10.**  $\text{LV}_{0.5}(\text{MODXOR}_k) \leq 2 \cdot 2^k$  for any  $k > 0$ .

**Theorem 11.**  $\text{ULV}_{0.5}(\text{MODXOR}_k) \leq 2 \cdot 2^k$  for any  $k > 0$ .

Similarly to OBDDs, exact AfAs can also be exponentially more efficient than their classical and quantum counterparts.

**Theorem 12.** *The language  $\text{MODXOR}_k$  for  $k > 0$  can be recognized by a  $(2k + 1)$ -state AfA  $A$  with zero-error.*

*Proof.* The AfA  $A$  does not use any classical state and it has  $2k + 1$  affine states,  $\{e_1, \dots, e_{2k+1}\}$ . The initial state is  $e_{2k+1}$  and the only accepting state is  $e_1$ . It starts its computation in  $v_0 = (0 \dots 0 1)^T$ .

During the computation, it keeps the results in the values of the first  $2k$  states, i.e. it sets each value to 0 or 1 depending the previous results and the current scanning symbols. More specifically, before each transition, the first  $2k$  entries are shifted to the right by one and the  $2k$ -th entry becomes the first entry. Then, depending on the scanned symbol, the value of the first entry is updated:

- If the scanned symbol is 0, then, for calculating  $XOR$  value, the first entry is multiplied by 1, i.e.,  $0 \rightarrow 0$  and  $1 \rightarrow 1$ .
- If the scanned symbol is 1, then, for calculating  $XOR$  value, the first entry is multiplied by  $-1$  and then 1 is added to this result, i.e.,  $0 \rightarrow 0 \rightarrow 1$  and  $1 \rightarrow -1 \rightarrow 0$ .

The last entry in the affine state is used to make the state vector well-formed. For example, if the state vector has  $0 \leq t \leq 2k$  1s in its first  $2k$  entries, then the last entry is  $1 - t$ . After reading the whole input, the first entry has the result and the rest of entries are summed to the second entry: if the first entry is 1 (resp., 0), then the rest of the vector contains only 0 (resp., 1). Therefore, any member (resp., non-member) is accepted by  $A$  with probability 1 (resp., 0).  $\square$

**Acknowledgements.** We thank Evgenijs Vihrovs (University of Latvia) for his helpful discussions and anonymous reviewers for their very helpful comments.

The work is partially supported by ERC Advanced Grant MQC, Latvian State Research Programme NeXIT project No. 1. The work is also performed according to the Russian Government Program of Competitive Growth of Kazan Federal University. The research on Las-Vegas OBDDs (Sect. 5) is supported by Russian Science Foundation Grant 17-71-10152

## References

1. Ablayev, F., Gainutdinova, A.: Complexity of quantum uniform and nonuniform automata. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 78–87. Springer, Heidelberg (2005). <https://doi.org/10.1007/11505877-7>
2. Ablayev, F., Gainutdinova, A., Khadiev, K., Yakaryılmaz, A.: Very narrow quantum OBDDs and width hierarchies for classical OBDDs. *Lobachevskii J. Math.* **37**(6), 670–682 (2016)
3. Ablayev, F.: Randomization and nondeterminism are incomparable for ordered read-once branching programs. In: *ECCC (021)* (1997)
4. Ablayev, F., Gainutdinova, A., Karpinski, M., Moore, C., Pollett, C.: On the computational power of probabilistic and quantum branching program. *Inf. Comput.* **203**(2), 145–162 (2005)

5. Ablayev, F., Gainutdinova, A., Khadiev, K., Yakaryılmaz, A.: Very narrow quantum OBDDs and width hierarchies for classical OBDDs. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) DCFS 2014. LNCS, vol. 8614, pp. 53–64. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-09704-6\\_6](https://doi.org/10.1007/978-3-319-09704-6_6)
6. Ablayev, F., Karpinski, M.: On the power of randomized branching programs. In: Meyer, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 348–356. Springer, Heidelberg (1996). [https://doi.org/10.1007/3-540-61440-0\\_141](https://doi.org/10.1007/3-540-61440-0_141)
7. Ambainis, A., Watrous, J.: Two-way finite automata with quantum and classical states. *Theor. Comput. Sci.* **287**(1), 299–311 (2002)
8. Ambainis, A., Yakaryılmaz, A.: Automata and quantum computing. Technical report 1507.01988, arXiv (2015)
9. Belovs, A., Montoya, J.A., Yakaryılmaz, A.: On a conjecture by Christian Choffrut. *Int. J. Found. Comput. Sci.* **28**(5), 483–502 (2017)
10. Díaz-Caro, A., Yakaryılmaz, A.: Affine computation and affine automaton. In: Kulikov, A.S., Woeginger, G.J. (eds.) CSR 2016. LNCS, vol. 9691, pp. 146–160. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-34171-2\\_11](https://doi.org/10.1007/978-3-319-34171-2_11)
11. Āuriš, P., Hromkovič, J., Rolim, J.D.P., Schnitger, G.: Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 117–128. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0023453>
12. Gainutdinova, A.F.: Comparative complexity of quantum and classical OBDDs for total and partial functions. *Russ. Math.* **59**(11), 26–35 (2015)
13. Gainutdinova, A., Yakaryılmaz, A.: Nondeterministic unitary OBDDs. In: Weil, P. (ed.) CSR 2017. LNCS, vol. 10304, pp. 126–140. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-58747-9\\_13](https://doi.org/10.1007/978-3-319-58747-9_13)
14. Hirvensalo, M., Moutot, E., Yakaryılmaz, A.: On the computational power of affine automata. In: Drewes, F., Martín-Vide, C., Truthe, B. (eds.) LATA 2017. LNCS, vol. 10168, pp. 405–417. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-53733-7\\_30](https://doi.org/10.1007/978-3-319-53733-7_30)
15. Hirvensalo, M., Seibert, S.: Lower bounds for Las Vegas automata by information theory. *RAIRO Theor. Inform. Appl.* **37**(1), 39–49 (2003)
16. Hromkovič, J., Schnitger, G.: On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. *Inf. Comput.* **169**(2), 284–296 (2001)
17. Khadiev, K.: On the hierarchies for deterministic, nondeterministic and probabilistic ordered read-k-times branching programs. *Lobachevskii J. Math.* **37**(6), 682–703 (2016)
18. Khadiev, K., Khadieva, A.: Reordering method and hierarchies for quantum and classical ordered binary decision diagrams. In: Weil, P. (ed.) CSR 2017. LNCS, vol. 10304, pp. 162–175. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-58747-9\\_16](https://doi.org/10.1007/978-3-319-58747-9_16)
19. Klauck, H.: On quantum and probabilistic communication: Las Vegas and one-way protocols. In: STOC 2000, pp. 644–651 (2000)
20. Kushilevitz, E., Nisan, N.: *Communication Complexity*. Cambridge University Press, New York (1997)
21. Moore, C., Crutchfield, J.P.: Quantum automata and quantum grammars. *Theor. Comput. Sci.* **237**(1–2), 275–306 (2000)
22. Sauerhoff, M.: Quantum vs. classical read-once branching programs. In: *Complexity of Boolean Functions*, vol. 06111, Dagstuhl Seminar Proceedings, Internationales Begegnungs und Forschungszentrum für Informatik (2006)



23. Sauerhoff, M., Sieling, D.: Quantum branching programs and space-bounded nonuniform quantum complexity. *Theor. Comput. Sci.* **334**(1), 177–225 (2005)
24. Savický, P., Žák, S.: A read-once lower bound and a  $(1, + k)$ -hierarchy for branching programs. *Theor. Comput. Sci.* **238**(1), 347–362 (2000)
25. Say, A.C.C., Yakaryılmaz, A.: Quantum finite automata: a modern introduction. In: Calude, C.S., Freivalds, R., Kazuo, I. (eds.) *Computing with New Resources*. LNCS, vol. 8808, pp. 208–222. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-13350-8\\_16](https://doi.org/10.1007/978-3-319-13350-8_16)
26. Villagra, M., Yakaryılmaz, A.: Language recognition power and succinctness of affine automata. *Nat. Comput.* **17**(2), 283–293 (2018). <https://doi.org/10.1007/s11047-017-9652-z>
27. Villagra, M., Yakaryılmaz, A.: Language recognition power and succinctness of affine automata. In: Amos, M., Condon, A. (eds.) *UCNC 2016*. LNCS, vol. 9726, pp. 116–129. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-41312-9\\_10](https://doi.org/10.1007/978-3-319-41312-9_10)
28. Wegener, I.: *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM, Philadelphia (2000)
29. Nakanishi, M., Khadiev, K., Prusis, K., Vihrovs, J., Yakaryılmaz, A.: Exact affine counter automata. *Electron. Proc. Theoret. Comput. Sci. (EPTCS)* **252**, 205–218 (2017). <https://doi.org/10.4204/EPTCS.252.20>



# State Complexity of Unambiguous Operations on Deterministic Finite Automata

Galina Jirásková<sup>1</sup> and Alexander Okhotin<sup>2</sup>(✉)

<sup>1</sup> Mathematical Institute, Slovak Academy of Sciences,  
Grešáková 6, 040 01 Košice, Slovak Republic  
[jiraskov@saske.sk](mailto:jiraskov@saske.sk)

<sup>2</sup> St. Petersburg State University,  
7/9 Universitetskaya nab., Saint Petersburg 199034, Russia  
[alexander.okhotin@spbu.ru](mailto:alexander.okhotin@spbu.ru)

**Abstract.** The paper determines the number of states in a deterministic finite automaton (DFA) necessary to represent “unambiguous” variants of the union, concatenation, and Kleene star operations on formal languages. For the disjoint union of languages represented by an  $m$ -state and an  $n$ -state DFA, the state complexity is  $mn - 1$ ; for the unambiguous concatenation, it is known to be  $m2^{n-1} - 2^{n-2}$  (Daley et al. “Orthogonal concatenation: Language equations and state complexity”, *J. UCS*, 2010), and this paper shows that this number of states is necessary already over a binary alphabet; for the unambiguous star, the state complexity function is determined to be  $\frac{3}{8}2^n + 1$ . In the case of a unary alphabet, disjoint union requires up to  $\frac{1}{2}mn$  states, unambiguous concatenation has state complexity  $m + n - 2$ , and unambiguous star requires  $n - 2$  states in the worst case.

## 1 Introduction

The basic operations on formal languages are union, concatenation and Kleene star. The main models for language description, namely, regular expressions and formal grammars, use these operations to express the structure of strings. An important special case of concatenation is *unambiguous concatenation*. Concatenation of two formal languages,  $K$  and  $L$ , is said to be *unambiguous*, if every string  $w$  in  $KL$  has a unique representation as  $w = uv$ , with  $u \in K$  and  $v \in L$ . The union operation also has its unambiguous special case: the *disjoint union*. These two operations are important, in particular, for giving rise to *unambiguous grammars*. One can also define the *unambiguous Kleene star*: the Kleene star of any language  $L$  with the property that every string in  $L^*$  has a unique decomposition as a concatenation of zero or more strings in  $L$ .

---

G. Jirásková—Research supported by VEGA grant 2/0084/15 and grant APVV-15-0091.

This paper investigates the succinctness of description of these three *unambiguous operations* by deterministic finite automata (DFA). The state complexity of union, concatenation and Kleene star in their unrestricted form has long been known: Maslov [17] was the first to determine their state complexity as  $mn$  states for the union,  $m2^n - 2^{n-1}$  states for the concatenation and  $\frac{3}{4}2^n$  states for the Kleene star, where  $m$  and  $n$  is the number of states in the DFA recognizing the arguments. These results were further elaborated by Yu et al. [21] and by Jirásek et al. [10]. For nondeterministic finite automata (NFA), the state complexity of these operations was determined by Holzer and Kutrib [9], a similar study for two-way finite automata (2DFA) was carried out by Jirásková and Okhotin [14], whereas Jirásek Jr. et al. [11] determined the state complexity of basic operations for unambiguous finite automata (UFA). In the literature, special consideration was given to the case of the unary alphabet, where state complexity results are different: for DFA, as established by Yu et al. [21] and by Pighizzini and Shallit [19], both union and concatenation require up to  $mn$  states for relatively prime  $m, n$ , and fewer states for other values of  $m, n$ ; the star requires  $(n - 1)^2 + 1$  states. The state complexity of basic operations for unary two-way automata was established by Kunc and Okhotin [15, 16], and by Okhotin [18] for unary UFA.

The question addressed in this paper is, do these state complexity results essentially depend on using ambiguity, that is, on taking a union of overlapping languages, on concatenating languages that allow some strings to be obtained in multiple ways, and on taking the star of languages with multiple partitions? If union, concatenation or star is restricted to be unambiguous, will it become substantially easier to express?

In the case of the **disjoint union** operation, investigated in Sect. 3, it turns out that it can be represented with one state less than the union in general: that is,  $mn - 1$  states are sufficient to represent disjoint union of a pair of DFA with  $m$  and with  $n$  states. It is also proved that this number of states is in the worst case necessary, with witness languages defined over a binary alphabet. For the unary alphabet, the state complexity is  $\frac{1}{2}mn$ , under certain assumptions on  $m$  and  $n$ .

For the **unambiguous concatenation**, it is already known from Daley et al. [4] that it requires exactly half as many states as concatenation in general: its state complexity is  $m2^{n-1} - 2^{n-2}$ . The witness languages of Daley et al. [4] are defined over a four-symbol alphabet. In this paper, new binary witness languages for this operation are constructed in Sect. 4. In the unary case, the state complexity of this operation is only  $m + n - 2$ .

Finally, as established in Sect. 5, representing the **unambiguous star** takes half as many states as star in general, plus one extra state: its state complexity is  $\frac{3}{8}2^n + 1$ , with witness languages over a binary alphabet. In the unary case, its state complexity is  $n - 2$ .

## 2 Basic Notions

As usual, a *deterministic finite automaton* (DFA) is defined as a quintuple  $A = (\Sigma, Q, q_0, \delta, F)$  where  $\Sigma$  is an input alphabet;  $Q$  is a finite non-empty set of states;  $q_0 \in Q$  is the initial state;  $\delta: Q \times \Sigma \rightarrow Q$  is the transition function;  $F \subseteq Q$  is the set of accepting states. The computation of  $A$  on a string  $w = a_1 \dots a_n$ , with  $a_1, \dots, a_n \in \Sigma$ , is the uniquely defined sequence of states  $r_0, \dots, r_n \in Q$ , in which  $r_0 = q_0$ ,  $r_i = \delta(r_{i-1}, a_i)$  for all  $i$ . If  $r_n \in F$ , the DFA is said to *accept* the string  $w$ . The language recognized by a DFA, denoted by  $L(A)$ , is the set of all strings it accepts.

It is convenient to extend the transition function to act on strings in  $\Sigma^*$ , as  $\delta: Q \times \Sigma^* \rightarrow Q$ , with  $\delta(q, \varepsilon) = q$  and  $\delta(q, aw) = \delta(\delta(q, a), w)$ . In these terms,  $L(A) = \{w \mid \delta(q_0, w) \in F\}$ . Also, the function is sometimes extended to act on sets of states, as  $\delta(S, w) = \{\delta(q, w) \mid q \in S\}$ .

A state  $q$  of a DFA is called *dead*, if no string is accepted from  $q$ . In a minimal DFA, there can be at most one dead state. A state  $q$  of a DFA is called *cyclic* if there is a non-empty string  $v$  with  $\delta(q, v) = q$ .

A DFA over a unary alphabet  $\Sigma = \{a\}$  is a chain of transitions by  $a$  that eventually turns back to one of the previous states. The repetitive part is called the *cycle*, and the earlier states outside of the cycle form the *tail*. Thus, the language is *ultimately periodic*, beginning from  $\ell$  (the length of the tail), with the length of the cycle as the period.

A *nondeterministic finite automaton* (NFA) is a quintuple  $A = (\Sigma, Q, Q_0, \delta, F)$ , where  $Q_0 \subseteq Q$  is the set of possible initial states and the transition function  $\delta: Q \times \Sigma \rightarrow 2^Q$  may define multiple next states. An NFA accepts a string  $w = a_1 \dots a_n$  if there exists a sequence of states  $r_0, \dots, r_n \in Q$ , with  $r_0 \in Q_0$ ,  $r_i \in \delta(r_{i-1}, a_i)$  for all  $i$ , and  $r_n \in F$ .

In some literature, NFAs are defined with a unique initial state, that is, with  $Q_0 = \{q_0\}$ . Every NFA can be converted to an NFA with a unique initial state by adding a new initial state.

An NFA  $A = (\Sigma, Q, Q_0, \delta, F)$  can be transformed to an equivalent DFA with states corresponding to subsets of  $Q$ . The subset  $Q_0$  is then the initial state of the DFA, its set of final states is  $\{S \subseteq Q \mid S \cap F \neq \emptyset\}$ , and its transition function  $\delta': 2^Q \times \Sigma \rightarrow 2^Q$  is defined by  $\delta'(S, a) = \bigcup_{q \in S} \delta(q, a)$ .

## 3 Union

In general, the union of an  $m$ -state DFA  $A = (\Sigma, P, p_0, \gamma, E)$  and an  $n$ -state DFA  $B = (\Sigma, Q, q_0, \delta, F)$  can be recognized by a DFA with  $mn$  states, known as the *direct product DFA*. Its set of states is the set of all pairs  $P \times Q$ , the initial state is  $(p_0, q_0)$ , the transition function simulates  $A$  on the first component and  $B$  on the second component, thus mapping  $(p, q)$  by  $a$  to  $(\gamma(p, a), \delta(q, a))$ ; and a pair  $(p, q)$  is accepting if  $p \in E$  or  $q \in F$ .

For automata without the disjointness restriction,  $mn$  states are in the worst case necessary [17]. It turns out that if the union is disjoint, then the construction can be improved by one state.

**Lemma 1.** For every  $m$ -state DFA  $A$  and for every  $n$ -state DFA  $B$ , with  $m, n \geq 2$  and with  $L(A) \cap L(B) = \emptyset$ , there is a DFA with  $mn - 1$  states that recognizes the disjoint union  $L(A) \cup L(B)$ .

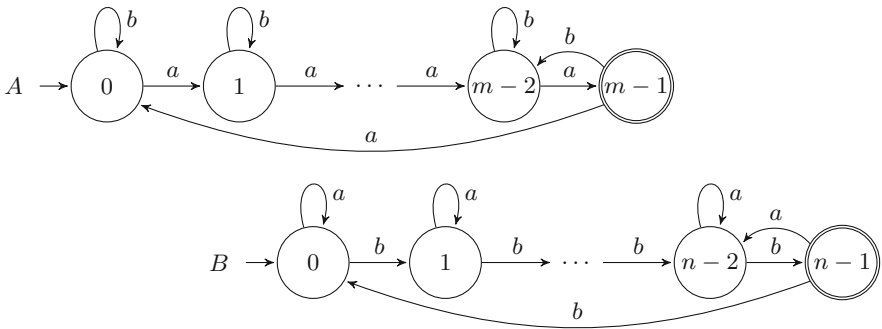
Indeed, if any pair  $(p, q)$ , with  $p$  accepting in  $A$  and  $q$  accepting in  $B$ , were reachable in the direct product automaton, then the union would not be disjoint. Thus, all such pairs can be excluded from the  $mn$ -state construction. Putting aside the trivial case of either automaton having no accepting states, at least one such pair exists.

It remains to show that this number of states,  $mn - 1$ , is necessary in the worst case.

**Lemma 2.** For all  $m, n \geq 2$ , there exist languages  $K, L \subseteq \{a, b\}^*$ , recognized by an  $m$ -state and an  $n$ -state DFA, respectively, with  $K \cap L = \emptyset$ , for which every DFA recognizing  $K \cup L$  has at least  $mn - 1$  states.

*Proof (a sketch).* The desired witness languages are recognized by the pair of automata shown in Fig. 1. Every string in  $K$  ends with  $a$ , while every string in  $L$  ends with  $b$ , and therefore,  $K$  and  $L$  are disjoint. In the direct product automaton, each pair  $(i, j)$  is reachable by  $a^i b^j$  if  $i \leq m - 2$ , and by  $a^{m-1} b^j a$  for  $i = m - 2$ .

For every two pairs  $(i, j)$  and  $(k, \ell)$ , if  $i < k$ , then the string  $a^{m-1-k}$  is accepted from  $(i, j)$ , but not from  $(k, \ell)$ . The case of  $j < \ell$  is symmetric.  $\square$



**Fig. 1.** Binary witnesses for disjoint union meeting the upper bound  $mn - 1$ .

The next theorem summarizes the results of the two lemmata above.

**Theorem 3 (Disjoint Union).** Let  $A$  and  $B$  be an  $m$ -state and  $n$ -state DFA, respectively, such that  $L(A) \cap L(B) = \emptyset$ . Then the language  $L(A) \cup L(B)$  is recognized by a DFA of at most  $mn - 1$  states. This upper bound is tight, and it is met by the binary witness languages recognized by DFAs shown in Fig. 1.  $\square$

The union operation in the case of a unary alphabet still requires all  $mn$  states, as long as  $m$  and  $n$  are relatively prime; the reduction of state complexity in the case when  $m$  and  $n$  have common divisors was studied by Pighizzini and Shallit [19]. However, if the union is disjoint, then it can be represented using half as many states.

**Theorem 4 (Unary Disjoint Union).** *Let  $m, n \geq 4$ . Let  $A$  be an  $m$ -state DFA and  $B$  an  $n$ -state DFA, both defined over a unary alphabet  $\Sigma = \{a\}$ , with  $L(A) \cap L(B) = \emptyset$ . Then the disjoint union  $L(A) \cup L(B)$  is recognized by a DFA with at most  $\lfloor \frac{1}{2}mn \rfloor$  states. This upper bound is tight whenever  $m$  and  $n$  are even numbers with  $\frac{m}{2}$  and  $\frac{n}{2}$  relatively prime, and it is met by the languages  $(a^m)^*$  and  $(a^n)^*$ .*

*Proof.* If either of the automata defines a finite language, then the union is recognized by a DFA with  $m + n$  states, which cannot exceed  $\frac{1}{2}mn$ : indeed,  $m + n \leq 2 \max(m, n) \leq \frac{1}{2} \min(m, n) \max(m, n) = \frac{1}{2}mn$ . Assume that both languages are infinite. Let  $k$  be the length of the cycle in  $A$ , and let  $\ell$  be the length of the cycle in  $B$ .

If the cycle lengths in  $A$  and in  $B$  are relatively prime, then the union is ambiguous. Let  $d$  be the greatest common divisor of their cycle lengths. Then the union  $L(A) \cup L(B)$  is recognized by a DFA with a cycle of length  $\text{lcm}(k, \ell) = \frac{1}{d}k\ell$ , which is at most  $\frac{1}{2}k\ell$ . In addition, there are  $\max(m - k, n - \ell)$  non-cyclic states. Assume that  $m - k \geq n - \ell$ . Then the number of states in the DFA is estimated as follows.

$$\frac{1}{2}k\ell + m - k \leq \frac{1}{2}k\ell + \frac{1}{2}(m - k)\ell = \frac{1}{2}m\ell \leq \frac{1}{2}mn$$

Now, consider the tightness. Since the first language consists of even-length strings and the other one of odd-length strings, their union is disjoint. Since  $\text{gcd}(m, n) = \frac{1}{2}mn$ , the union is periodic with this period, and therefore every DFA recognizing the union must have at least this many states.  $\square$

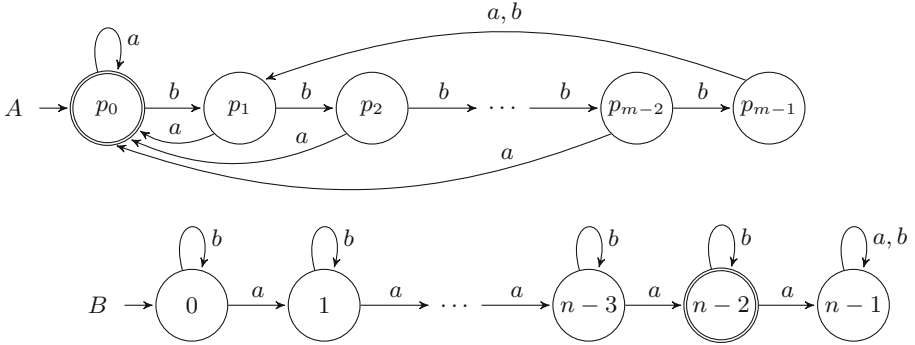
## 4 Concatenation

The state complexity of unambiguous concatenation has been investigated before, and the following result is known.

**Theorem A (Daley et al. [20]).** *Let  $A$  be an  $m$ -state DFA, and  $B$  an  $n$ -state DFA. If the concatenation  $L(A)L(B)$  is unambiguous, then it can be represented by a DFA with  $m2^{n-1} - 2^{n-2}$  states. The bound is tight for a four-symbol alphabet.*

This paper improves the tightness result by showing that the upper bound cannot be reduced already for a binary alphabet.

**Lemma 5.** *Let  $m, n \geq 3$  and  $A$  and  $B$  be the DFAs shown in Fig. 2. Then the concatenation  $L(A)L(B)$  is unambiguous and every DFA recognizing  $L(A)L(B)$  must have at least  $m2^{n-1} - 2^{n-2}$  states.*



**Fig. 2.** Binary witnesses for unambiguous concatenation meeting the upper bound  $m2^{n-1} - 2^{n-2}$ .

*Proof.* The concatenation is unambiguous, because every non-empty string in  $L(A)$  ends with an  $a$ , whereas every string in  $L(B)$  contains exactly  $n - 2$  occurrences of  $a$ . Thus, the only possible partition of a string into  $L(A) \cdot L(B)$  is to split it right after the  $(n - 1)$ -th last occurrence of  $a$ .

Let  $P = \{p_0, p_1, \dots, p_{m-1}\}$  be the state set of the DFA  $A$ , and  $Q = \{0, 1, \dots, n - 1\}$  the state set of  $B$ . An NFA for  $L(A)L(B)$  is constructed by omitting the dead state  $n - 1$  of  $B$ , by adding the transitions from  $p_i$  to  $0$  by  $a$ , for all  $p_i \in P \setminus \{p_{m-1}\}$ , and by making both  $p_0$  and  $0$  initial.

Each reachable subset is represented as  $(p, S)$  with  $p \in P$  and  $S \subseteq Q \setminus \{n - 1\}$ .

The first goal is to show that for each set  $S \subseteq Q$ , the state  $(p_0, \{0\} \cup S)$  is reachable, and so is each state  $(p_i, S)$ , with  $1 \leq i \leq m - 1$ . The proof is by induction on  $|S|$ . In the base case,  $|S| = 0$ , each pair  $(p_i, \emptyset)$ , with  $i \in \{1, 2, \dots, m - 1\}$ , is reached as follows.

$$(p_0, \{0\}) \xrightarrow{b} (p_1, \{0\}) \xrightarrow{(b^{m-2}a)^{n-2}} (p_1, \{n - 2\}) \xrightarrow{b^{m-2}a} (p_1, \emptyset) \xrightarrow{b^{i-1}} (p_i, \emptyset)$$

Let  $2 \leq k \leq n$  and assume that for each set  $S'$  with  $|S'| = k - 1$ , each state  $(p_i, S')$  with  $1 \leq i \leq m - 1$  and the state  $(p_0, \{0\} \cup S)$  is reachable. Let  $S \subseteq Q$  and  $|S| = k$ . Consider several cases:

- (1)  $i = 1$ . Let  $j = \min S$ . Take  $S' = \{q - j - 1 \mid q \in S \setminus \{j\}\}$ . Then  $S' \subseteq Q$  and  $|S'| = k - 1$ , so the state  $(p_1, S')$  is reachable by the induction assumption. The state  $(p_1, S)$  is reachable from it as follows.

$$(p_1, S') \xrightarrow{ab} (p_1, \{0\} \cup \{q - j \mid q \in S \setminus \{j\}\}) \xrightarrow{(b^{m-2}a)^j} (p_1, \{j\} \cup \{q \mid q \in S \setminus \{j\}\}) = (p_1, S)$$

- (2)  $2 \leq i \leq m - 1$ . Then  $(p_1, S)$  was shown to be reachable in case (1), and  $(p_i, S)$  is reached from it by the string  $b^{i-1}$ .
- (3)  $i = 0$ . Then  $(p_1, \{q - 1 \mid q \in S \setminus \{0\}\})$  is reachable as in case (1), and  $(p_0, \{0\} \cup S)$  is reached from it by the symbol  $a$ .

This proves the reachability of  $(m - 1)2^n + 2^{n-1} = m2^n - 2^{n-1}$  states.

To prove distinguishability, let  $(p_i, S)$  and  $(p_j, T)$  be two distinct states. If  $S \neq T$ , let  $j \in Q$  be in their symmetric difference, and assume, without loss of generality, that  $j \in S$  and  $j \notin T$ . Then the string  $a^{n-2-j}$  is accepted from  $(p_i, S)$ , but not from  $(p_j, T)$ . Let  $S = T$ , and, without loss of generality, let  $i < j$ . Then the string  $b^{m-1-j}a$  distinguishes  $(p_i, S)$  and  $(p_j, S)$  since

$$(p_j, S) \xrightarrow{b^{m-1-j}} (p_{m-1}, S) \xrightarrow{a} (p_1, \{q + 1 \mid q \in S\}),$$

$$(p_i, S) \xrightarrow{b^{m-1-j}} (p_{m-1-(j-i)}, S) \xrightarrow{a} (p_0, \{0\} \cup \{q + 1 \mid q \in S\}),$$

so, the resulting states differ in the second component, and therefore are distinguishable.  $\square$

All the above results are summarized in the next theorem. Then, the unary case is discussed.

**Theorem 6 (Unambiguous Concatenation).** *Let  $A$  and  $B$  be an  $m$ -state and  $n$ -state DFA, respectively, such that the concatenation  $L(A)L(B)$  is unambiguous. Then the language  $L(A)L(B)$  is recognized by a DFA of at most  $m2^{n-1} - 2^{n-2}$  states. This upper bound is tight, and it is met by the binary witness languages recognized by the DFAs shown in Fig. 2.*

**Theorem 7 (Unary Unambiguous Concatenation).** *Let  $m, n \geq 2$ , let  $A$  be an  $m$ -state and  $B$  an  $n$ -state DFA over a unary alphabet  $\Sigma = \{a\}$ , and let the concatenation  $L(A)L(B)$  be unambiguous. Then  $L(A)L(B)$  is recognized by a DFA with at most  $m + n - 2$  states, and this upper bound is tight.*

*Proof.* If both languages are infinite, then the concatenation is ambiguous. Therefore, one of the automata defines a finite language; since concatenation is commutative, there is no loss of generality in the assumption that this is  $A$ . Then  $A$  recognizes some subset of  $\{\varepsilon, a, a^2, \dots, a^{m-2}\}$ .

Let  $L(B)$  be periodic with period  $k$ , beginning from  $\ell$ ; then,  $k + \ell \leq n$ . Then, the concatenation  $L(A)L(B)$  is periodic with period  $k$ , beginning from  $m + \ell - 2$ , and is therefore recognized by a DFA with  $m + n - 2$  states.

In the worst case,  $m + n - 2$  states are necessary to represent the unambiguous concatenation of two unary languages represented by an  $m$ -state and an  $n$ -state DFA, which is witnessed by the singleton languages  $\{a^{m-2}\}$  and  $\{a^{n-2}\}$ . Their concatenation  $\{a^{m+n-4}\}$  requires a DFA with  $m + n - 2$  states.  $\square$

## 5 Star

The star of an  $n$ -state DFA is representable by a DFA with  $\frac{3}{4}2^n$  states, and this number is in the worst case necessary [17]. However, if the star is unambiguous, then the necessary number of states can be reduced. The proof is based on the property that for the star to be unambiguous, the automaton has to have a dead state. This property is in turn based on the following auxiliary result.



**Lemma 8.** *If a DFA  $A = (\Sigma, Q, q_0, \delta, F)$ , with  $L(A) \neq \emptyset$ , has no dead states, then there is an accepting state  $q \in F$  which is in a cycle, that is,  $\delta(q, u) = q$  for some non-empty string  $u \in \Sigma^*$ .*

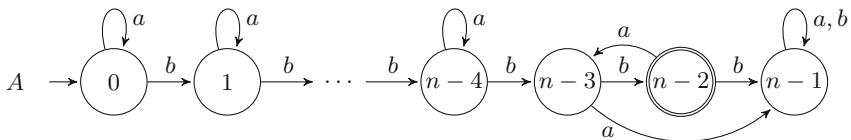
*Proof.* Let  $u_0 \in L(A)$ . Then  $\delta(q_0, u_0) = q_1$  for some accepting state  $q_1$ . Since  $A$  does not have any dead state, there is a non-empty string  $u_1$  accepted from  $q_1$ . Let  $q_2 = \delta(q_1, u_1)$  be the state, in which it is accepted. This yields a sequence of accepting states  $q_i$  and non-empty strings  $u_i$ , with  $i \geq 1$ . Since  $A$  has finitely many accepting states, this sequence eventually revisits some state, that is,  $q_i = q_j$  for some  $i < j$ . Then the state  $q_i$  is cyclic, with  $q_i = \delta(q_i, u_1 \dots u_{j-1})$ .  $\square$

**Lemma 9.** *If  $A$  is a DFA and the star  $L(A)^*$  is unambiguous, then  $A$  has a dead state.*

*Proof.* Let  $A = (\Sigma, Q, q_0, \delta, F)$  be a minimal DFA for  $L$ . Suppose for a contradiction that  $A$  does not have any dead state. Then, by Lemma 8, there is an accepting state  $p \in F$  and a nonempty string  $v \in \Sigma^*$ , with  $\delta(p, v) = p$ . Consider the sequence of states  $q_i$ , with  $i \geq 0$ , defined by  $q_i = \delta(q_0, v^i)$ . Since the number of states in  $A$  is finite, there are numbers  $j \geq 0$  and  $k \geq 1$  with  $q_j = q_{j+k}$ , and thus  $\delta(q_j, v^k) = q_j$ . Now let  $u$  be a string, by which  $p$  is reached from  $q_0$ , and let  $w$  be any string accepted from  $q_j$ . Then the string  $uv^{j+k}w$  can be partitioned as  $u \cdot v^{j+k}w$  or  $uv^k \cdot v^jw$ , which is a contradiction with the unambiguity of  $L^*$ . Therefore,  $A$  must have a dead state.  $\square$

**Lemma 10.** *For every  $n \geq 4$ , the language  $L_n = (a^*b)^{n-3}b(ab)^*$  is recognized by an  $n$ -state DFA, the star  $L_n^*$  is unambiguous, and every DFA recognizing  $L_n^*$  must have at least  $\frac{3}{8}2^n + 1$  states.*

*Proof.* The language  $L_n$  is recognized by the DFA  $A$  shown in Fig. 3, Construct an NFA  $A^*$  from  $A$  by omitting the dead state  $n - 1$ , by adding the transition  $(n - 3, b, 0)$ , and by adding one more initial and final state  $s$ ; see Fig. 4. The NFA  $A^*$  is unambiguous, since the intersection of every reachable and every co-reachable set of  $A^*$  is of size at most one.



**Fig. 3.** A binary witness for unambiguous star meeting the upper bound  $\frac{3}{8}2^n + 1$ .

Let us show that the subset automaton  $\mathcal{D}(A^*)$  has  $\frac{3}{8}2^n + 1$  reachable and pairwise distinguishable states. The initial subset is  $\{s, 0\}$  and it is sent to the state  $[0, n - 3]$  by  $b^{n-3}(ba)^n$ . Next, every subset  $S$  of  $[0, n - 3]$  can be shifted cyclically by one, that is, it can be sent to  $\{(s + 1) \bmod (n - 2) \mid s \in S\}$ , by

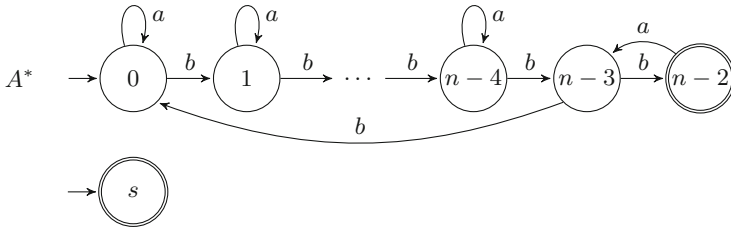


Fig. 4. NFA  $A^*$  for the star of the language accepted by DFA  $A$  from Fig. 3.

reading  $b$  if  $n - 3 \notin S$ , by reading  $ba$  if  $\{n - 4, n - 3\} \subseteq S$ , and by reading  $baa$  if  $n - 3 \in S$  and  $n - 4 \notin S$ . Moreover, the state  $n - 3$  can be eliminated from every subset of  $[0, n - 3]$  containing the state  $n - 3$  by reading  $a$ . It follows that every subset of  $[0, n - 3]$  is reachable from  $[0, n - 3]$ .

Now let  $n - 2 \in S$ . Then also  $0 \in S$ . Let  $S' = \{s - 1 \mid s \in S \setminus \{0\}\}$ . Then  $S' \subseteq [0, n - 3]$ , so  $S'$  is reachable as shown above. Since  $n - 3 \in S'$ , the set  $S'$  is sent to  $S$  by  $b$ . This proves reachability.

To prove distinguishability, let  $S$  and  $S'$  be any two distinct reachable subsets of the subset automaton  $\mathcal{D}(A^*)$ . Then they must differ in some state  $i \in [0, n - 2]$ . Assume, without loss of generality, that  $i \in S$  and  $i \notin S'$ . Then, the string  $b^{n-2-i}$  is accepted from  $S$ , but not from  $S'$ . □

The next theorem summarizes the results on the star operation. Then, the unary case is discussed.

**Theorem 11 (Unambiguous Star).** *Let  $n \geq 4$  and  $A$  be an  $n$ -state DFA such that  $L(A)^*$  is unambiguous. Then the language  $L(A)^*$  is accepted by a DFA of at most  $\frac{3}{8}2^n + 1$  states. This upper bound is tight, and it is met by the binary witness language  $(a^*b)^{n-3}b(ab)^*$ .*

*Proof.* To get an NFA  $A^*$  for  $L^*$  from the DFA  $A$ , first omit the dead state. Then add the transition  $(q, a, q_0)$  whenever  $\delta(q, a) \in F$ . Finally, add one more initial and final state  $s$  with no transitions going from it. In the subset automaton  $\mathcal{D}(A^*)$ , the initial subset is  $\{s, q_0\}$ , and no other reachable subset contains  $q_0$ . Moreover, no subset which contains a final state of  $A$  but does not contain state  $q_0$  is reachable. In total, the number of reachable subsets in  $\mathcal{D}(A^*)$  is at most  $1 + 2^{n-1} - 2^{n-2} = \frac{3}{8}2^n + 1$ . The tightness of this upper bound follows from Lemma 10. □

**Theorem 12 (Unary Unambiguous Star).** *Let  $A$  be an  $n$ -state unary DFA such that  $L(A)^*$  is unambiguous. Then  $L(A)^*$  is accepted by a DFA of at most  $n - 2$  states. This upper bound is tight, and it is met by the unary language  $\{a^{n-2}\}$ .*

*Proof.* In the unary case, for  $L^*$  to be unambiguous,  $L$  must be a singleton. Furthermore, if an  $n$ -state DFA recognizes a singleton  $\{a^\ell\}$ , then  $\ell \leq n - 2$ , whereas the star of this language,  $(a^\ell)^*$ , is recognized by a DFA with  $\ell$  states.

Thus, unambiguous star of a unary DFA is representable using  $n - 2$  states. This number of states is necessary, witnessed by the language  $\{a^{n-2}\}$ .  $\square$

## 6 Summary of Results

State complexity of basic operations on regular languages and of their unambiguous variants studied in this paper is compared in Table 1 for three automata models: DFA, UFA and NFA.

**Table 1.** State complexity of standard and unambiguous operations for DFA, UFA and NFA: union ( $\cup$ ), disjoint union ( $\uplus$ ), concatenation ( $\cdot$ ), unambiguous concatenation (UNAMB $\cdot$ ), Kleene star ( $*$ ), unambiguous Kleene star (UNAMB $*$ ).

	DFA	UFA	NFA
$\cup$	$mn$ [17]	$\leq m + O(n2^{0.79m})$ [11]	$m + n$ [9]
$\uplus$	$mn - 1$	$\leq m + n$	$\leq m + n$
$\cdot$	$m2^n - 2^{n-1}$ [17]	$\frac{3}{4}2^{m+n} - 1$ [11]	$m + n$ [9]
UNAMB $\cdot$	$m2^{n-1} - 2^{n-2}$ [20]	$\leq m + n$	$\leq m + n$
$*$	$\frac{3}{4}2^n$ [17]	$\frac{3}{4}2^n$ [11]	$n + 1$ [9]
UNAMB $*$	$\frac{3}{8}2^n + 1$	$\leq n + 1$	$\leq n + 1$

For the three unambiguous operations on DFA, their state complexity has been established in this paper. These operations are easy to apply to UFA using the standard constructions for the union, concatenation and star of NFA: indeed, since the operations are unambiguous, they preserve the unambiguity of the automata involved; however, it remains to establish matching lower bounds.

Another line of related state complexity research is concerned with basic operations on *prefix-free languages*, that is, those with the property that  $uv \in L$ , with  $v \in \Sigma^+$ , implies that  $u \notin L$ . *Suffix-free languages* are defined similarly. Notably, for prefix-free and for suffix-free languages, both concatenation and star are unambiguous, and hence the state complexity results on these subcases are natural tight upper bounds for these two operations. For prefix-free languages and for the DFA model, union has state complexity  $mn - 2$  [12], the state complexity of concatenation is  $m + n - 1$  [7, 12], whereas the star has state complexity  $n$  [7, 12]. For suffix-free languages, the results are completely different: union has state complexity  $mn - m - n + 2$  [5, 13], concatenation has  $(m - 1)2^{n-2} + 1$  [3, 5], and for the star it is  $2^{n-2} + 1$  [3, 5].

One more related research direction is the recent study of variants of the basic operations on languages defined over the field  $GF(2)$  instead of the standard Boolean logic [1]. The union operation turns into the symmetric difference, whereas concatenation gives rise to the following new *GF(2)-concatenation* operation.

$$K \odot L = \{ w \mid \# \text{ of partitions } w = uv, \text{ with } u \in K \text{ and } v \in L, \text{ is odd} \}$$

The  $GF(2)$ -star is defined similarly, so as to preserve only the strings with an odd number of partitions. Notably, the unambiguous operations studied in this paper are a special case of the  $GF(2)$  operations in the same way as they are a special case of classical operations: indeed, the differences between classical and  $GF(2)$  operations are in the treatment of ambiguity. As the  $GF(2)$  operations preserve regularity, their state complexity is worth being compared to the unambiguous and the classical cases. So far, it has been proved that for DFA,  $GF(2)$ -concatenation has state complexity  $m \cdot 2^n$ , while the state complexity of the  $GF(2)$ -star is  $2^n + 1$  [1].

Another incomparable extension of unambiguous concatenation and star are the *unique concatenation* and the *unique star* [20], defined similarly, using the uniqueness of partition as the condition of membership. The state complexity of unique concatenation is at most  $m3^n - 3^{n-1}$ , and for the unique star the upper bound is  $2 \cdot 3^{n-1} - \frac{3}{4}2^n + 2$  [20], their tightness remains open.

**Table 2.** State complexity of operations in the case of a unary alphabet.

	DFA	UFA	NFA
$\cup$	$\leq mn$ [19]		$m + n$ [9]
$\uplus$	$\leq \frac{1}{2}mn$	$\leq m + n$	$\leq m + n$
$\cdot$	$\leq mn$ [21]		$m + n$ [9]
UNAMB $\cdot$	$m + n - 2$	$\leq m + n$	$\leq m + n$
$*$	$(n - 1)^2 + 1$ [21]	$(n - 1)^2 + 1$ [18]	$n + 1$ [9]
UNAMB $*$	$n - 2$	$\leq n + 1$	$\leq n + 1$

In the next Table 2, the state complexity of all the same operations is compared in the case of a unary alphabet. Again, for DFA, the state complexity of unambiguous operations has been established, whereas for UFA and for NFA there are only obvious upper bounds. For UFA, the state complexity of standard operations remains to be investigated, with only a few results known [18].

## References

1. Bakinova, E., Basharin, A., Batmanov, I., Lyubort, K., Okhotin, A., Sazhneva, E.: Formal languages over  $GF(2)$ . In: Klein, S.T., Martín-Vide, C., Shapira, D. (eds.) LATA 2018. LNCS, vol. 10792, pp. 68–79. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-77313-1\\_5](https://doi.org/10.1007/978-3-319-77313-1_5)
2. Brzozowski, J.A., Szykuła, M.: Complexity of suffix-free regular languages. J. Comput. Syst. Sci. **89**, 270–287 (2017). <https://doi.org/10.1016/j.jcss.2017.05.011>
3. Cmorik, R., Jirásková, G.: Basic operations on binary suffix-free languages. In: Kotásek, Z., Bouda, J., Černá, I., Sekanina, L., Vojnar, T., Antoš, D. (eds.) MEMICS 2011. LNCS, vol. 7119, pp. 94–102. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-25929-6\\_9](https://doi.org/10.1007/978-3-642-25929-6_9)

4. Daley, M., Domaratzki, M., Salomaa, K.: Orthogonal concatenation: language equations and state complexity. *J. Univers. Comput. Sci.* **16**(5), 653–675 (2010). <https://doi.org/10.3217/jucs-016-05-0653>
5. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. *Theoret. Comput. Sci.* **410**, 2537–2548 (2009). <https://doi.org/10.1016/j.tcs.2008.12.054>
6. Han, Y.-S., Salomaa, K.: Nondeterministic state complexity for suffix-free regular languages. In: *DCFS 2010, EPTCS*, vol. 31, pp. 189–196 (2010). <https://doi.org/10.4204/EPTCS.31.21>
7. Han, Y.-S., Salomaa, K., Wood, D.: Operational state complexity of prefix-free regular languages. In: *Automata, Formal Languages, and Related Topics*, pp. 99–115 (2009)
8. Han, Y.-S., Salomaa, K., Wood, D.: Nondeterministic state complexity of basic operations for prefix-free regular languages. *Fundamenta Informaticae* **90**(1–2), 93–106 (2009). <https://doi.org/10.3233/FI-2009-0008>
9. Holzer, M., Kutrib, M.: Nondeterministic descriptonal complexity of regular languages. *Int. J. Found. Comput. Sci.* **14**, 1087–1102 (2003). <https://doi.org/10.1142/S0129054103002199>
10. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. *Int. J. Found. Comput. Sci.* **16**(3), 511–529 (2005). <https://doi.org/10.1142/S0129054105003133>
11. Jirásek, J., Jirásková, G., Šebej, J.: Operations on unambiguous finite automata. In: *Brlek, S., Reutenauer, C. (eds.) DLT 2016. LNCS*, vol. 9840, pp. 243–255. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-53132-7\\_20](https://doi.org/10.1007/978-3-662-53132-7_20)
12. Jirásková, G., Krausová, M.: Complexity in prefix-free regular languages. In: *DCFS 2010, EPTCS*, vol. 31, pp. 197–204. <https://doi.org/10.4204/EPTCS.31.22>
13. Jirásková, G., Olejár, P.: State complexity of intersection and union of suffix-free languages and descriptonal complexity. In: *NCMA 2009, books@ocg.at*, vol. 256, 151–166 (2009)
14. Jirásková, G., Okhotin, A.: On the state complexity of operations on two-way finite automata. *Inf. Comput.* **253**(1), 36–63 (2017). <https://doi.org/10.1016/j.ic.2016.12.007>
15. Kunc, M., Okhotin, A.: State complexity of union and intersection for two-way non-deterministic finite automata. *Fundamenta Informaticae* **110**(1–4), 231–239 (2011). <https://doi.org/10.3233/FI-2011-540>
16. Kunc, M., Okhotin, A.: State complexity of operations on two-way deterministic finite automata over a unary alphabet. *Theoret. Comput. Sci.* **449**, 106–118 (2012). <https://doi.org/10.1016/j.tcs.2012.04.010>
17. Maslov, A.N.: Estimates of the number of states of finite automata. *Soviet Math. Dokl.* **11**, 1373–1375 (1970)
18. Okhotin, A.: Unambiguous finite automata over a unary alphabet. *Inf. Comput.* **212**, 15–36 (2012). <https://doi.org/10.1016/j.ic.2012.01.003>
19. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal’s function. *Int. J. Found. Comput. Sci.* **13**(1), 145–159 (2002). <https://doi.org/10.1142/S012905410200100X>
20. Rampersad, N., Ravikumar, B., Santean, N., Shallit, J.: State complexity of unique rational operations. *Theoret. Comput. Sci.* **410**, 2431–2441 (2009). <https://doi.org/10.1016/j.tcs.2009.02.035>
21. Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages. *Theoret. Comput. Sci.* **125**, 315–328 (1994). [https://doi.org/10.1016/0304-3975\(92\)00011-F](https://doi.org/10.1016/0304-3975(92)00011-F)



# Cycle Height of Finite Automata

Chris Keeler<sup>(✉)</sup> and Kai Salomaa

School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada  
{keeler,ksalomaa}@cs.queensu.ca

**Abstract.** A nondeterministic finite automaton (NFA)  $A$  has cycle height  $\mathcal{K}$  if any computation of  $A$  can visit at most  $\mathcal{K}$  cycles, and  $A$  has finite cycle height if it has cycle height  $\mathcal{K}$  for some  $\mathcal{K}$ . We give a polynomial time algorithm to decide whether an NFA has finite cycle height and, in the positive case, to compute its optimal cycle height. Nondeterministic finite automata of finite cycle height recognize the polynomial density regular languages.

## 1 Introduction

Deterministic and nondeterministic finite automata define the class of regular languages and have been systematically studied for over 60 years. At the same time, many important questions on finite automata and regular languages remain open [8, 10]. The last decades have seen much work on the descriptive complexity, or state complexity, of regular languages [4, 6, 7].

In this paper we consider a structural property of finite automata called *cycle height*. A nondeterministic finite automaton (NFA) is said to have finite cycle height if no two cycles overlap. A finite cycle height NFA  $A$  has cycle height  $\mathcal{K}$  if all computations of  $A$  visit no more than  $\mathcal{K}$  non-equivalent cycles.<sup>1</sup> The acyclic NFAs have cycle height zero and the nearly acyclic NFAs [9] have cycle height one.

Note that cycle height differs from the notion of *cycle rank* [5] which counts the degree of nesting of cycles in an NFA. Also Msiska and van Zijl [12] estimate the size blow-up of the subset construction by counting how many times a computation passes through a simple cycle. The notion is in some sense related to cycle height, but their point of view is different because the algorithm modifies the NFA by removing nested cycles.

A language  $L$  has *polynomial density* if the number of strings of length  $n$  in  $L$  is bounded by a polynomial in  $n$ . Szilard et al. [15] have shown that a language recognized by a deterministic finite automaton (DFA)  $A$  has polynomial density if all strings have a certain *tiered* property with respect to  $A$ . The tiered property is related to our notion of cycle height of NFAs, although in [15] the tiered words are defined only with respect to a DFA. As noted by

<sup>1</sup> By non-equivalent cycles we mean cycles that are not permutations of each other. The notion will be defined formally in Sect. 2.

Kutrib et al. [11], from [15] it follows that, for a polynomial density regular language  $L$ , the degree of the polynomial giving the density of  $L$  is computable. Using more advanced techniques, Gawrychowski et al. [3] have shown that for an  $m$ -state DFA  $A$  recognizing a polynomial density language the degree of the polynomial giving the density of the language can be computed in  $O(m)$  time, assuming the alphabet size is constant.

The contributions of this paper are as follows. In Sect. 3 we give a polynomial time algorithm to decide whether an NFA  $A$  has finite cycle height and, in the positive case, to compute the cycle height of  $A$ . We show that NFAs with finite cycle height recognize the polynomial density regular languages, but an NFA recognizing a polynomial density language need not have finite cycle height. Based on results from [15] it then follows that a DFA  $A$  has finite cycle height if and only if the language  $L(A)$  has polynomial density. Furthermore, if  $A$  has finite cycle height, the degree of the polynomial bounding the density of  $A$  is the cycle height of  $A$  minus one. This would give a polynomial time algorithm to compute the density of a language recognized by a DFA, however, the time complexity is worse than in the known algorithm from Gawrychowski et al. [3]. Finally in Sect. 4 we study upper and lower bounds for the *depth path width* [9] of NFAs with finite cycle height. The depth path width of an NFA  $A$ , roughly speaking, quantifies the overall path expansion in computations of  $A$  by counting the number of complete computations of  $A$  on all possible inputs of a given length.

## 2 Preliminaries

We assume the reader to be familiar with the basics of formal languages and finite automata [14]. The set of strings over a finite alphabet  $\Sigma$  is  $\Sigma^*$ , the set of strings of length  $m \geq 0$  is  $\Sigma^m$  and  $\varepsilon$  is the empty string. The cardinality of a finite set  $F$  is denoted  $|F|$  and  $\mathbb{N}$  is the set of non-negative integers.

A *nondeterministic finite automaton* (NFA) is a tuple  $A = (Q, \Sigma, \delta, q_0, F)$  where  $Q$  is the finite set of states,  $\Sigma$  is the input alphabet,  $\delta : Q \times \Sigma \rightarrow 2^Q$  is the transition function,  $q_0 \in Q$  is the initial state and  $F \subseteq Q$  is the set of final states. The transition function  $\delta$  is in the usual way extended as a function  $Q \times \Sigma^* \rightarrow 2^Q$  and the language recognized by  $A$  is  $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$ . If  $|\delta(q, b)| \leq 1$  for all  $q \in Q$  and  $b \in \Sigma$ , the automaton  $A$  is a *deterministic finite automaton* (DFA). It is well known that the NFAs and DFAs recognize the class of *regular languages*.

Unless otherwise mentioned, we always assume that an NFA or a DFA does not have useless states, that is, states that cannot be used in an accepting computation. Note that we can avoid a DFA having a “useless” dead state because we allow DFAs to have undefined transitions.

The *density* of a language  $L$  is a function  $\mathbb{N} \rightarrow \mathbb{N}$  defined as  $\varrho_L(n) = |L \cap \Sigma^n|$ . A language  $L$  is said to have polynomial density if there exists an integer  $d \geq 0$  such that  $\varrho_L(n) \in O(n^d)$ . Density is sometimes instead defined as the ratio  $|L \cap \Sigma^n|/|\Sigma^n|$ , with our notion of density instead being referred to as *population* [15].

### 2.1 Cycle Height and Depth Path Width

First we recall some definitions related to computations and cycles of an NFA. In the following  $A = (Q, \Sigma, \delta, q_0, F)$  is always an NFA.

A (*state*) *path* of the NFA  $A$  with underlying string  $w = b_1b_2 \cdots b_k$ ,  $b_i \in \Sigma$ ,  $i = 1, \dots, k$ ,  $k \geq 0$ , is a sequence of states  $(p_0, p_1, \dots, p_k)$ , where  $p_j \in \delta(p_{j-1}, b_j)$ ,  $j = 1, \dots, k$ . A path beginning in the start state  $q_0$ , is a *computation* of  $A$  on the underlying string  $w$ . A computation that ends in an accepting state of  $F$  is an accepting computation. A computation  $(q_0, p_1, \dots, p_\ell)$  is a *complete computation* on a string  $b_1b_2 \cdots b_k$  if  $\ell = k$ . The set of all computations of  $A$  on the string  $w$  is denoted  $\text{comp}_A(w)$ .

A path  $(p_0, p_1, \dots, p_k)$ ,  $k \geq 1$ , with underlying string  $b_1b_2 \cdots b_k$  is a *cycle* if  $p_0 = p_k$ . A cycle with one transition from a state to itself is called a *self-loop*.

Cycles that are obtained from each other by a cyclical shift are said to be *equivalent*: For  $0 < i < k$ , the above cycle (with  $p_0 = p_k$ ) is equivalent to the cycle  $(p_i, \dots, p_k, p_1, \dots, p_{i-1}, p_i)$  having underlying string  $b_{i+1} \cdots b_k b_1 \cdots b_i$ . In the following, unless otherwise mentioned, by a cycle we always mean an equivalence class of cycles and, thus, by two distinct cycles we mean two non-equivalent cycles.

We say that an NFA  $A$  has *finite cycle height* if for any distinct cycles  $C_1$  and  $C_2$  of  $A$ , either  $C_1$  is unreachable from  $C_2$  or  $C_2$  is unreachable from  $C_1$ . It is easy to see that an NFA has finite cycle height if and only if no two different cycles overlap. Additionally, this condition implies a strict ordering on the cycles, since the reachability between two distinct cycles holds in at most in one direction.

A finite cycle height NFA  $A$  has cycle height  $\mathcal{K} \in \mathbb{N}$  if no computation of  $A$  can contain states belonging to  $\mathcal{K} + 1$  different cycles. Intuitively, this means that no computation can “pass through”  $\mathcal{K} + 1$  different cycles. Note that since  $A$  has finite cycle height and a strict ordering on its cycles, a computation can “pass through” a cycle at most once. We say that  $A$  has *strict cycle height*  $\mathcal{K}$  if it has cycle height  $\mathcal{K}$  but not cycle height  $\mathcal{K} - 1$ . Note that if  $A$  has cycle height  $\mathcal{K}$ , then there exists a unique  $0 \leq \mathcal{K}' \leq \mathcal{K}$  such that  $A$  has strict cycle height  $\mathcal{K}'$ .

An acyclic NFA has cycle height zero and a nearly acyclic NFA [9] has cycle height one (by its definition). Since a finite cycle height NFA cannot have overlapping cycles, the Lemma below is immediate. An  $n$ -state NFA with cycle height  $n$  is given in Fig. 2.

**Lemma 1.** *The finite cycle height of an  $n$ -state NFA is at most  $n$ . For each  $n \in \mathbb{N}$  and  $\mathcal{K} \leq n$  there exists an  $n$ -state NFA with strict cycle height  $\mathcal{K}$ .*

To conclude this section we recall the notion of depth path width, which counts the number of complete computations of given length. The *depth path width* [9] of  $A$  on strings of length  $\ell \in \mathbb{N}$  is

$$\text{DPW}(A, \ell) = \sum_{w \in \Sigma^\ell} |\text{comp}_A(w)|.$$

The depth path width of the NFA  $A$  is defined as  $\text{DPW}^{\text{sup}}(A) = \sup_{\ell \in \mathbb{N}} (\text{DPW}(A, \ell))$ .



In Sect. 4 we will use the following Lemma.

**Lemma 2.** *Let  $A$  be an NFA and  $\ell \in \mathbb{N}$ .*

- (i) *If  $A'$  is an NFA obtained from  $A$  by changing the alphabet symbol labeling one transition, then  $\text{DPW}(A', \ell) = \text{DPW}(A, \ell)$ .*
- (ii) *If  $A''$  is an NFA obtained from  $A$  by adding one new transition, then*

$$\text{DPW}(A'', \ell) \geq \text{DPW}(A, \ell).$$

### 3 Polynomial Time Algorithm for Cycle Height

We present an algorithm which determines whether or not an NFA  $A$  has finite cycle height, and if so, returns the strict cycle height of  $A$ .

The idea of Algorithm 1 is as follows: For an NFA  $A$ , we first split  $A$  into its strongly connected components (SCCs). We then ensure that  $A$  does not have any nested cycles, which would prevent finite cycle height. This is done by checking that each SCC is either an acyclic singleton (consisting of only one state and no transitions) or a simple cycle (where consecutive states are connected by a unique transition).

After this, the algorithm creates an acyclic graph  $G = (V, E)$ ,  $V = \{v_0, \dots, v_{k-1}\}$ , where each vertex represents one of  $A$ 's strongly connected components. Each edge  $(v_i, v_j)$  represents a connection in  $A$  between the two SCCs  $s_i$  and  $s_j$ , for  $0 \leq i < j \leq k - 1$ . The weights of these edges represent the type of SCC to which the edge leads. That is, if  $(v_i, v_j)$  is a 0-weight edge in  $E$ , then  $s_j$  is (and  $v_j$  represents) an acyclic singleton. If  $(v_i, v_j)$  is a 1-weight edge in  $E$ , then  $s_j$  is (and  $v_j$  represents) a simple cycle SCC.

The algorithm then determines the minimum distance from  $v_0$  (the vertex representing the SCC containing  $q_0$ ) to all other  $v_j$ . Since each 1-weight edge leads to a vertex representing a cyclical SCC, the maximum-cost path starting from  $v_0$  will lead through the most vertices representing cyclical SCCs. In fact, the length of the maximum-cost path in  $G$  starting from  $v_0$  is the integer  $\mathcal{K}$ , such that  $A$  has strict cycle height  $\mathcal{K}$ .

In the algorithm, for states  $q_a$  and  $q_b$  of  $A$ , the *distance* from  $q_a$  to  $q_b$  is the length of the shortest string that takes  $q_a$  to  $q_b$ . If  $q_b$  is not reachable from  $q_a$  the distance is  $\infty$ .

*Complexity analysis of Algorithm 1:* The input is an NFA  $(Q, \Sigma, \delta, q_0, F)$  with  $m$  states. Creating the distance matrix takes  $\Theta(m^3)$  time and  $\Theta(m^2)$  space using the Floyd-Warshall reachability algorithm [2]. Creating the set of strongly connected components can be done in  $\Theta(m + |\delta|)$  time using Tarjan's SCC algorithm [16]. Checking for the existence of an SCC which is not a simple cycle is naturally bounded by the number of states and transitions in  $A$ . Clearly then, the "if" part on line 4 is not as computationally hard as the "else" part on line 6. For the else part, the two for all statements multiply the inner statements' complexity by  $O(\binom{m}{2})$ , as they enumerate all ordered pairs and there are maximally  $m$  SCCs.

**Algorithm 1.** Computing the Cycle Height of an NFA

---

```

1: Let  $A = (Q, \Sigma, \delta, q_0, F)$  be an NFA where  $|Q| = m$ .
2: Create a distance matrix  $M$ , where  $M[q_a, q_b]$  is the distance from state  $q_a \in Q$  to state  $q_b \in Q$ .
3: Let  $s_0, s_1, \dots, s_{k-1}$ ,  $k \geq 1$ , be the strongly connected components of  $A$ .
4: if there exists  $s_i$ ,  $0 \leq i \leq k-1$ , such that  $s_i$  is not a simple cycle or acyclic singleton then
5:   return “ $A$  does not have finite cycle height”
6: else
7:   if the start state  $q_0$  is in an acyclic singleton then
8:     startBias = 0
9:   else
10:    startBias = 1
11:   end if
12: Create an acyclic graph  $G=(V, E)$ ,  $V=\{v_0, \dots, v_{k-1}\}$ ,  $E=\emptyset$ , where each  $v_i$  represents the strongly connected component  $s_i$ , for  $0 \leq i \leq k-1$ .
13: for all  $s_i$ ,  $0 \leq i < k-1$  select one state  $q_i$  in  $s_i$  and do
14:   for all  $s_j$ ,  $i < j \leq k-1$  select one state  $q_j$  in  $s_j$  and do
15:     if  $M[q_i, q_j] \neq \infty$  then
16:       if  $s_j$  is an acyclic singleton consisting of state  $s_j$  then
17:         Add a 0-weight edge to  $E$  from vertex  $v_i$  to vertex  $v_j$ .
18:       else
19:         Add a 1-weight edge to  $E$  from vertex  $v_i$  to vertex  $v_j$ .
20:       end if
21:     end if
22:   end for
23: end for
24: Let  $D$  be the distances from  $v_0$  to all other vertices in  $V$ .
25: return  $\max_{v \in V}(D[v]) + startBias$ 
26: end if

```

---

Since we know that there is a strict ordering on the cycles, we do not need to compare all pairs of SCCs, as  $q_i$  is never reachable from  $q_j$  when  $i < j$ .

Determining the reachability between SCCs is done in constant time with the help of the distance matrix  $M$ . We create the shortest-path tree  $D$  for  $G$  using the modified Dijkstra’s algorithm given in [13], which takes  $O(|E| + |V| \cdot \log C)$  time, where  $C$  is the largest edge value. An upper bound for the runtime of the algorithm is

$$\Theta(m^3 + m + |\delta| + \binom{m}{2} + |E| + |V| \cdot \log C)$$

Since 0 and 1 are the only edge values used, the constant  $C$  is ignored and the runtime simplifies to  $\Theta(\max(m^3, |\delta|))$ .

Using Algorithm 1, we obtain the following two results. Note that Theorem 1 assumes that the input NFA has finite cycle height. This property can be decided using Theorem 2.

**Theorem 1.** *If  $A$  is an NFA with  $m$  states, transition function  $\delta$  and finite cycle height, then we can compute in time  $O(\max(m^3, |\delta|))$  and space  $O(\max(m^2, |\delta|))$  the strict cycle height of  $A$ .*

If we consider the alphabet to be fixed, as is often done, the time bound of Theorem 1 simplifies to  $O(m^3)$ .

Second, if we modify the algorithm so that the distance matrix  $M$  is never calculated, and the else part on line 6 just returns 1 (instead of lines 10 through 25), then we can decide whether an NFA has finite cycle height just using Tarjan’s SCC algorithm, checking that each SCC is a simple cycle or an acyclic singleton.

**Theorem 2.** *If  $A$  is an NFA with  $m$  states and a transition function  $\delta$ , we can decide in time  $O(m + |\delta|)$  whether or not  $A$  has finite cycle height.*

### 3.1 Relationship with Polynomial Density Languages

The cycle height of an NFA  $A$  can be related to language classes recognized by  $A$ . It is known that nearly acyclic NFAs, or NFAs of cycle height one, recognize exactly the constant density languages [9]. We recall the following characterization of polynomial density regular languages from [15].

**Proposition 1** ([15]). *A regular language is of polynomial density (of degree at most  $k$ ) if and only if it can be represented as a finite union of regular expressions of the form  $x \cdot y_1^* z_1 \dots y_t^* z_t$ , with each  $t \leq k + 1$  and  $x, y_1, z_1, \dots, y_t, z_t \in \Sigma^*$ .*

Using the above characterization we can verify that the languages recognized by finite cycle height NFAs have polynomial density. Note that Szilard et al. [15] define a notion of a  $t$ -tiered string, and this notion is closely related to our notion of cycle height.<sup>2</sup> If all strings are  $t$ -tiered with respect to an automaton  $A$  then the language of  $A$  has a representation as in Proposition 1. Using this observation we can extend Proposition 1 for NFAs of given cycle height.

**Proposition 2.** *If  $A$  is an NFA with cycle height  $\mathcal{K}$ , then*

$$L(A) = \bigcup_{i=1}^r x_i \cdot [y_{i,1}^* z_{i,1} \dots y_{i,t_i}^* z_{i,t_i}],$$

for  $x_i, y_{i,j}, z_{i,j} \in \Sigma^*$ , and some integers  $j, t_i$  and  $r$ , such that  $t_i \leq \mathcal{K}$  for all  $i$ .

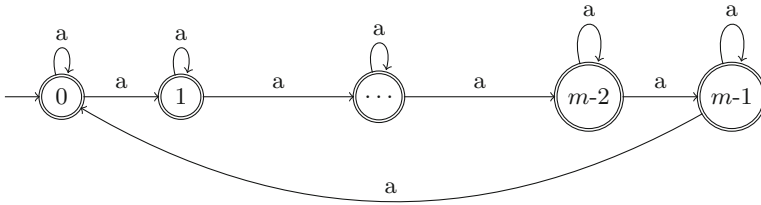
As a corollary, we get an explicit upper bound for the density of a language recognized by an NFA of cycle height  $\mathcal{K}$ .

**Corollary 1.** *If  $A$  is an NFA with cycle height  $\mathcal{K}$ , then  $\varrho_{L(A)}(n) \in O(n^{\mathcal{K}-1})$ .*

The reverse is not true: an NFA recognizing a polynomial density language need not have finite cycle height. As a counterexample, we give the NFA in Fig. 1, which does not have finite cycle height, but whose language,  $L = a^*$ , has constant density.

For a DFA  $A$  that has strict cycle height  $\mathcal{K}$ , we can use results of [15] to strengthen Corollary 1 such that it gives the precise density of  $L(A)$ .

<sup>2</sup> Strictly speaking, the  $t$ -tiered words are defined in [15] only with respect to a DFA.



**Fig. 1.** NFA with language  $a^*$

**Lemma 3.** *Let  $A$  be a DFA with strict cycle height  $\mathcal{K}$ . Then the density of  $L(A)$  is in  $\Omega(n^{\mathcal{K}-1})$ .*

*Proof.* Since  $A$  has strict cycle height  $\mathcal{K}$ , there exists a string  $w$  such that the accepting computation of  $A$  on  $w$  visits  $\mathcal{K}$  different cycles. (Here we rely on the assumption that  $A$  has no useless states, which implies that any computation can be extended to an accepting computation.) This means that  $w$  is  $\mathcal{K}$ -tiered with respect to  $A$  (as defined in [15]) and the claim follows from Lemma 1 of [15]. □

From Lemma 3 and Corollary 1 we see that the cycle height of a DFA exactly characterizes the density of the recognized language. Note that cycle height zero DFAs recognize finite languages.

**Corollary 2.** *If  $A$  is a DFA with strict cycle height  $\mathcal{K} \geq 1$  then the density of  $L(A)$  is  $\Theta(n^{\mathcal{K}-1})$ .*

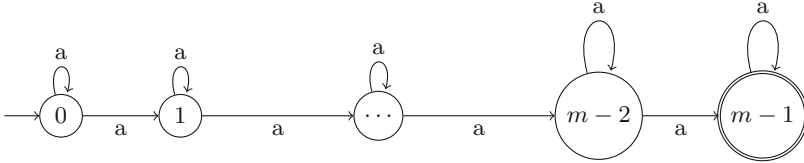
From Theorem 6 of [15] we know that if the density of a regular language  $L$  is non-polynomial, then  $\varrho_L(n) = 2^{\Omega(n)}$ . The gap between polynomial and exponential densities occurs because there do not exist any regular languages whose density functions contain non-integer exponents, e.g.  $\sqrt{n}$ , or  $2^{\sqrt{n}}$ . Together with Corollary 2 this gives:

**Corollary 3.** *If the cycle height of a DFA  $A$  is not finite then the density of  $L(A)$  is  $2^{\Omega(n)}$ .*

Corollary 2 and Theorem 1 would yield a polynomial time algorithm to compute the exact density of a regular language, however, the time complexity cannot compete with the algorithm of Gawrychowski et al. [3]. For an  $m$ -state DFA over a fixed alphabet, an algorithm based on our Theorem 1 to compute the degree of the polynomial giving the density of the language would require  $O(m^3)$  time. In comparison, for a DFA  $A$  over a fixed alphabet the algorithm given by Theorem 9 of Gawrychowski et al. [3] works in linear time and, even for an NFA  $A$ , the algorithm of Gawrychowski et al. [3] does the computation in time  $O(m^2)$ , where  $m$  is the number of states of  $A$ .

### 4 Depth Path Width of Finite Cycle Height NFAs

First we consider bounds for the number of transitions of an NFA with finite cycle height. The definition of NFAs with strict cycle height  $m$  implies that an NFA of the form given in Fig. 2 has the minimal number of transitions among all NFAs of strict cycle height  $m$ .



**Fig. 2.** Unary NFA with strict cycle height  $m$  having a minimal number of transitions

For NFAs with strict cycle height  $\mathcal{K}$ , the following Lemma gives bounds for the number of cycles and transitions as a function of the number of states.

**Lemma 4.** *If  $A = (Q, \Sigma, \delta, q_0, F)$  is an NFA with cycle height  $\mathcal{K}$ , then  $\mathcal{K} \leq |Q|$ , and  $|\delta| \leq \mathcal{K} + |\Sigma| \cdot \binom{|Q|}{2}$ . If  $A$  has strict cycle height  $\mathcal{K}$ , then we have also that  $2 \cdot \mathcal{K} - 1 \leq |\delta|$ .*

We examine the number of complete computations of a given length of a unary NFA  $A$  with strict cycle height  $\mathcal{K}$ , that is, the depth path width of  $A$ .

**Lemma 5.** *Let  $A_{\mathcal{K}} = (Q, \{a\}, \delta, q_0, F)$  be a unary NFA with strict cycle height  $\mathcal{K}$ . Then  $\text{DPW}(A_{\mathcal{K}}, \ell) \geq \sum_{i=0}^{\mathcal{K}-1} \binom{\ell}{i}$ ,  $\ell \in \mathbb{N}$ .*

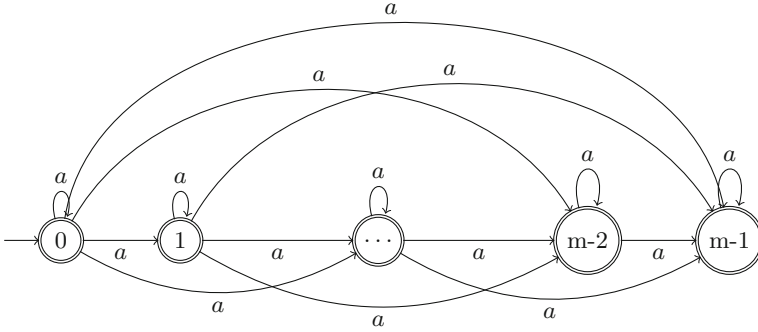
Next we compute an upper bound for the depth path width of  $\mathcal{K}$ -state unary NFAs having strict cycle height  $\mathcal{K}$ .

**Lemma 6.** *Let  $A_{\mathcal{K}}$  be a  $\mathcal{K}$ -state unary NFA with strict cycle height  $\mathcal{K}$ . Then for  $\ell \in \mathbb{N}$ ,  $\text{DPW}(A_{\mathcal{K}}, \ell) \leq \binom{\ell + \mathcal{K} - 1}{\mathcal{K} - 1}$ .*

*Proof.* Since  $A_{\mathcal{K}}$  has strict cycle height  $\mathcal{K}$ , each of the  $\mathcal{K}$  states must be part of a distinct cycle, that is, each state has a self-loop and no other transition can be part of a cycle. By Lemma 2 (ii) to get an upper bound for the depth path width, we can add to  $A_{\mathcal{K}}$  a maximal number of acyclic transition. That is, without loss of generality  $A_{\mathcal{K}}$  is as in Fig. 3 (with  $\mathcal{K} = m$ ).

It remains to compute the depth path width of  $A_{\mathcal{K}}$ . As the base case, we observe that  $\text{DPW}(A_1, \ell) = \binom{\ell + 1 - 1}{0} = \binom{\ell}{0} = 1$ . Inductively, we assume that the claim holds for  $A_{\mathcal{K}}$  and now the inductive claim is that

$$\text{DPW}(A_{\mathcal{K}+1}, \ell) = \binom{\ell + \mathcal{K}}{\mathcal{K}}$$



**Fig. 3.** An  $m$ -state unary NFA with strict cycle height  $m$  and maximal number of transitions

The value  $DPW(A_{\mathcal{K}+1}, \ell)$  counts the number of computations of length  $\ell$  that are in  $A_{\mathcal{K}}$ , as well as all of the computations with  $\ell - 1$  transitions from  $A_{\mathcal{K}}$  and one final transition,  $\delta(q, a) = \mathcal{K}$ , for some  $0 \leq q \leq \mathcal{K} - 1$ . More formally:

$$DPW(A_{\mathcal{K}+1}, \ell) = DPW(A_{\mathcal{K}}, \ell) + DPW(A_{\mathcal{K}+1}, \ell - 1) \tag{1}$$

Using our inductive assumption to replace the values in the right hand side of (1), we get:

$$DPW(A_{\mathcal{K}+1}, \ell) = \binom{\ell + \mathcal{K} - 1}{\mathcal{K} - 1} + \binom{\ell + \mathcal{K} - 1}{\mathcal{K}} = \binom{\ell + \mathcal{K}}{\mathcal{K}}$$

where the last equality uses Pascal’s triangle recursion rule [17]. □

Lemma 6 gives an upper bound for the depth path width of a  $\mathcal{K}$ -state unary NFA with strict cycle height  $\mathcal{K}$ . Next we consider the depth path width of unary NFAs with strict cycle height  $\mathcal{K}$  that have more than  $\mathcal{K}$  states.

**Lemma 7.** *Let  $A_{\mathcal{K}} = (Q, \{a\}, \delta, q_0, F)$  be an  $\mathcal{K}$ -state NFA with strict cycle height  $\mathcal{K}$  (as in Lemma 6). Let  $B_{\mathcal{K}+nc}$  be  $A_{\mathcal{K}}$  with one additional state that is not involved in any cycle. Then  $DPW(B_{\mathcal{K}+nc}, \ell) \leq DPW(A_{\mathcal{K}+1}, \ell)$ ,  $\ell \in \mathbb{N}$ .*

Since NFAs with finite strict cycle height will have maximal depth path width when they have the same number of states and cycles, the result from Lemma 6 is an upper bound for all  $\mathcal{K}$ -state NFAs with strict cycle height  $\mathcal{K}$ . Combining the results from these Lemmas, we observe the following corollary.

**Corollary 4.** *If  $A_{\mathcal{K}} = (Q, \Sigma, \delta, q_0, F)$  is a  $\mathcal{K}$ -state unary NFA with strict cycle height  $\mathcal{K}$ , then*

$$\sum_{i=0}^{\mathcal{K}-1} \binom{\ell}{i} \leq DPW(A_{\mathcal{K}}, \ell) \leq \binom{\ell + \mathcal{K} - 1}{\mathcal{K} - 1}$$

*The upper bound also holds for  $\mathcal{K}$ -state unary NFAs with cycle height  $\mathcal{K}$ . The lower bound also holds for  $\mathcal{K}$ -state non-unary NFAs with strict cycle height  $\mathcal{K}$ .*

The results of Sect. 4 give upper and lower bounds for the number of complete computations of NFAs whose number of states matches exactly their strict cycle height.

If the number of states exceeds the number of cycles, there will naturally be fewer complete computations. In this case, however, the depth path width of these NFAs cannot be just a function of the number of cycles, and must necessarily involve the number of states.

*Problem 1.* What is the maximum number of complete computations for an  $m$ -state NFA with strict cycle height  $\mathcal{K}$ , when  $m > \mathcal{K}$ ?

### 4.1 Experimental Results

To acquire results for the maximum number of complete computations of NFAs with strict finite cycle height and a non-unary alphabet, we first need the following definition.

**Definition 1** ([18]). *Pascal’s generalized triangle, denoted  $\mathcal{P}^N$ , is an extension of Pascal’s triangle, defined as:*

$$\mathcal{P}^N(0, 0) = \mathcal{P}^N(x, 0) = \mathcal{P}^N(0, y) = 1$$

$$\mathcal{P}^N(x, y) = \mathcal{P}^N(x - 1, y) + \mathcal{P}^N(x, y - 1) + (N \cdot \mathcal{P}^N(x - 1, y - 1)),$$

where  $\mathcal{P}^N(x, y)$  is the  $y^{\text{th}}$  element in the  $x^{\text{th}}$  row of the  $N^{\text{th}}$  generalized triangle, for  $x, y \geq 0$ . It is obvious that  $\mathcal{P}^0$  reduces to the normal Pascal’s triangle.

Recalling the result from Lemma 6 for unary machines, we can see easily that:

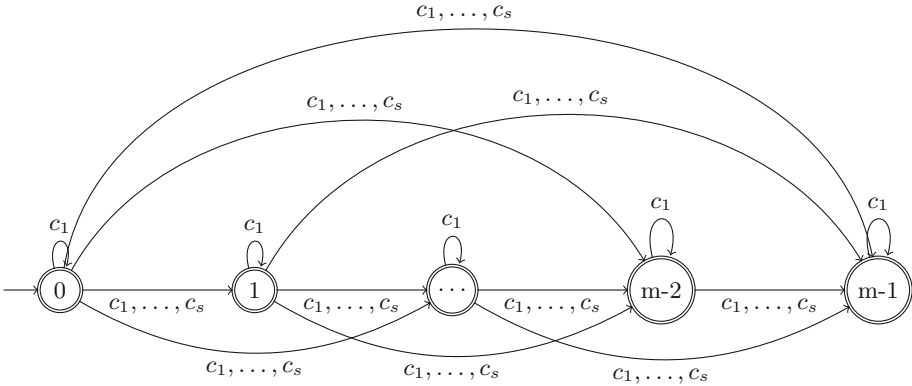
$$DPW(A_{\mathcal{K}}, \ell) = \mathcal{P}^0(\mathcal{K} - 1, \ell) = \binom{\ell + \mathcal{K} - 1}{\mathcal{K} - 1}$$

For NFAs with a binary alphabet we look to  $\mathcal{P}^1$ , which corresponds to the Delannoy numbers [1], and has the following closed form:

$$\mathcal{P}^1(x, y) = \sum_{i=0}^{\min(x,y)} 2^i \cdot \binom{x}{i} \cdot \binom{y}{i}$$

By extrapolating upon the closed form for the Delannoy numbers, we established a candidate equation upper bounding the number of complete computations of NFAs of the form given in Fig. 4. Based on experimental testing of NFAs of this form (for  $1 \leq \mathcal{K} \leq 7$ ,  $1 \leq |\Sigma| \leq 8$ , and  $1 \leq \ell \leq 10$ ), we believe that the following conjecture captures the number of complete computations as the length of the computation increases.

The selection of characters used on the self-loops (in the case of Fig. 4,  $c_1$ ) is arbitrary for the purposes of counting computations, and does not have to be the same for every state.



**Fig. 4.**  $\mathcal{K}$ -state NFA with strict cycle height  $\mathcal{K}$  and  $|\Sigma| = s$

*Conjecture 1.* Let  $A_{\mathcal{K}}^{|\Sigma|} = (Q, \Sigma, \delta, q_0, F)$  be a  $\mathcal{K}$ -state NFA with strict cycle height  $\mathcal{K}$ , as in Fig. 4. Then

$$DPW(A_{\mathcal{K}}^{|\Sigma|}, \ell) = \mathcal{P}^{|\Sigma|-1}(\mathcal{K} - 1, \ell) = \sum_{i=0}^{\min(\mathcal{K}-1, \ell)} |\Sigma|^i \cdot \binom{\mathcal{K} - 1}{i} \cdot \binom{\ell}{i} \quad (2)$$

Furthermore, since (2) and the structure of NFAs of the form given in Fig. 4 scale down to the unary case, we believe that this is an upper bound on the number of complete computations of any  $\mathcal{K}$ -state NFA with strict cycle height  $\mathcal{K}$ .

*Conjecture 2.* Let  $A_{\mathcal{K}}^{|\Sigma|} = (Q, \Sigma, \delta, q_0, F)$  be a  $\mathcal{K}$ -state NFA with strict cycle height  $\mathcal{K}$ . Then

$$DPW(A_{\mathcal{K}}^{|\Sigma|}, \ell) \leq \sum_{i=0}^{\min(\mathcal{K}-1, \ell)} |\Sigma|^i \cdot \binom{\mathcal{K} - 1}{i} \cdot \binom{\ell}{i} \quad (3)$$

**Acknowledgments.** Research supported by NSERC grant OGP0147224.

## References

1. Banderier, C., Schwer, S.R.: Why Delannoy numbers? *J. Statist. Plann. Inference* **135**, 40–54 (2004)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 3rd edn. MIT Press, Cambridge (2009)
3. Gawrychowski, P., Krieger, D., Rampersad, N., Shallit, J.: Finding the growth rate of a regular or context-free language in polynomial time. *Int. J. Found. Comput. Sci.* **21**(4), 597–618 (2010)
4. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptive complexity of machines with limited resources. *J. Univ. Comput. Sci.* **8**(2), 193–234 (2002)



5. Gruber, H.: Digraph complexity measures and applications in formal language theory. *Discrete Math. Theor. Comput. Sci.* **14**(2), 189–204 (2012)
6. Gruber, H., Holzer, M.: From finite automata to regular expressions and back - a summary on descriptonal complexity. *Int. J. Found. Comput. Sci.* **26**(8), 1009–1040 (2015)
7. Han, Y.-S., Salomaa, A., Salomaa, K.: Ambiguity, nondeterminism and state complexity of finite automata. *Acta Cybernetica* **23**, 141–157 (2017)
8. Holzer, M., Kutrib, M.: Descriptonal and computational complexity of finite automata - a survey. *Inform. Comput.* **209**(3), 456–470 (2011)
9. Keeler, C., Salomaa, K.: Branching measures and nearly acyclic NFAs. In: Pighizzini, G., C ampeanu, C. (eds.) DCFS 2017. LNCS, vol. 10316, pp. 202–213. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-60252-3\\_16](https://doi.org/10.1007/978-3-319-60252-3_16)
10. Kutrib, M., Pighizzini, G.: Recent trends in descriptonal complexity of formal languages. *Bull. EATCS* (111) (2013)
11. Kutrib, M., Meckel, K., Wendlandt, M.: Parameterized prefix distance between regular languages. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 419–430. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-04298-5\\_37](https://doi.org/10.1007/978-3-319-04298-5_37)
12. Msiska, M., van Zijl, L.: Interpreting the subset construction using finite sublanguages. In: *Proceedings of Prague Stringology Conference 2016*, pp. 48–62 (2016)
13. Ahuja, R.K., Mehlhorn, K., Orlin, J.B., Tarjan, R.E.: Faster algorithms for the shortest path problem. *J. ACM* **37**(2), 213–223 (1990)
14. Shallit, J.: *Second Course in Formal Languages and Automata Theory*. Cambridge University Press, New York (2009)
15. Szilard, A., Yu, S., Zhang, K., Shallit, J.: Characterizing regular languages with polynomial densities. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629, pp. 494–503. Springer, Heidelberg (1992). [https://doi.org/10.1007/3-540-55808-X\\_48](https://doi.org/10.1007/3-540-55808-X_48)
16. Tarjan, R.E.: Depth-first search and linear graph algorithms. *SIAM J. Comput.* **1**(2), 146–160 (1972)
17. Tattersall, J.J.: *Elementary Number Theory in Nine Chapters*, Cambridge University Press (2005)
18. Wong, C.K., Maddocks, T.W.: A generalized Pascal’s triangle. *Fibonacci Quart.* **13**(2), 134–136 (1975)



# Finite Automata with Undirected State Graphs

Martin Kutrib, Andreas Malcher<sup>(✉)</sup>, and Christian Schneider

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany  
{kutrib, andreas.malcher}@informatik.uni-giessen.de,  
Christian.Schneider@math.uni-giessen.de

**Abstract.** We investigate finite automata whose state graphs are undirected. This means that for any transition from state  $p$  to  $q$  consuming some letter  $a$  from the input there exists a symmetric transition from state  $q$  to  $p$  consuming a letter  $a$  as well. So, the corresponding language families are subregular and, in particular in the deterministic case, subreversible. In detail, we study the operational descriptiveness of deterministic and nondeterministic undirected finite automata. To this end, the different types of automata on alphabets with few letters are characterized. Then the operational state complexity of the Boolean operations as well as the operations concatenation and iteration is investigated, where tight upper and lower bounds are derived for unary as well as arbitrary alphabets under the condition that the corresponding language classes are closed under the operation considered.

## 1 Introduction

The operation problem for a language family is the question of costs (in terms of states) of operations on languages from this family with respect to their representations. More than two decades ago the operation problem for regular languages represented by deterministic finite automata as studied in [9, 10] renewed the interest in descriptiveness issues of finite automata in general. In the meantime, impressively many results have been obtained for a large number of language families. A recent survey of the several branches and details can be found in [2], which is also a valuable and comprehensive source of references. It seems that the recent studies of operational state complexity focus on subregular languages. Subregular language families of particular interest are the families of languages accepted by types of reversible finite automata. Reversibility is a fundamental principle in physics. Since abstract computational models with discrete internal states may serve as prototypes of computing devices which can physically be constructed, it is interesting to know whether these abstract models are able to obey physical laws. The observation that loss of information results in heat dissipation [7] strongly suggests to study computations without loss of information. Recent results on reversible finite automata can be found, for example, in [3–5, 8].

Here, we are interested in a strict form of reversible finite automata, namely, we do not only require that every state of the automaton has a unique predecessor for a given input letter, but that this predecessor can already be reached by a forward transition with the same input letter. These automata can be seen as finite automata whose state graphs are undirected. So, this notion is even stronger than the concept of time-symmetry studied in [1,6]. Time-symmetry appears in physical reality when a system can go back in time by applying the same transition function as for forward computations after a weak transformation of the phase-space. For example, in Newtonian mechanics one can go back in time by applying the same dynamics after a transformation that leaves masses and positions unchanged but reverses the sign of the momenta. While time-symmetric machines themselves cannot distinguish whether they run forward or backward in time, for undirected automata the time directions fade away since they are both available in the transition.

In the next section, we present the necessary notations and give an introductory example. Since the definition of undirected finite automata implies strong restrictions on the possible state graphs and, thus, the possible automata themselves, it is possible to characterize the different types of undirected automata with small alphabets in Sect. 3. These characterizations are a powerful tool to derive tight bounds on the operational state complexity of deterministic (Sect. 4) and nondeterministic (Sect. 5) undirected finite automata. All bounds obtained are summarized in Table 1. Finally, in Sect. 6 we discuss open and untouched problems for future work. We remark that some proofs are omitted due to space constraints.

## 2 Preliminaries

Let  $\Sigma^*$  denote the set of all words over the finite alphabet  $\Sigma$ . The *empty word* is denoted by  $\lambda$ , and  $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ . The *reversal* of a word  $w$  is denoted by  $w^R$ . For the *length* of  $w$  we write  $|w|$ . For the number of occurrences of a symbol  $a$  in  $w$  we use the notation  $|w|_a$ . Set inclusion is denoted by  $\subseteq$  and strict set inclusion by  $\subset$ . We write  $2^S$  for the power set and  $|S|$  for the cardinality of a set  $S$ .

A *nondeterministic finite automaton* (NFA) is a system  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , where  $Q$  is the finite set of *internal states*,  $\Sigma$  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  $\delta : Q \times \Sigma \rightarrow 2^Q$  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  $\delta : Q \times \Sigma \rightarrow Q$ .

So, by definition, 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.

If the state graph induced by some finite automaton is undirected then we obtain the subclasses of *nondeterministic undirected finite automata* (NUDFA)

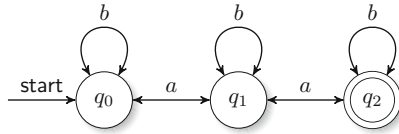
and *deterministic undirected finite automata* (DUDEFA). Formally, for undirected finite automata it is required that  $q \in \delta(p, a)$  if and only if  $p \in \delta(q, a)$ , for all  $p, q \in Q$  and  $a \in \Sigma$ . The *language accepted* by a finite automaton  $M$  is

$$L(M) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\},$$

where the transition function is recursively extended to  $\delta: Q \times \Sigma^* \rightarrow 2^Q$ .

In order to illustrate the definitions we continue with an example.

*Example 1.* The NUDFA  $M = \langle \{q_0, q_1, q_3\}, \{a, b\}, \delta, q_0, \{q_2\} \rangle$  whose transition function is given through the state graph depicted in Fig. 1 accepts the language  $\{w \in \{a, b\}^+ \mid |w|_a \bmod 2 = 0\}$ . ■



**Fig. 1.** State graph of an nondeterministic undirected finite automaton accepting the language  $\{w \in \{a, b\}^+ \mid |w|_a \bmod 2 = 0\}$ .

### 3 Characterization of Undirected Finite Automata with Small Alphabets

The definition of undirected finite automata implies strong restrictions on the possible state graphs and, thus, the possible automata themselves. So, there are only a very few different types of *minimal* deterministic undirected finite automata with a unary or binary input alphabet. The situation for nondeterministic devices is still rather restricted, but there are more types distinguishable. The following characterizations of possible types of state graphs is used for later results on the closure properties of the families of accepted languages as well as their operational state complexity. In the rest of the section, we tacitly assume  $\Sigma = \{a\}$  for unary languages and  $\Sigma = \{a, b\}$  for binary languages.

#### 3.1 Unary Nondeterministic Undirected Finite Automata

We start with unary NUDFAs.

**Theorem 2.** *Let  $M$  be a unary NUDFA. Then, language  $L(M)$  is of one of the following types:*

1.  $L_1 = \emptyset$ ,
2.  $L_2 = \{\lambda\}$ ,
3.  $L_{3,k} = \{a^n \mid n \geq k \text{ and even}\}$ , for some  $k \geq 0$ ,
4.  $L_{4,k} = \{a^n \mid n \geq k \text{ and } n \text{ odd}\}$ , for some  $k \geq 0$ ,

5.  $L_{5,k,l} = \{a^n \mid n \geq k \text{ and } n \text{ even for } k \leq n < l\}$ , for some  $k \geq 0$  and  $l > k$ ,
6.  $L_{6,k,l} = \{a^n \mid n \geq k \text{ and } n \text{ odd for } k \leq n < l\}$  for some  $k \geq 0$  and  $l > k$ .

*Proof.* Let  $M = \langle Q, \{a\}, \delta, q_0, F \rangle$  be an NUDFA with unary input alphabet. Basically, the type of the accepted language is determined by two numbers (if they exist)  $k \geq 0$  and  $l > k$ . The number  $k$  is defined to be the shortest path from the initial state  $q_0$  to some accepting state  $q_+ \in F$ . If  $k$  exists, that is, if  $L(M)$  is non-empty, then  $l$  is defined dependent on whether  $k$  is even or odd. If  $k$  is even,  $l$  is defined to be the shortest path longer than  $k$  that leads from the initial state  $q_0$  to an accepting state  $q'_+$  and has odd length. If such a path does not exist,  $l$  remains undefined. Similarly, if  $k$  is odd then  $l$  must be even.

Now, we can determine the type of  $L(M)$ . If there is no path from  $q_0$  to some accepting state then  $L(M) = L_1 = \emptyset$ . Otherwise the number  $k$  is defined.

If  $k = 0$  and, thus,  $q_0 \in F$  then  $\lambda \in L(M)$ . If additionally  $\delta(q_0, a)$  is undefined then  $L(M) = L_2 = \{\lambda\}$ .

Next, assume that  $k$  is defined,  $\delta(q_0, a)$  is defined, and  $l$  is undefined. Since  $M$  is undirected we obtain  $q_0 \in \delta(q_0, a^2)$ . We conclude that  $a^k$  belongs to  $L(M)$  and, for all  $i \geq 0$ , the input  $a^{k+2i}$  belongs to  $L(M)$  as well. Since  $l$  is undefined there are no words whose parity is different from  $a^k$  in  $L(M)$ . Therefore, if  $k$  is even we derive  $L(M) = L_{3,k} = \{a^n \mid n \geq k \text{ and } n \text{ even}\}$ . Similarly, if  $k$  is odd, we have  $L(M) = L_{4,k} = \{a^n \mid n \geq k \text{ and } n \text{ odd}\}$ .

Finally, let  $k$ ,  $\delta(q_0, a)$ , and  $l$  be defined. As before we may conclude that the input  $a^{k+2i}$  belongs to  $L(M)$ , for all  $i \geq 0$ . On the other hand, since  $l$  is defined and  $l > k$  we also have  $a^{l+2i} \in L(M)$ , for all  $i \geq 0$ . The parities of  $k$  and  $l$  are different. Therefore, all words  $a^{l+i}$ , for  $i \geq 0$ , belong to  $L(M)$ . This implies  $L(M) = L_{5,k,l} = \{a^n \mid n \geq k \text{ and } n \text{ even for } k \leq n < l\}$  if  $k$  is even, and  $L(M) = L_{6,k,l} = \{a^n \mid n \geq k \text{ and } n \text{ odd for } k \leq n < l\}$  if  $k$  is odd.  $\square$

Note that the languages  $\{a^n \mid n \geq k\}$  are equal to  $L_{5,k,k+1}$ , if  $k$  is even, and equal to  $L_{6,k,k+1}$ , if  $k$  is odd. Next, we turn to the minimal numbers of states that are necessary for some NUDFA to accept the languages of different type.

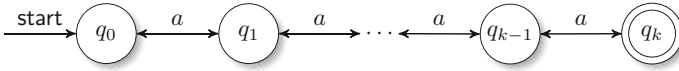
**Lemma 3.** *Let  $M$  be a unary NUDFA that accepts a language of type  $L_1$  or  $L_2$ . Then one state is sufficient and necessary for  $M$ .*

**Lemma 4.** *Let  $k \geq 1$  and  $M$  be a unary NUDFA that accepts a language of type  $L_{3,k}$  or  $L_{4,k}$ . Then  $k + 1$  states are sufficient and necessary for  $M$ .*

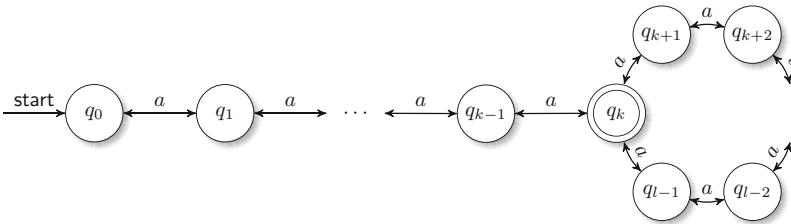
*Proof.* Since the shortest word accepted has length  $k$ , automaton  $M$  must have at least  $k + 1$  states. Otherwise,  $L(M)$  would be empty or a shorter word would be accepted. The NUDFA depicted in Fig. 2 accepts  $L_{3,k}$  or  $L_{4,k}$  with  $k + 1$  states.  $\square$

**Lemma 5.** *Let  $k \geq 0$ ,  $l > k$ , and  $M$  be a unary NUDFA that accepts a language of type  $L_{5,k,l}$  or  $L_{6,k,l}$ . Then  $l$  states are sufficient and necessary for  $M$ .*

The NUDFA depicted in Fig. 3 accepts  $L_{5,k,l}$  or  $L_{6,k,l}$  with  $l$  states.



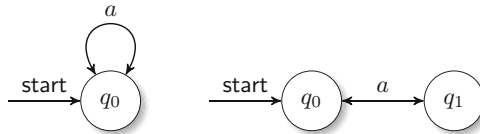
**Fig. 2.** State graph of a minimal nondeterministic undirected finite automaton accepting the language  $L_{3,k}$  or  $L_{4,k}$ .



**Fig. 3.** State graph of a minimal nondeterministic undirected finite automaton accepting the language  $L_{5,k,l}$  or  $L_{6,k,l}$ .

### 3.2 Unary and Binary Deterministic Undirected Finite Automata

The situation for unary deterministic undirected finite automata is straightforward. The only two possible state graphs are depicted in Fig. 4. Any of the states can be made accepting or rejecting which yields four different languages that are accepted by unary DUDFAs.



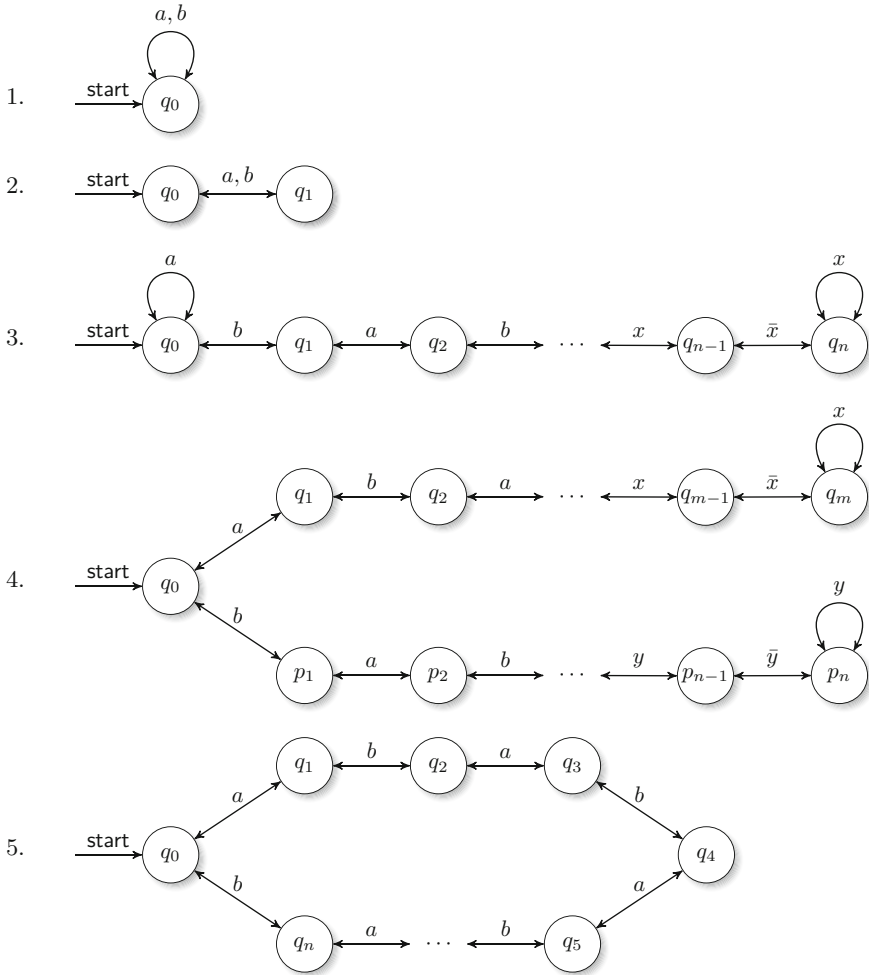
**Fig. 4.** Possible state graphs of unary deterministic undirected finite automata.

**Corollary 6.** *Let  $M$  be a unary DUDFA. Then language  $L(M)$  is one of the following:*

1.  $L_1 = \emptyset$ ,
2.  $L_2 = a^*$ ,
3.  $L_3 = \{a^n \mid n \text{ is even}\}$ ,
4.  $L_4 = \{a^n \mid n \text{ is odd}\}$ .

While the family of binary languages accepted by NUDFAs is fairly rich, in the following the types of possible state graphs of deterministic undirected finite automata accepting binary languages are analyzed.

**Theorem 7.** Let  $M$  be a minimal binary DUDFA. Then its state graph is of one of the five types depicted in Fig. 5, where  $m, n \geq 1$ , the letters  $a$  and  $b$  are interchangeable,  $x, y \in \{a, b\}$ ,  $\bar{x} = a \iff x = b$ , and  $\bar{y} = a \iff y = b$ .



**Fig. 5.** Possible state graphs of binary deterministic undirected finite automata, where  $m, n \geq 1$ , the letters  $a$  and  $b$  are interchangeable,  $x, y \in \{a, b\}$ ,  $\bar{x} = a \iff x = b$ , and  $\bar{y} = a \iff y = b$ .

### 4 Deterministic State Complexity

In this section, we investigate the state complexity of DUDFAs. We start with such automata on unary alphabets. It has been shown in Corollary 6 that in this

case the automata have an easy form which makes it possible to show closure under the Boolean operations and reversal, and to establish upper and lower bounds of two states for each of the operations. In the second part of the section, we investigate the state complexity of the Boolean operations for DUDFAs over arbitrary alphabets and can establish tight bounds as well.

**Theorem 8.** *For any integers  $m, n \geq 1$  let  $A$  be an  $m$ -state and  $B$  be an  $n$ -state DUDFA over the same unary alphabet. Then two states are sufficient and necessary in the worst case for a DUDFA to accept  $\overline{L(A)}$ ,  $L(A) \cap L(B)$ ,  $L(A) \cup L(B)$ , or  $L(A)^R$ .*

Now, we turn to arbitrary alphabets and consider first the complementation of DUDFAs.

**Theorem 9.** *For any integer  $n \geq 1$  let  $A$  be an  $n$ -state DUDFA. Then  $n$  states are sufficient and necessary in the worst case for a DUDFA to accept the language  $\overline{L(A)}$ .*

Next, we continue with the intersection operation. To this end, we provide the following preparatory lemma which is needed to prove a tight lower bound.

**Lemma 10.** *Let  $m \geq 4$  be an even integer. There are natural numbers  $x_1, x_2, y_1,$  and  $y_2$  such that the following equations hold simultaneously. Moreover, both equations take on the minimal value for  $x_1, x_2, y_1,$  and  $y_2$ .*

$$m - 1 + 2mx_1 = 1 + (2m - 2)y_1, \tag{1}$$

$$m + 1 + 2mx_2 = 2m - 3 + (2m - 2)y_2. \tag{2}$$

**Theorem 11.** *For any integers  $m, n \geq 1$  let  $A$  be an  $m$ -state and  $B$  be an  $n$ -state DUDFA. Then  $m \cdot n$  states are sufficient for a DUDFA to accept the language  $L(A) \cap L(B)$ .*

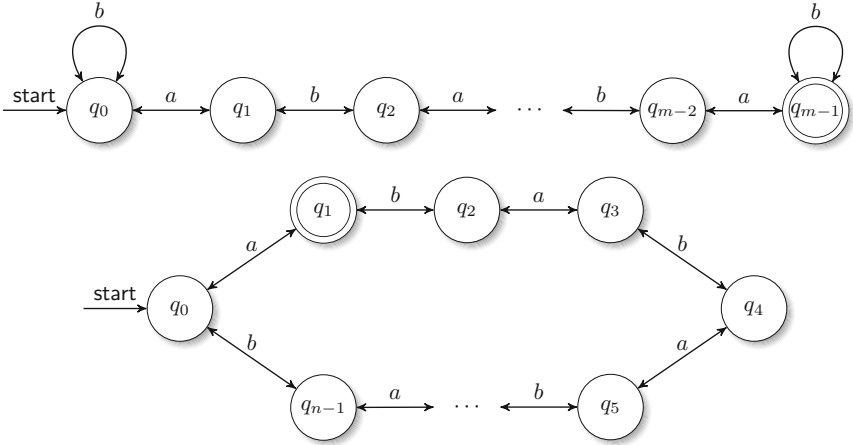
*Proof.* For the upper bound we use the usual cross-product construction. Let  $A = \langle Q_A, \Sigma, \delta_A, q_{0,A}, F_A \rangle$  be an  $m$ -state DUDFA and  $B = \langle Q_B, \Sigma, \delta_B, q_{0,B}, F_B \rangle$  be an  $n$ -state DUDFA. Then define  $C = \langle Q_A \times Q_B, \Sigma, \delta, (q_{0,A}, q_{0,B}), F_A \times F_B \rangle$ , where  $\delta((q_1, q_2), a) = (\delta_A(q_1, a), \delta_B(q_2, a))$  for all  $q_1 \in Q_A, q_2 \in Q_B,$  and  $a \in \Sigma$ . Clearly,  $C$  is an  $(m \cdot n)$ -state DUDFA that accepts  $L(A) \cap L(B)$ .  $\square$

**Theorem 12.** *There are infinitely many integers  $m, n \geq 1$  with  $n = 2m - 2$  such that  $A$  is an  $m$ -state DUDFA,  $B$  is an  $n$ -state DUDFA, and  $m \cdot n$  states are necessary for any DUDFA to accept the language  $L(A) \cap L(B)$ .*

*Proof.* For the lower bound we consider two minimal DUDFAs  $A$  and  $B$ , where  $A$  has  $m$  states for even  $m \geq 4$  and  $B$  has  $n = 2m - 2$  states. Both DUDFAs have the form depicted in Fig. 6.

Now, let  $C$  be a minimal DUDFA accepting  $L(A) \cap L(B)$ . First, consider words in  $L(A) \cap L(B)$  that have the form  $(ab)^\ell a$  for some  $\ell \geq 1$ . Then the left hand side of Eq. (1) indicates the length of such words that are accepted





**Fig. 6.** The DUFA  $A$  (upper automaton) and  $B$  (lower automaton) used for proof of the lower bound.

by  $A$ , whereas the right hand side of Eq. (1) indicates the length of such words that are accepted by  $B$ . According to Lemma 10 there are integers  $x_1 = m/2$ ,  $x_2 = m/2 - 2$ ,  $y_1 = m/2 + 1$ , and  $y_2 = m/2 - 2$  such that Eqs. (1) and (2) hold simultaneously while assuming a minimal value. Hence, due to the equality and the minimality we obtain a word  $(ab)^{(m^2+m-2)/2}a$  of minimal length  $m^2 + m - 1$  which belongs to  $L(A) \cap L(B)$ . Analogously, considering words in  $L(A) \cap L(B)$  of the form  $(ba)^\ell b$  for some  $\ell \geq 1$  the left hand side and right hand side of Eq. (2) provides a word  $(ba)^{(m^2-3m)/2}b$  of minimal length  $m^2 - 3m + 1$  which belongs to  $L(A) \cap L(B)$  as well. Since both minimal lengths are different, automaton  $C$  has to have two different paths from the initial state to an accepting state which implies, according to Theorem 7, that  $C$  is of type 4 or type 5. Let us first assume that  $C$  is of type 4. Then, in particular the upper path contains an accepting state  $f$  that is entered after reading  $w = (ab)^{(m^2+m-2)/2}a$  and the upper path ends in some state  $q$  that ends in a loop. If  $f = q$ , then there is a transition from  $f$  to  $f$  on input  $a$  which implies that input  $wa$  is accepted by  $C$ . This is a contradiction, since  $wa$  is not accepted by  $B$ . If  $f \neq q$ , then there are two possibilities, namely, the remaining path from  $f$  to  $q$  processes inputs of the form  $(ba)^k b^*$  or of the form  $(ba)^{k-1}ba^*$  for some  $k \geq 1$ . In either case we can construct a word  $w'$  which is accepted by  $C$ , but not by  $B$  which gives the contradiction. In the first case, we consider  $w' = w(ba)^k b(ab)^k$  and in the second case we set  $w' = w(ba)^{k-1}ba(ba)^{k-1}b$ . Hence, we conclude that  $C$  is of type 5. This means that  $C$  has two different paths which start in the initial state and end in the same state. Thus,  $C$  has at least  $m^2 + m + m^2 - 3m + 2 - 2 = 2m^2 - 2m = m(2m - 2) = m \cdot n$  states.  $\square$

Finally, we study the union operation and obtain the same tight bound as for intersection. For the upper bound we can again use the usual cross-product construction as in the proof of Theorem 11.

**Theorem 13.** *For any integers  $m, n \geq 1$  let  $A$  be an  $m$ -state and  $B$  be an  $n$ -state DUDFA. Then  $m \cdot n$  states are sufficient for a DUDFA to accept the language  $L(A) \cup L(B)$ .*

**Theorem 14.** *There are infinitely many integers  $m, n \geq 1$  with  $n = 2m - 2$  such that  $A$  is an  $m$ -state DUDFA,  $B$  is an  $n$ -state DUDFA, and  $m \cdot n$  states are necessary for any DUDFA to accept the language  $L(A) \cup L(B)$ .*

*Proof.* For the lower bound we consider two DUDFAs  $A'$  and  $B'$  accepting the complements of the languages accepted by  $A$  and  $B$  used in the proof of Theorem 12. Both DUDFAs can be obtained due to Theorem 9 by interchanging accepting and non-accepting states. The resulting DUDFAs are minimal, have  $m$  and  $n$  states, respectively,  $L(A') = \overline{L(A)}$  and  $L(B') = \overline{L(B)}$ . Now, let  $C'$  be a minimal DUDFA accepting  $L(A') \cup L(B')$  and assume that  $C'$  has  $\ell < m \cdot n$  states. By applying Theorem 9 we can then construct an  $\ell$ -state DUDFA  $C$  such that  $L(C) = \overline{L(C')} = \overline{L(A') \cup L(B')} = \overline{L(A')} \cap \overline{L(B')} = L(A) \cap L(B)$ . This is a contradiction to the proof of Theorem 12 where it is shown that every DUDFA accepting  $L(A) \cap L(B)$  needs at least  $m \cdot n$  states.  $\square$

## 5 Nondeterministic State Complexity

In this section, we complement the state complexity results by investigating NUDFAs. We start again with such automata on unary alphabets. Owing to the characterization given in Theorem 2 we can show tight bounds by a detailed case analysis. If the underlying alphabet is at least binary then NUDFAs are only closed under intersection. However, it is possible to obtain tight bounds as well. We start with the unary case.

**Theorem 15.** *For any integers  $m, n \geq 1$  let  $A$  be an  $m$ -state and  $B$  be an  $n$ -state NUDFA over the same unary alphabet. Then  $\max(m, n) + 1$  states are sufficient and necessary in the worst case for an NUDFA to accept  $L(A) \cap L(B)$ .*

*Proof.* According to Theorem 2,  $L(A)$  and  $L(B)$  belong to one of the types  $L_1$ ,  $L_2$ ,  $L_{3,k}$ ,  $L_{4,k}$ ,  $L_{5,k,l}$ , or  $L_{6,k,l}$  for some  $k \geq 0$  and  $l > k$ . Hence, we have to show that each combination leads to an NUDFA with at most  $\max(m, n) + 1$  states. The results of all combinations are summarized in the following table, where  $k = \max(k_1, k_2)$ ,  $l = \max(l_1, l_2)$ ,  $L_{7,k_1,l_1,k_2,l_2} = L_{5,\max(k_1,l_2),l_1}$ , if  $l_1 > l_2$ ,  $L_{7,k_1,l_1,k_2,l_2} = L_{6,\max(k_2,l_1),l_2}$ , if  $l_1 \leq l_2$ , and  $L_{8,k_2} = L_2$ , if  $k_2 = 0$ , and  $L_{8,k_2} = L_1$  otherwise.

$\cap$	$L_1$	$L_2$	$L_{3,k_1}$	$L_{4,k_1}$	$L_{5,k_1,l_1}$	$L_{6,k_1,l_1}$
$L_{6,k_2,l_2}$	$L_1$	$L_1$	$L_{3,\max(k_1,l_2)}$	$L_{4,k}$	$L_{7,k_1,l_1,k_2,l_2}$	$L_{6,k,l}$
$L_{5,k_2,l_2}$	$L_1$	$L_{8,k_2}$	$L_{3,k}$	$L_{4,\max(k_1,l_2)}$	$L_{5,k,l}$	$L_{7,k_2,l_2,k_1,l_1}$
$L_{4,k_2}$	$L_1$	$L_1$	$L_1$	$L_{4,k}$	$L_{4,\max(k_2,l_1)}$	$L_{4,k}$
$L_{3,k_2}$	$L_1$	$L_{8,k_2}$	$L_{3,k}$	$L_1$	$L_{3,k}$	$L_{3,\max(l_1,k_2)}$
$L_2$	$L_1$	$L_2$	$L_{8,k_1}$	$L_1$	$L_{8,k_1}$	$L_1$
$L_1$	$L_1$	$L_1$	$L_1$	$L_1$	$L_1$	$L_1$

We will not consider all combinations in detail, but exemplarily discuss two intersections. First, we consider  $L_{3,k_1} \cap L_{6,k_2,l_2}$  which is

$$\begin{aligned} & \{a^n \mid n \geq k_1 \text{ and } n \text{ even}\} \cap \{a^n \mid n \geq k_2 \text{ and } n \text{ odd for } k_2 \leq n < l_2\} \\ &= \{a^n \mid n \geq k_1 \text{ and } n \text{ even and } n \geq l_2\} = L_{3,\max(k_1,l_2)}. \end{aligned}$$

Second, we consider  $L_{5,k_1,l_1} \cap L_{6,k_2,l_2}$  where necessarily  $l_1$  is odd and  $l_2$  is even. If  $l_1 > l_2$ ,

$$\begin{aligned} L_{5,k_1,l_1} \cap L_{6,k_2,l_2} &= \{a^n \mid n \geq k_1 \text{ and } n \text{ even for } k_1 \leq n < l_1\} \cap \{a^n \mid n \geq l_2\} \\ &= \{a^n \mid n \geq \max(k_1, l_2) \text{ and } n \text{ even for } k_1 \leq n < l_1\} \\ &= L_{5,\max(k_1,l_2),l_1}. \end{aligned}$$

Otherwise, if  $l_1 < l_2$ ,

$$\begin{aligned} L_{5,k_1,l_1} \cap L_{6,k_2,l_2} &= \{a^n \mid n \geq l_1\} \cap \{a^n \mid n \geq k_2 \text{ and } n \text{ odd for } k_2 \leq n < l_2\} \\ &= \{a^n \mid n \geq \max(k_2, l_1) \text{ and } n \text{ odd for } k_2 \leq n < l_2\} \\ &= L_{6,\max(k_2,l_1),l_2}. \end{aligned}$$

For the upper bound, we can read off the table the following cases. If  $L_1$ ,  $L_2$ , or  $L_{8,t}$  is obtained, we know that one state is sufficient to accept the languages. Hence,  $1 \leq \max(m, n) + 1$  is an upper bound. If  $L_{3,t}$  or  $L_{4,t}$  is obtained, we know that  $t + 1$  states are sufficient to accept the languages. Now, we have  $t \in \{k_1, k_2, l_1, l_2\}$ . Since  $k_1 < l_1 = m$  and  $k_2 < l_2 = n$ , we obtain that  $k_1 + 1 \leq m - 1 + 1 \leq \max(m, n) + 1$ ,  $k_2 + 1 \leq n - 1 + 1 \leq \max(m, n) + 1$ ,  $l_1 + 1 = m + 1 \leq \max(m, n) + 1$ , and  $l_2 + 1 = n + 1 \leq \max(m, n) + 1$ . Thus,  $\max(m, n) + 1$  is an upper bound.

Finally, if  $L_{5,t_1,t_2}$  or  $L_{6,t_1,t_2}$  is obtained, we know that  $t_2$  states are sufficient to accept the languages. Now, we have  $t_2 \in \{l_1, l_2, \max(l_1, l_2)\}$ . Since  $l_1 = m$  and  $l_2 = n$ , we obtain that  $l_1 = m \leq \max(m, n) + 1$ ,  $l_2 = n \leq \max(m, n) + 1$ , and  $\max(l_1, l_2) = \max(m, n) \leq \max(m, n) + 1$ . Thus,  $\max(m, n) + 1$  is an upper bound also for these cases as well.

For the lower bound, we first consider  $L_{3,k_1} \cap L_{6,k_2,l_2} = L_{3,\max(k_1,l_2)}$ . We know that  $L_{3,k_1}$  is accepted with  $k_1 + 1 = m$  states owing to Lemma 4 and  $L_{6,k_2,l_2}$  is accepted with  $l_2 = n$  states owing to Lemma 5. Moreover, to accept  $L_{3,\max(k_1,l_2)}$  at least  $\max(k_1, l_2) + 1 = \max(m - 1, n) + 1$  states are necessary due to Lemma 4.

Second, consider the symmetric case  $L_{6,k_1,l_1} \cap L_{3,k_2} = L_{3,\max(l_1,k_2)}$  with  $m = l_1$  and  $n = k_2 + 1$ . Then  $\max(m, n - 1) + 1$  states are necessary to accept  $L_{3,\max(l_1,k_2)}$ . Altogether,  $\max(\max(m - 1, n) + 1, \max(m, n - 1) + 1) = \max(m, n) + 1$  is the lower bound.  $\square$

**Theorem 16.** *For any integers  $m, n \geq 1$  let  $A$  be an  $m$ -state and  $B$  be an  $n$ -state NUDFA over the same unary alphabet. Then  $m + n - 1$  states are sufficient and necessary in the worst case for an NUDFA to accept  $L(A) \cdot L(B)$ .*

Since the reversal of a unary language  $L$  is the language  $L$  again, the following statement is obvious.

**Theorem 17.** *For any integer  $m \geq 1$ , let  $A$  be an  $m$ -state unary NUDFA. Then  $m$  states are sufficient and necessary in the worst case for an NUDFA to accept  $L(A)^R$ .*

Finally, we consider the general intersection operation for NUDFAs.

**Theorem 18.** *For any integers  $m, n \geq 1$  let  $A$  be an  $m$ -state and  $B$  be an  $n$ -state NUDFA. Then  $m \cdot n$  states are sufficient and necessary in the worst case for an NUDFA to accept the language  $L(A) \cap L(B)$ .*

The results on the deterministic and nondeterministic state complexity obtained are summarized in Table 1.

**Table 1.** Summary of the state complexities of the operations studied in this paper. It is marked by — if an automata class is not closed under the corresponding operation.

	unary DUDFA	DUDFA	unary NUDFA	NUDFA
$L_1 \cup L_2$	2	$mn$	—	—
$L_1 \cap L_2$	2	$mn$	$\max(m, n) + 1$	$mn$
$\bar{L}$	2	$m$	—	—
$L_1 L_2$	—	—	$m + n - 1$	—
$L^*$	—	—	—	—
$L^R$	2	?	$m$	—

## 6 Conclusions

In this paper, we have introduced deterministic and nondeterministic finite automata with undirected state graphs. It was possible to characterize the languages accepted by such automata in the unary case for deterministic and nondeterministic automata as well as in the binary case for deterministic automata. This characterization enabled us to study the deterministic and nondeterministic operational state complexity in depth and an almost complete picture with tight

bounds could be obtained. The deterministic state complexity of the reversal operation as well as the question of whether the language class is closed under reversal are currently open questions. For DUDFAs with one accepting state the construction of a DUDFA accepting the reversal is straightforward and needs no additional states. However, the construction cannot directly be generalized to DUDFAs with more than one accepting state.

Concerning the operational state complexity we have investigated so far only those operations under which the corresponding language classes are closed. However, even in the case of non-closure we still obtain regular languages. Thus, it would clearly be of interest to determine upper and lower bounds for the remaining operations from this point of view.

Since the language classes accepted by DUDFAs and NUDFAs are subregular, it would be interesting to devise an effective algorithm that decides, given an arbitrary finite automaton  $A$ , whether or not  $L(A)$  could be accepted by an NUDFA or DUDFA as well. In the positive case, such an algorithm should construct an equivalent NUDFA or DUDFA. Then the determination of the exact trade-off concerning the number of states between NUDFAs and DUDFAs as well as arbitrary NFAs and DFAs becomes a challenging task.

## References

1. Gajardo, A., Kari, J., Moreira, A.: On time-symmetry in cellular automata. *J. Comput. Syst. Sci.* **78**, 1115–1126 (2012)
2. Gao, Y., Moreira, N., Reis, R., Yu, S.: A survey on operational state complexity. *J. Autom. Lang. Comb.* **21**, 251–310 (2016)
3. Holzer, M., Jakobi, S., Kutrib, M.: Minimal reversible deterministic finite automata. In: Potapov, I. (ed.) *DLT 2015*. LNCS, vol. 9168, pp. 276–287. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-21500-6\\_22](https://doi.org/10.1007/978-3-319-21500-6_22)
4. Holzer, M., Kutrib, M.: Reversible nondeterministic finite automata. In: Phillips, I., Rahaman, H. (eds.) *RC 2017*. LNCS, vol. 10301, pp. 35–51. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-59936-6\\_3](https://doi.org/10.1007/978-3-319-59936-6_3)
5. Kutrib, M.: Reversible and irreversible computations of deterministic finite-state devices. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) *MFCS 2015*. LNCS, vol. 9234, pp. 38–52. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48057-1\\_3](https://doi.org/10.1007/978-3-662-48057-1_3)
6. Kutrib, M., Worsch, T.: Time-symmetric machines. In: Dueck, G.W., Miller, D.M. (eds.) *RC 2013*. LNCS, vol. 7948, pp. 168–181. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38986-3\\_14](https://doi.org/10.1007/978-3-642-38986-3_14)
7. Landauer, R.: Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.* **5**, 183–191 (1961)
8. Lavado, G.J., Pighizzini, G., Prigioniero, L.: Weakly and strongly irreversible regular languages. In: Csuhaj-Varjú, E., Dömösi, P., Vaszil, G. (eds.) *Automata and Formal Languages (AFL 2017)*. EPTCS, vol. 252, pp. 143–156 (2017)
9. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages*, Chap. 2, vol. 1, pp. 41–110. Springer, Heidelberg (1997). [https://doi.org/10.1007/978-3-642-59136-5\\_2](https://doi.org/10.1007/978-3-642-59136-5_2)
10. Yu, S.: State complexity of regular languages. *J. Autom. Lang. Comb.* **6**, 221–234 (2001)



# Further Closure Properties of Input-Driven Pushdown Automata

Alexander Okhotin<sup>1</sup> and Kai Salomaa<sup>2</sup>(✉)

<sup>1</sup> St. Petersburg State University, 7/9 Universitetskaya nab.,  
Saint Petersburg 199034, Russia  
alexander.okhotin@spbu.ru

<sup>2</sup> School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada  
ksalomaa@cs.queensu.ca

**Abstract.** The paper investigates the closure of the language family defined by input-driven pushdown automata (IDPDA) under the following operations: insertion  $ins(L, K) = \{xyz \mid xz \in L, y \in K\}$ , deletion  $del(L, K) = \{xz \mid xyz \in L, y \in K\}$ , square root  $\sqrt{L} = \{w \mid ww \in L\}$ , and the first half  $\frac{1}{2}L = \{u \mid \exists v : |u| = |v|, uv \in L\}$ . For  $K$  and  $L$  recognized by nondeterministic IDPDA, with  $m$  and with  $n$  states, respectively, insertion requires  $mn + 2m$  states, as long as  $K$  is well-nested; deletion is representable with  $2n$  states, for well-nested  $K$ ; square root requires  $n^3 - O(n^2)$  states, for well-nested  $L$ ; the well-nested subset of the first half is representable with  $2^{O(n^2)}$  states. Without the well-nestedness constraints, non-closure is established in each case.

## 1 Introduction

*Input-driven pushdown automata* (IDPDA), also known under the name of *visibly pushdown automata*, are an important special class of pushdown automata, introduced by Mehlhorn [16] and later studied, in particular, by Alur and Madhusudan [1, 2]. In these automata, the input symbol determines whether the automaton should push a stack symbol, pop a stack symbol or leave the stack untouched. These symbols are called *left brackets*, *right brackets* and *neutral symbols*, and the symbol pushed at each left bracket is always popped when reading the corresponding right bracket. Input-driven automata are important as a model of hierarchically structured data, such as XML documents or computation traces for recursive procedure calls.

Input-driven automata exist in deterministic (DIDPDA) and nondeterministic (NIDPDA) variants, which, as shown by von Braunmühl and Verbeek [3], are equivalent in power. Input-driven automata are also notable for their appealing closure properties, which almost rival those of finite automata. For instance, they are closed under all Boolean operations, concatenation, Kleene

---

A. Okhotin—Supported by Russian Science Foundation, project 18-11-00100.

star, reversal [1,21], quotient [23] and edit distance neighbourhood [22] (for binary operations it is assumed that the automata defining its two arguments use the same partition of the alphabet into classes). Besides the closure results, the descriptive complexity of these operations has also been estimated in the literature. For instance, the concatenation of an  $m$ -state and an  $n$ -state DIDPDA in the worst case requires a DIDPDA with  $m2^{\Theta(n \log n)}$  states [21], whereas both for the union and for the intersection its state complexity is  $\Theta(mn)$  [24]. For NIDPDA, the state complexity of concatenation and of union is  $m+n+O(1)$  [1], and for the intersection it is  $\Theta(mn)$  [10]. Further state complexity results were established for an intermediate family of *unambiguous input-driven automata* (UIDPDA) [20]. For more details on input-driven automata and their complexity, the readers are directed to a fairly recent survey [19].

The purpose of this paper is to investigate the closure and the state complexity of input-driven pushdown automata with respect to several further operations on languages, which are fairly standard in the theoretical research in formal language theory—yet their application to IDPDA has not yet been considered.

The first operation, investigated in Sect. 3, is **insertion**,  $\text{ins}(L, K) = \{xyz \mid xz \in L, y \in K\}$ . For finite automata, the closure under this operation is folklore, and its precise state complexity has recently been determined by Han et al. [8]. For input-driven automata, the closure is currently known only for singleton  $K$  [22]. This paper demonstrates the closure under the assumption that  $K$  consists only of well-nested strings, and determines the worst-case number of states in an NIDPDA recognizing  $\text{ins}(L, K)$  as  $mn + 2m$ , where  $m$  is the number of states in the NIDPDA for  $K$ , while the NIDPDA for  $L$  has  $n$  states. For unrestricted  $K$ , the non-closure is established.

Another related operation is **deletion**,  $\text{del}(L, K) = \{xz \mid \exists y \in K : xyz \in L\}$ . It is well-known that if  $L$  is regular, then the result is regular for an arbitrary  $K$ . The state complexity of this operation for finite automata has recently been studied by Han et al. [9]. For NIDPDA, as shown in Sect. 4,  $\text{del}(L, K)$  is representable with  $2n$  states, as long as  $K$  consists only of well-nested strings; without this assumption, there is a non-closure result.

The third operation, studied in Sect. 5, is the **square root**,  $\sqrt{L} = \{w \mid ww \in L\}$ . Assuming that  $L$  consists of only well-nested strings and is recognized by an  $n$ -state DIDPDA, its square root can be represented by a DIDPDA with  $n^n$  states, and this bound is tight; this proof uses the *behaviour functions* of DIDPDA [18,21] in the same way as in the similar result for deterministic finite automata (DFA) [15]. For NIDPDA, as well as for nondeterministic finite automata (NFA), it is shown that this operation requires at most  $n^3$  states and at least  $(n-1)(n-2)(n-3)$  states in the worst case. If  $L$  is not restricted to well-nested strings, a non-closure is established.

Section 6 considers the operation of taking the **first half** of all even-length strings in a language,  $\frac{1}{2}L = \{u \mid \exists v : |u| = |v|, uv \in L\}$ . This is a particular case of *proportional removals*, studied by Maslov [15] and by Domaratzki [4] for DFA, and by Goč et al. [6] for NFA. For NIDPDA, if  $L$  consists of well-nested strings,

then  $\frac{1}{2}L$  is recognized by an NIDPDA with  $2^{O(n^2)}$  states. For not necessarily well-nested  $L$ , there is again a non-closure.

The last Sect. 7 proposes an open problem on the state complexity of **scattered substrings** for IDPDA.

## 2 Input-Driven Automata

A *deterministic input-driven pushdown automaton* (DIDPDA) [1, 16] is a special case of a deterministic pushdown automaton, in which the input alphabet  $\Sigma$  is split into three disjoint sets of *left brackets*  $\Sigma_{+1}$ , *right brackets*  $\Sigma_{-1}$  and *neutral symbols*  $\Sigma_0$ . If the input symbol is a left bracket from  $\Sigma_{+1}$ , then the automaton always pushes one symbol onto the stack. For a right bracket from  $\Sigma_{-1}$ , the automaton must pop one symbol. Finally, for a neutral symbol in  $\Sigma_0$ , the automaton may not use the stack. In this paper, symbols from  $\Sigma_{+1}$  and  $\Sigma_{-1}$  shall be denoted by left and right angled brackets, respectively ( $<$ ,  $>$ ), whereas lower-case Latin letters from the beginning of the alphabet ( $a, b, c, \dots$ ) shall be used for symbols from  $\Sigma_0$ .

A *nondeterministic input-driven pushdown automaton* (NIDPDA) [1, 3] is a similarly restricted case of a nondeterministic pushdown automaton, in which the type of the action on the stack is determined by the input symbol, whereas the actual next state and the symbol pushed onto the stack may be selected nondeterministically.

Input-driven automata are often restricted to operate on input strings, in which the brackets are *well-nested*. When an input-driven automaton reads a left bracket  $< \in \Sigma_{+1}$ , it pushes a symbol onto the stack. This symbol is popped at the exact moment when the automaton encounters the matching right bracket  $> \in \Sigma_{-1}$ . Thus, a computation of an input-driven automaton on any well-nested substring leaves the stack contents untouched, as illustrated in Fig. 1.

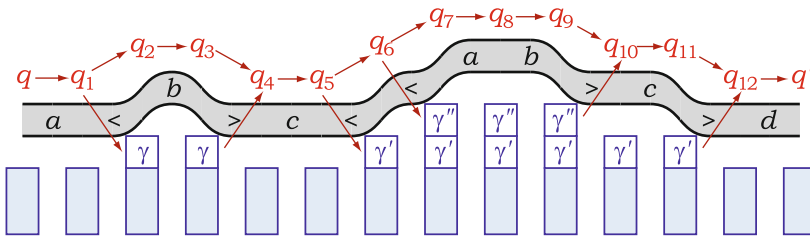


Fig. 1. The computation of an IDPDA on a well-nested string.

Alur and Madhusudan [1] also considered input-driven automata operating on potentially ill-nested input strings. For every unmatched left bracket, the symbol pushed to the stack when reading this bracket is never popped, and remains in the stack to the end of the computation. An unmatched right bracket



is read with an empty stack: instead of popping a stack symbol, the automaton merely detects that the stack is empty and makes a special transition, which leaves the stack empty.

**Definition 1 (von Braunmühl and Verbeek [3]; Alur and Madhusudan [1]).** A nondeterministic input-driven pushdown automaton (NIDPDA) over an alphabet  $\tilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$  consists of

- a finite set  $Q$  of states, with set of initial states  $Q_0 \subseteq Q$  and accepting states  $F \subseteq Q$ ;
- a finite stack alphabet  $\Gamma$ , and a special symbol  $\perp \notin \Gamma$  for the empty stack;
- for a neutral symbol  $c \in \Sigma_0$ , a transition function  $\delta_c: Q \rightarrow 2^Q$  gives the set of possible next states;
- for each left bracket symbol  $< \in \Sigma_{+1}$ , the behaviour of the automaton is described by a function  $\delta_{<}: Q \rightarrow 2^{Q \times \Gamma}$ , which, for a given current state, provides a set of pairs  $(q, \gamma)$ , with  $q \in Q$  and  $\gamma \in \Gamma$ , where each pair means that the automaton enters the state  $q$  and pushes  $\gamma$  onto the stack;
- for every right bracket symbol  $> \in \Sigma_{-1}$ , there is a function  $\delta_{>}: Q \times (\Gamma \cup \{\perp\}) \rightarrow 2^Q$  specifying possible next states, assuming that the given stack symbol is popped from the stack (or that the stack is empty).

A configuration is a triple  $(q, w, x)$ , with the current state  $q \in Q$ , remaining input  $w \in \Sigma^*$  and stack contents  $x \in \Gamma^*$ . Possible next configurations are defined as follows.

$$\begin{aligned} (q, cw, x) \vdash_A (q', w, x), & \quad c \in \Sigma_0, q \in Q, q' \in \delta_c(q) \\ (q, <w, x) \vdash_A (q', w, \gamma x), & \quad < \in \Sigma_{+1}, q \in Q, (q', \gamma) \in \delta_{<}(q) \\ (q, >w, sx) \vdash_A (q', w, x), & \quad > \in \Sigma_{-1}, q \in Q, s \in \Gamma, q' \in \delta_{>}(q, \gamma) \\ (q, >w, \varepsilon) \vdash_A (q', w, \varepsilon), & \quad > \in \Sigma_{-1}, q' \in \delta_{>}(q, \perp) \end{aligned}$$

The language recognized by  $A$  is the set of all strings  $w \in \Sigma^*$ , on which the automaton, having begun its computation in the configuration  $(q_0, w, \varepsilon)$ , eventually reaches a configuration of the form  $(q, \varepsilon, x)$ , with  $q \in F$  and with any stack contents  $x \in \Gamma^*$  (for well-nested inputs, the stack is empty at this point).

An NIDPDA is deterministic (DIDPDA), if there is a unique initial state and every transition provides exactly one action.

Von Braunmühl and Verbeek [3], proved that every  $n$ -state NIDPDA operating on well-nested strings can be transformed to a  $2^{n^2}$ -state DIDPDA. Alur and Madhusudan [1] extended this construction to allow ill-nested inputs, so that a DIDPDA has  $2^{2n^2}$  states; in the worst case,  $2^{\Omega(n^2)}$  states are necessary.

The known closure results under Boolean operations, concatenation, star, reversal, quotient and edit distance neighbourhood are all valid for input-driven automata operating on ill-nested strings.

### 3 Insertion

The insertion operation is a binary operation on languages:  $\text{ins}(L, K)$  is the set of all strings obtained by taking any string from  $L$  and any position in this string, and inserting any string from  $K$  at that position.

$$\text{ins}(L, K) = \{xyz \mid xz \in L, y \in K\}$$

The question is: when both  $K$  and  $L$  are recognized by IDPDA, shall  $\text{ins}(L, K)$  always be recognized by an IDPDA? Provided that  $K$  does not contain any ill-nested strings, the answer is positive, established by the following effective construction.

**Lemma 1.** *Let  $L$  be a language recognized by an NIDPDA  $\mathcal{B}$  with the set of states  $Q$  and with a stack alphabet  $\Gamma$ . Let  $K$  be a set of well-nested strings, which is recognized by an NIDPDA  $\mathcal{A}$  with the set of states  $P$  and with a stack alphabet  $\Theta$ . Then, the language  $\text{ins}(L, K)$  is recognized by an NIDPDA with the set of states  $Q \cup \tilde{Q} \cup (P \times Q)$ , where  $\tilde{Q} = \{\tilde{q} \mid q \in Q\}$ , and with the stack alphabet  $\Gamma \cup \Theta$ .*

*Proof.* The new automaton begins by simulating the computation of  $\mathcal{B}$  in the states  $\tilde{Q}$ .

At any moment, while in a state  $\tilde{q} \in \tilde{Q}$ , it may nondeterministically guess that a substring accepted by  $\mathcal{A}$  begins at the next symbol. If the next symbol is  $c \in \Sigma_0$ , the new automaton may make a transition from  $\tilde{q}$  by  $c$  to a pair  $(p, q)$ , where  $p$  is any state, to which  $\mathcal{A}$  may go from its initial state  $p_0$  by reading  $c$ . If the next symbol is a left bracket  $< \in \Sigma_{+1}$ , the new automaton may choose to go from  $\tilde{q}$  by the left bracket  $<$  to a pair  $(p, q)$ , pushing  $\theta$ , where  $\mathcal{A}$  may go from  $p_0$  to  $p$  by this bracket  $<$ , pushing  $\theta$ .

Thus, the simulation of  $\mathcal{A}$  on a substring begins. It proceeds in the states from  $P \times Q$ , with the simulation carried out in the first component, whereas the state of  $\mathcal{B}$  remembered in the second component remains unchanged. Whenever the simulating automaton is in a state  $(p, q)$  with  $p$  accepting, it may decide that the well-nested substring being inserted has now ended, and accordingly make simulate  $\mathcal{B}$ 's transition from the state  $q$  on the next input symbol. After that, the simulation continues as  $\mathcal{B}$  in the states from  $Q$ .  $\square$

This construction is optimal in the worst case, because of the recent result by Han et al. [8] for NFA. Han et al. [8] constructed a pair of witness languages,  $K$  and  $L$ , recognized by an  $m$ -state NFA and by an  $n$ -state, respectively, and proved that every NFA recognizing the language  $\text{ins}(L, K)$  must have at least  $mn + 2m$  states. Assuming that the partition of the alphabet is fixed, with all symbols in  $\Sigma_0$ , and with no brackets, an NIDPDA cannot do anything more than an NFA, and for that reason, the lower bound by Han et al. [8] also applies to NIDPDA. This yields the following result.

**Theorem 1.** *The state complexity of inserting a language recognized by an  $m$ -state NIDPDA into a language of well-nested strings recognized by an  $n$ -state NIDPDA is exactly  $mn + 2m$ .*

On the other hand, if the strings being inserted are not required to be well-nested, then the closure property no longer holds, as shown in the next example.

*Example 1.* The following language is recognized by an input-driven automaton.

$$L = \{ \langle^n \rangle^n \mid n \geq 1 \}$$

However, no input-driven automaton recognizes the language  $\text{ins}(L, \langle \rangle^* \langle^* \rangle)$ .

*Proof.* If  $\text{ins}(L, \langle \rangle^* \langle^* \rangle)$  is recognized by some input-driven automaton, then so is its intersection with the regular language  $\langle^+ \rangle^+ \langle^+ \rangle^+$ . The intersection ensures that the insertion is made exactly in the middle of a string  $\langle^n \rangle^n$ .

$$\text{ins}(L, \langle \rangle^* \langle^* \rangle) \cap \langle^+ \rangle^+ \langle^+ \rangle^+ = \{ \langle^{n+1} \rangle^i \langle^j \rangle^{n+1} \mid n, i, j \geq 1 \}$$

Let the latter intersection be recognized by a DIDPDA with a set of states  $Q$ . Let  $n = |Q|^2 + 1$ , and consider the accepting computation on the string  $\langle^n \rangle^n \langle^n \rangle^n$ . In the first half of this computation, that is, on the prefix  $\langle^n \rangle^n$ , for every  $i \in \{1, \dots, n\}$ , let  $p_i$  and  $q_i$  be the states reached after reading  $\langle^i$  and  $\langle^n \rangle^{n-i}$ , respectively. When the automaton reads the  $(i + 1)$ -th left bracket in the state  $p_i$ , it pushes the stack symbol  $s_i = \gamma_{\langle}(p_i)$ , which is popped when reading the  $(n - i)$ -th right bracket in the state  $q_{j+1}$ , so that  $q_j = \delta_{\rangle}(q_{j+1}, s_i)$ . There are  $|Q|^2$  different pairs  $(p_i, q_i)$ , and therefore, for some  $i$  and  $j$ , these pairs coincide:  $(p_i, q_i) = (p_j, q_j)$ , with  $i < j$ . Then the two segments of computation between  $p_{i+1}$  and  $p_j$  and between  $q_j$  and  $q_{i+1}$ , along with the stack symbols linking them to each other, can be cut, obtaining an accepting computation on the string  $\langle^{n+j-i} \rangle^{n+j-i} \langle^{n-i} \rangle^{n-i}$ , which does not belong to the language.  $\square$

## 4 Deletion

The deletion operation is a binary operation on languages:  $\text{del}(L, K)$  is the set of all strings obtained by taking any string from  $L$  and removing any substring belonging to  $K$  from that string.

$$\text{del}(L, K) = \{ xz \mid \exists y \in K : xyz \in L \}$$

If the set of strings being deleted consists of well-nested strings, there is the following closure result.

**Lemma 2.** *Let a language  $L$  be recognized by an NIDPDA  $\mathcal{B}$  with states  $Q$  and stack alphabet  $\Gamma$ . Let  $K$  be a language containing only well-nested strings, recognized by an NIDPDA  $\mathcal{A}$ . Then, the language  $\text{del}(L, K)$  is recognized by an NIDPDA with the set of states  $Q \cup \tilde{Q}$ , where  $\tilde{Q} = \{ \tilde{q} \mid q \in Q \}$ , and with the stack alphabet  $\Gamma$ .*

*Proof.* The new automaton first simulates  $\mathcal{B}$  in the states from  $\tilde{Q}$ . At some point, while in some state  $\tilde{q}$ , it guesses that a string accepted by  $\mathcal{A}$  has been deleted beginning from the next position. Let  $q' \in Q$  be any such state of  $\mathcal{B}$ ,

that there exists a well-nested string  $y$ , which is accepted by  $\mathcal{A}$ , whereas  $\mathcal{B}$  may read  $y$  beginning from the state  $q$ , and finish reading it in the state  $q'$ . Then the simulating automaton may guess such a state  $q'$  and continue its simulation, as if it is currently in the state  $q'$ .

The construction is effective, because all pairs  $(q, q')$  satisfying the above conditions can be determined by first constructing an IDPDA  $\mathcal{D}_{q,q'}$  that recognizes the set of all strings  $y$  as defined above, and then applying the known emptiness test [2] to this IDPDA.  $\square$

If ill-nested strings may be deleted, then, as in the case of insertion, the family is no longer closed under this operation.

*Example 2.* The following language is recognized by an input-driven automaton.

$$L = \{ \langle^m \ll^n \gg^n \rangle^m \mid m, n \geq 1 \}$$

However, no input-driven automaton recognizes the language  $\text{del}(L, \ll^*)$ .

*Proof.* If  $\text{del}(L, \ll^*)$  is recognized by some input-driven automaton, then so is its intersection with the regular language  $\langle^+ \gg^+ \rangle^+$ , which ensures that *all* double left brackets ( $\ll$ ) are erased.

$$\text{del}(L, \ll^*) \cap \langle^+ \gg^+ \rangle^+ = \{ \langle^m \gg^i \rangle^m \mid m, i \geq 1 \}$$

If the latter intersection is recognized by a DIDPDA with a set of states  $Q$ , then let  $m = |Q|$ . Then, in the computation on  $\langle^m \gg^m \rangle^m$ , the stack must be empty after reading the prefix  $\langle^m \gg^m$ . In the last part of the computation, while reading the suffix  $\rangle^m$ , the automaton passes through a sequence of states  $p_0, \dots, p_m$ , behaving like a DFA, with every next state determined by a transition by the empty stack as  $p_{i+1} = \delta_{\rangle}(p_i, \perp)$ . Since  $m + 1 > |Q|$ , two of these states must coincide:  $p_i = p_j$ , with  $0 \leq i < j \leq m$ . Then this segment of the computation can be cut out without affecting the acceptance, and the automaton accepts the string  $\langle^m \gg^m \rangle^{m-(j-i)}$ , which is not in the language.  $\square$

## 5 Square Root

For a string of the form  $w w$ , the string  $w$  is its *square root*, denoted by  $\sqrt{w w} = w$ . The square root of a language is defined as the set of square roots of all its applicable elements.

$$\sqrt{L} = \{ w \mid w w \in L \}$$

The regular languages are closed under this operation. Indeed, for every  $n$ -state DFA recognizing  $L$ , one can construct a DFA with  $n^n$  states that computes the *behaviour function* of the original DFA on the string  $w$ : this is a function  $f_w : Q \rightarrow Q$  that maps each state  $q \in Q$  to the state reached by the DFA after reading  $w$  beginning from the state  $q$ . Denote by  $Q^Q$  the set of all such functions. Then, the set of states of the constructed DFA is  $Q^Q$ .

This construction generalizes to input-driven automata, under the assumption that  $L$  contains only well-nested strings. Notably, in such a case, the language  $\sqrt{L}$  also contains only well-nested strings.

**Lemma 3.** *If a language  $L$  contains only well-nested strings, and is recognized by a DIDPDA with the set of states  $Q$ , then the language  $\sqrt{L}$  is recognized by a DIDPDA with the set of states  $Q^Q$  and with the stack alphabet  $\Sigma_{+1} \times Q^Q$ .*

The new DIDPDA processing  $w$  constructs the *behaviour function* [18, 21] of the original DIDPDA on  $w$ . This is one of the basic constructions for DIDPDA, based on the following observation: when a DIDPDA with a set of states  $Q$  processes a well-nested string  $w$  and begins in a state  $q$ , it finishes reading that string in some state  $f_w(q)$ , where  $f_w: Q \rightarrow Q$  is its *behaviour function on  $w$* , and the stack is left untouched. Thus,  $f$  completely characterizes the behaviour of a DIDPDA on  $w$ . For any given DIDPDA  $\mathcal{A}$ , it is possible to construct an  $n^n$ -state DIDPDA, where  $n = |Q|$ , that reaches the end of an input  $w$  in a state representing the behaviour of  $\mathcal{A}$  on the longest well-nested suffix of  $w$ . This construction is necessary for optimal constructions representing operations on DIDPDA [21].

*Proof (of Lemma 3).* This is a known DIDPDA construction for the behaviour function  $f_w: Q \rightarrow Q$  of  $\mathcal{A}$  on the input string  $w$ . Then the function  $f_w \circ f_w$  is its behaviour of  $\mathcal{A}$  on  $ww$ , and the simulating automaton accepts in a state  $f \in Q^Q$  whenever  $(f \circ f)(q_0) \in F$ .

A matching lower bound for the state complexity of the square root on DFA is known. It immediately applies to DIDPDA, showing that  $n^n$  is the exact complexity of the square root for this model as well.

**Lemma 4 (Maslov [15]).** *For every  $n \geq 1$ , there exists such an  $n$ -state DFA over a 3-symbol alphabet (equivalently, an  $n$ -state DIDPDA over the alphabet  $\Sigma_{+1} = \Sigma_{-1} = \emptyset$ ,  $\Sigma_0 = \{a, b, c\}$ ) that every DFA (as well as a DIDPDA) recognizing the language  $\sqrt{L}$  requires at least  $n^n$  states.*

For NIDPDA, the state complexity of the square root is quite different. As to the authors' knowledge, the number of states in an NFA recognizing the square root of an  $n$ -state NFA is not yet known, and it is natural to establish it first.

**Theorem 2.** *Square root of an  $n$ -state NFA is recognized by an  $n^3$ -state NFA.*

*For every  $n$ , there is a language  $L_n$  over a three-symbol alphabet, which is recognized by an  $n$ -state DFA, whereas every NFA recognizing  $\sqrt{L_n}$  must have at least  $(n - 1)(n - 2)(n - 3)$  states.*

*Proof.* Let  $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$  be any  $n$ -state NFA. The NFA  $\mathcal{B}$  recognizing  $\sqrt{L(\mathcal{A})}$  uses the set of triples  $Q \times Q \times Q$  as its states. In the beginning of its computation on a string  $w$ , it guesses the state  $p$  reached by  $\mathcal{A}$  after reading  $w$ , remembers this state as the last component on the triple, and begins simulating two computations of  $\mathcal{A}$ , one beginning from an initial state and the other beginning from  $p$ . Accordingly, its set of initial states is  $\{(q_0, p, p) \mid q_0 \in Q_0, p \in Q\}$ , and it uses the following transitions.

$$\delta'((q, r, p), a) = \{(q', r', p) \mid q' \in \delta(q, a), r' \in \delta(r, a)\}$$

When  $\mathcal{B}$  finishes reading  $w$  in a state  $(q, r, p)$ , it has verified that  $\mathcal{A}$ , upon reading  $w$ , can move from  $q_0$  to  $q$ , as well as from  $p$  to  $r$ . If  $q = p$ , this confirms that  $\mathcal{A}$  can move from  $q_0$  to  $r$  upon reading  $w$ , and if  $r$  is accepting, then  $\mathcal{B}$  should accept its input string  $w$ . Accordingly, the set of accepting states of  $\mathcal{B}$  is  $\{(p, r, p) \mid p \in Q, r \in F\}$ .

Turning to the lower bound, the language  $L_n$  is the standard “universal” witness language, as in Lemma 4. It is recognized by a DFA with the states  $\{0, \dots, n - 1\}$ , and has the following property: for every function  $f: \{0, \dots, n - 1\} \rightarrow \{0, \dots, n - 1\}$ , there is such a string  $x_f$ , that, upon reading  $x_f$  beginning from a state  $i$ , the DFA finishes reading it in the state  $f(i)$ . The state 0 is initial and  $n - 2$  is accepting.

Now the lower bound is proved using the standard *fooling set* method. Let  $(i, j, k)$  be a triple of states, with  $i, k \in \{0, \dots, n - 2\}$ ,  $j \in \{1, \dots, n - 3\}$  and  $i \neq k$ . Define the string  $u_{i,j,k} = x_f$ , where  $f$  is a function defined by  $f(0) = i$ ,  $f(j) = k$  and  $f(t) = n - 1$  for all remaining arguments. The other string is  $v_{i,j,k} = x_g$ , where  $g(i) = j$ ,  $g(k) = n - 2$  and  $g(t) = n - 1$  for the rest of the arguments. Then, the concatenation  $u_{i,j,k}v_{i,j,k}$  maps the initial state to  $j$ , and  $j$  to the accepting state. This means that the square  $(u_{i,j,k}v_{i,j,k})^2$  maps the initial state to the accepting state, and therefore  $u_{i,j,k}v_{i,j,k}$  belongs to  $\sqrt{L(\mathcal{A})}$ . On the other hand, for any two different pairs on this list,  $(u_{i,j,k}, v_{i,j,k})$  and  $(u_{i',j',k'}, v_{i',j',k'})$ , at least one of the mismatched concatenations  $u_{i,j,k}v_{i',j',k'}$  and  $u_{i',j',k'}v_{i,j,k}$  is not in  $\sqrt{L(\mathcal{A})}$ . Then, every NFA recognizing the square root must have at least as many states as there are pairs in this fooling set, and there are  $(n - 1)(n - 2)(n - 3)$  such pairs.  $\square$

The construction for NFA easily generalizes to NIDPDA, whereas the lower bound applies to NIDPDA as it is.

**Theorem 3.** *Assume that a language  $L$  contains only well-nested strings, and let it be recognized by an NIDPDA with set of states  $Q$ , and with stack alphabet  $\Gamma$ . Then the language  $\sqrt{L}$  is recognized by an NIDPDA with the set of states  $Q^3$  and with the stack alphabet  $\Gamma^2$ .*

*For every  $n$ , there is a language  $L_n$  over an alphabet that consists of four neutral symbols, which is recognized by an  $n$ -state DFA, whereas every NIDPDA recognizing  $\sqrt{L_n}$  must have at least  $(n - 1)(n - 2)(n - 3)$  states.*

Thus, the state complexity of the square root has been established as  $n^3 - O(n^2)$  both for NFA and for NIDPDA. For the latter, the construction relies on all strings’ being well-nested.

In the general case without the well-nestedness assumption, input-driven automata again demonstrate a non-closure.

*Example 3.* The following language is recognized by an input-driven automaton.

$$L = \{ >^i <^n >^n <^j \mid i, n, j \geq 1 \} \cup \{ <^m >^m <^n >^n \mid m, n \geq 1 \}$$

Its square root has the following form.

$$\sqrt{L} = \{ >^n <^n \mid n \geq 1 \} \cup \{ <^n >^n \mid n \geq 1 \}$$

It is not recognized by any input-driven automaton, for any partition of the alphabet.

## 6 Proportional Removals

For a string of even length,  $w = a_1 \dots a_{2\ell}$ , its *first half*, denoted by  $\frac{1}{2}w$ , is the string  $a_1 \dots a_\ell$ , obtained by discarding the second half of  $w$ . This operation is extended to languages element-wise, as follows.

$$\frac{1}{2}(L) = \{ \frac{1}{2}w \mid w \in L, |w| \text{ is even} \}$$

As reported by Seiferas and McNaughton [26], Yamada, Stearns and Hartmanis were the first to prove that the regular languages are closed under this operation. Maslov [15] determined its state complexity for DFA as  $2^{\Theta(\sqrt{n \log n})}$ , relying on the known determinization of unary NFAs using  $e^{(1+o(1))\sqrt{n \ln n}}$  states. Later Domaratzki [4] has further investigated its state complexity, and Goč et al. [6] proved that for NFA, the state complexity of “one half” is  $\Theta(n^2)$ .

For input-driven automata, there is a construction somewhat similar to those used for finite automata. Unfortunately, it cannot anymore rely on determinizing unary NFA, and for that reason is much less efficient with respect to the number of states.

**Lemma 5.** *For an  $n$ -state NIDPDA recognizing a language  $L$ , the well-nested subset of the language  $\frac{1}{2}(L)$  is recognized by an NIDPDA with  $2^{O(n^2)}$  states.*

This time no assumptions are made on the well-nestedness of strings in  $L$ , but the construction produces an automaton that defines the intersection  $\frac{1}{2}L$  with the set of well-nested strings.

*Proof.* The states of the new automaton are of the form  $(q, \hat{q}, p)$ , where  $q$  is the state of the original automaton’s ongoing simulation,  $\hat{q}$  is the guessed state of the original automaton in the end of the simulation, whereas  $p$  is a state of a certain finite automaton.

For each state  $q_i$  of the given input-driven automaton, there exists an NFA  $\mathcal{A}_i$  that accepts a string  $a^\ell$  if and only if there exists any string of length  $\ell$  accepted by the input-driven automaton, beginning in the state  $q_i$  with the empty stack. Such a finite automaton exists by Parikh’s theorem, and is effectively obtained as follows. First, the NIDPDA is transformed to a grammar of size  $O(n^2)$ . Then, the efficient construction for Parikh’s theorem by Esparza et al. [5] is applied to this grammar, producing an NFA of size  $2^{O(n^2)}$ .

Given the well-nested first half  $u$  of some string, the new automaton begins its computation in any state of the form  $(q_0, q_i, p_0^{(i)})$ , where  $q_0$  is the original automaton’s initial state,  $q_i$  is any of its states, and  $p_0^{(i)}$  is the initial state of the finite automaton  $\mathcal{A}_i$ . Then, the new automaton simulates the behaviour of the original automaton on  $u$ , along with running  $\mathcal{A}_i$  on the same string. In the end, the new automaton accepts if its first component reaches the state  $q_i$  guessed in the beginning, while the simulated  $\mathcal{A}_i$  accepts the string.  $\square$

The construction in Lemma 5 should be taken as a rough upper bound on the state complexity of one half. Improving this construction is proposed as an open problem.

If the exact value of  $\frac{1}{2}L$  is required, that is, without the intersection with the set of well-nested strings, then there is yet another non-closure result.

*Example 4.* The following language consists only of well-nested strings and is recognized by an input-driven automaton.

$$L = \{ \langle^m \rangle^m \langle^n \rangle^n (cc)^n cc \mid m, n \geq 1 \}$$

*Proof.* Each string is of length  $2m+4n+2$ , and its first half is of length  $m+2n+1$ . This first half belongs to the set  $\langle^* \rangle^* \langle^* \rangle^* c$  if and only if  $m = n$ . For this reason, the set of first halves of strings in  $L$ , under intersection with the regular language  $\langle^* \rangle^* \langle^* \rangle^* c$ , has the following form.

$$\frac{1}{2}L \cap \langle^* \rangle^* \langle^* \rangle^* c = \{ \langle^n \rangle^n \langle^n \rangle^n c \mid n \geq 1 \}$$

The latter language is certainly not recognized by any input-driven automaton, and therefore neither is  $\frac{1}{2}L$ .  $\square$

## 7 Scattered Substrings

For each string  $w = a_1 \dots a_n$ , any string  $a_{i_1} \dots a_{i_\ell}$ , with  $1 \leq i_1 < \dots < i_\ell \leq n$ , is its *scattered substring*. Denote by  $sub(w)$  the set of all scattered substrings of  $w$ , and let  $sub(L) = \bigcup_{w \in L} sub(w)$  for a language  $L$ . By the Higman–Haines theorem [7, 11], the language  $sub(L)$  is regular for an arbitrary language  $L$ , and can therefore be recognized by an IDPDA without paying attention to the bracket structure, by pushing and popping dummy stack symbols on the brackets.

For regular languages, the state complexity of scattered substrings is  $2^{\Theta(n)}$  [12, 17], with recent further results by Karandikar et al. [13].

What is the state complexity of this operation for IDPDA? As proved by van Leeuwen [14], for  $L$  given by a grammar, the language  $sub(L)$  is *effectively* regular. However, van Leeuwen's [14] constructive proof does not include any estimation of the size of the regular language; it uses the finiteness of the basis to prove that the construction terminates. For that reason, no upper bound on the state complexity of scattered substrings for IDPDA is known.

## References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: ACM Symposium on Theory of Computing (STOC 2004), Chicago, USA, 13–16 June 2004, pp. 202–211 (2004). <https://doi.org/10.1145/1007352.1007390>
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM **56**(3) (2009). <https://doi.org/10.1145/1516512.1516518>



3. von Braunmühl, B., Verbeek, R.: Input driven languages are recognized in  $\log n$  space. *Ann. Discret. Math.* **24**, 1–20 (1985). [https://doi.org/10.1016/S0304-0208\(08\)73072-X](https://doi.org/10.1016/S0304-0208(08)73072-X)
4. Domaratzki, M.: State complexity of proportional removals. *J. Autom. Lang. Comb.* **7**(4), 455–468 (2002)
5. Esparza, J., Ganty, P., Kiefer, S., Luttenberger, M.: Parikh’s theorem: a simple and direct automaton construction. *Inf. Process. Lett.* **111**(12), 614–619 (2011). <https://doi.org/10.1016/j.ipl.2011.03.019>
6. Goč, D., Palioudakis, A., Salomaa, K.: Nondeterministic state complexity of proportional removals. *Int. J. Found. Comput. Sci.* **25**(7), 823–836 (2014). <https://doi.org/10.1142/S0129054114400103>
7. Haines, L.H.: On free monoids partially ordered by embedding. *J. Comb. Theory* **6**, 94–98 (1969). [https://doi.org/10.1016/S0021-9800\(69\)80111-0](https://doi.org/10.1016/S0021-9800(69)80111-0)
8. Han, Y.-S., Ko, S.-K., Ng, T., Salomaa, K.: State complexity of insertion. *Int. J. Found. Comput. Sci.* **27**(7), 863–878 (2016). <https://doi.org/10.1142/S0129054116500349>
9. Han, Y.-S., Ko, S.-K., Salomaa, K.: State complexity of deletion and bipolar deletion. *Acta Informatica* **53**(1), 67–85 (2016). <https://doi.org/10.1007/s00236-015-0245-y>
10. Han, Y.-S., Salomaa, K.: Nondeterministic state complexity of nested word automata. *Theor. Comput. Sci.* **410**, 2961–2971 (2009). <https://doi.org/10.1016/j.tcs.2009.01.004>
11. Higman, G.: Ordering by divisibility in abstract algebras. *Proc. Lond. Math. Soc.* **s3-2**(1), 326–336 (1952)
12. Gruber, H., Holzer, M., Kutrib, M.: More on the size of Higman-Haines sets: effective constructions. *Fundamenta Informaticae* **91**(1), 105–121 (2009). <https://doi.org/10.3233/FI-2009-0035>
13. Karandikar, P., Niewerth, M., Schnoebelen, Ph.: On the state complexity of closures and interiors of regular languages with subwords and superwords. *Theor. Comput. Sci.* **610**(A), 91–107 (2016). <https://doi.org/10.1016/j.tcs.2015.09.028>
14. van Leeuwen, J.: Effective construction in well-partially-ordered free monoids. *Discrete Math.* **21**(3), 237–252 (1978). [https://doi.org/10.1016/0012-365X\(78\)90156-5](https://doi.org/10.1016/0012-365X(78)90156-5)
15. Maslov, A.N.: Estimates of the number of states of finite automata. *Soviet Math. Dokl.* **11**, 1373–1375 (1970)
16. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J., van Leeuwen, J. (eds.) *ICALP 1980*. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980). [https://doi.org/10.1007/3-540-10003-2\\_89](https://doi.org/10.1007/3-540-10003-2_89)
17. Okhotin, A.: On the state complexity of scattered substrings and superstrings. *Fundamenta Informaticae* **99**(3), 325–338 (2010). <https://doi.org/10.3233/FI-2010-252>
18. Okhotin, A.: Input-driven languages are linear conjunctive. *Theor. Comput. Sci.* **618**, 52–71 (2016). <https://doi.org/10.1016/j.tcs.2016.01.007>
19. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. *SIGACT News* **45**(2), 47–67 (2014). <https://doi.org/10.1145/2636805.2636821>
20. Okhotin, A., Salomaa, K.: Descriptive complexity of unambiguous input-driven pushdown automata. *Theor. Comput. Sci.* **566**, 1–11 (2015). <https://doi.org/10.1016/j.tcs.2014.11.015>
21. Okhotin, A., Salomaa, K.: State complexity of operations on input-driven pushdown automata. *J. Comput. Syst. Sci.* **86**, 207–228 (2017). <https://doi.org/10.1016/j.jcss.2017.02.001>

22. Okhotin, A., Salomaa, K.: Edit distance neighbourhoods of input-driven pushdown automata. In: Weil, P. (ed.) CSR 2017. LNCS, vol. 10304, pp. 260–272. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-58747-9\\_23](https://doi.org/10.1007/978-3-319-58747-9_23)
23. Okhotin, A., Salomaa, K.: The quotient operation on input-driven pushdown automata. In: Pighizzini, G., Câmpeanu, C. (eds.) DCFS 2017. LNCS, vol. 10316, pp. 299–310. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-60252-3\\_24](https://doi.org/10.1007/978-3-319-60252-3_24)
24. Piao, X., Salomaa, K.: Operational state complexity of nested word automata. *Theor. Comput. Sci.* **410**, 3290–3302 (2009). <https://doi.org/10.1016/j.tcs.2009.05.002>
25. Salomaa, K.: Limitations of lower bound methods for deterministic nested word automata. *Inf. Comput.* **209**, 580–589 (2011). <https://doi.org/10.1016/j.ic.2010.11.021>
26. Seiferas, J.I., McNaughton, R.: Regularity-preserving relations. *Theor. Comput. Sci.* **2**(2), 147–154 (1976). [https://doi.org/10.1016/0304-3975\(76\)90030-X](https://doi.org/10.1016/0304-3975(76)90030-X)



# State Complexity Characterizations of Parameterized Degree-Bounded Graph Connectivity, Sub-Linear Space Computation, and the Linear Space Hypothesis

Tomoyuki Yamakami<sup>(✉)</sup>

Faculty of Engineering, University of Fukui,  
3-9-1 Bunkyo, Fukui 910-8507, Japan  
TomoyukiYamakami@gmail.com

**Abstract.** The linear space hypothesis is a practical working hypothesis, which originally states the insolvability of a restricted 2CNF Boolean formula satisfiability problem parameterized by the number of Boolean variables. From this hypothesis, it follows that the degree-3 directed graph connectivity problem (3DSTCON) parameterized by the number of vertices in a given graph cannot belong to PsubLIN, composed of decision problems computable by polynomial-time, sub-linear-space deterministic Turing machines. This hypothesis immediately implies  $L \neq NL$  and it was used as a solid foundation to obtain new lower bounds on the computational complexity of various NL search and NL optimization problems. The state complexity of transformation refers to the cost of converting one type of finite automata to another type, where the cost is measured in terms of the increase of the number of inner states of the converted automata from that of the original automata. We relate the linear space hypothesis to the state complexity of transforming restricted 2-way nondeterministic finite automata to computationally equivalent 2-way alternating finite automata having narrow computation graphs. For this purpose, we present state complexity characterizations of 3DSTCON and PsubLIN. We further characterize a non-uniform version of the linear space hypothesis in terms of the state complexity of transformation.

## 1 Backgrounds and an Overview

### 1.1 Parameterized Problems and the Linear Space Hypothesis

The *nondeterministic logarithmic-space* complexity class NL has been discussed since early days of computational complexity theory. Typical NL decision problems include the 2CNF Boolean formula satisfiability problem (2SAT) as well

as the directed  $s$ - $t$  connectivity problem<sup>1</sup> (DSTCON) of determining whether there exists a path from a given vertex  $s$  to another vertex  $t$  in a given directed graph  $G$ . These problems are also known to be NL-complete under log-space many-one reductions. The NL-completeness is so robust that even if we restrict our interest within graphs whose vertices are limited to be of degree at most 3, the corresponding decision problem, 3DSTCON, remains NL-complete.

In practice, when we measure the computational complexity of given problems, we tend to be more concerned with *parameterizations* of the problems. In other words, we treat the size of specific “input objects” given to the problem as a “practical” *size parameter*  $n$  and use it to measure how much resources are needed for algorithms to solve those problems. To emphasize the choice of such a size parameter  $m : \Sigma^* \rightarrow \mathbb{N}$  for a decision problem  $L$  over an alphabet  $\Sigma$ , it is convenient to write  $(L, m)$ , which gives rise to a *parameterized decision problem*. Since we deal only with such parameterized problems in the rest of this paper, we often drop the adjective “parameterized” as long as it is clear from the context.

Instances  $x = \langle G, s, t \rangle$  to 3DSTCON are usually parameterized by the number of vertices and that of edges in the graph  $G$ . It was shown in [2] that DSTCON with  $n$  vertices and  $m$  edges can be solved in  $O(m+n)$  steps using only  $n^{1-c/\sqrt{\log n}}$  space for a suitable constant  $c > 0$ . However, it is unknown whether we can reduce this space usage down to  $n^\varepsilon \text{polylog}(m+n)$  for a certain fixed constant  $\varepsilon \in [0, 1)$ . Such a bound is informally called “sub-linear” in a strong sense. It has been conjectured that, for every constant  $\varepsilon \in [0, 1)$ , no polynomial-time  $O(n^\varepsilon)$ -space algorithm solves DSTCON with  $n$  vertices (see references in, e.g., [1, 4]). For convenience, we denote by PsubLIN the collection of all parameterized decision problems  $(L, m)$  solvable deterministically in time polynomial in  $|x|$  using space at most  $m(x)^\varepsilon \ell(|x|)$  for certain constants  $\varepsilon \in [0, 1)$  and certain polylogarithmic (or polylog, in short) functions  $\ell$  [11].

The *linear space hypothesis* (LSH), proposed in [11], is a practical working hypothesis, which originally asserts the insolvability of a restricted form of 2SAT, denoted 2SAT<sub>3</sub>, together with the size parameter  $m_{vbl}(\phi)$  indicating the number of variables in each given Boolean formula  $\phi$ , in polynomial time using sub-linear space. As noted in [11], it is unlikely that 2SAT replaces 2SAT<sub>3</sub>. From this hypothesis, nevertheless, we immediately obtain the separation  $L \neq NL$ , which many researchers believe to hold. It was also shown in [11] that  $(2\text{SAT}_3, m_{vbl})$  can be replaced by  $(3\text{DSTCON}, m_{ver})$ , where  $m_{ver}(\langle G, s, t \rangle)$  refers to the number of vertices in  $G$ . LSH has acted as a reasonable foundation to obtain new lower bounds of several NL-search and NL-optimization problems [11, 12]. To find more applications of this hypothesis, we need to translate the hypothesis into other fields. In this paper, we look for a logically equivalent statement in automata theory, in hope that we would find more applications of LSH in this theory.

---

<sup>1</sup> This problem is also known as the graph accessibility problem and the graph reachability problem.

## 1.2 Finite Automata and State Complexity Classes

The purpose of this work is to look for an automata-theoretical statement that is logically equivalent to the linear space hypothesis; in particular, we seek a new characterization of the relationship between 3DSTCON and PsubLIN in terms of the state complexity of transforming a certain type of finite automata to another type with no direct reference to 3DSTCON or PsubLIN.

It is often cited from [3] (re-proven in [8]) that, if  $L = NL$ , then every  $n$ -state two-way nondeterministic finite automaton (or 2nfa) can be converted into an  $n^{O(1)}$ -state two-way deterministic finite automaton (or 2dfa) that agrees with it on all inputs of length at most  $n^{O(1)}$ . Conventionally, we call by *unary finite automata* automata working only on unary inputs (i.e., inputs over a one-letter alphabet). Geffert and Pighizzini [6] strengthened the aforementioned result by proving that the assumption of  $L = NL$  leads to the following: for any  $n$ -state unary 2nfa, there is a unary 2dfa of at most  $n^{O(1)}$ -states agreeing with it on all strings of length at most  $n$ . Within a few years, Kapoutsis [8] gave a similar characterization using  $L/poly$ , a non-uniform version of  $L$ :  $NL \subseteq L/poly$  if and only if (iff) there is a polynomial  $p$  such that any  $n$ -state 2nfa has a 2dfa of at most  $p(n)$  states agreeing with the 2nfa on strings of length at most  $n$ . Another incomparable characterization was given by Kapoutsis and Pighizzini [9]:  $NL \subseteq L/poly$  iff there is a polynomial  $p$  satisfying that any  $n$ -state unary 2nfa has an equivalent unary 2dfa of states at most  $p(n)$ . In this paper, we want to seek a similar automata characterization for the linear space hypothesis.

Sakoda and Sipser [10] further laid out a complexity-theoretical framework to discuss the state complexity by giving formal definitions to state-complexity based classes (such as 2D, 2N/poly, 2N/unary), each of which is generally composed of non-uniform families of languages recognized by finite automata of specified types and input sizes. Such complexity-theoretical treatments of families of finite automata were also considered by Kapoutsis [7,8] and Kapoutsis and Pighizzini [9]. For those state complexity classes, it was proven in [8,9] that  $2N/poly \subseteq 2D$  iff  $NL \subseteq L/poly$  iff  $2N/unary \subseteq 2D$ .

## 1.3 Main Contributions

As the main contribution of this paper, firstly we provide with two characterizations of 3DSTCON and PsubLIN in terms of the state complexity of finite automata, and secondly we give a characterization of LSH in terms of the state complexity of transforming a restricted form of 2nfa to another restricted form of two-way alternating finite automaton (or 2afa), which takes  $\forall$ -states and  $\exists$ -states alternatingly, producing alternating  $\forall$ -levels and  $\exists$ -levels in its (directed) computation graph made up of configurations. The significance of our characterization includes the fact that LSH can be expressed completely by the state complexity of finite automata of certain types *with no clear reference* to  $(2SAT_3, m_{vbl})$ ,  $(3DSTCON, m_{ver})$ , or even PsubLIN; therefore, this characterization may help us apply LSH to a wider range of NL-complete problems.

To describe our result precisely, we further need to explain our terminology. A *simple 2nfa* is a 2nfa having a “circular” input tape<sup>2</sup> (in which both endmarkers are located next to each other) whose tape head “sweeps” the tape (i.e. it moves only to the right), and making nondeterministic choices only at the right endmarker. For a positive integer  $c$ , a *c-branching 2nfa* makes only at most  $c$  nondeterministic choices at every step and a family of 2nfa’s is called *constant-branching* if there is a constant  $c \geq 1$  for which every 2nfa in the family is  $c$ -branching. A *c-narrow 2afa* is a 2afa whose computation graphs have width (i.e., the number of distinct vertices at a given level) at every  $\forall$ -level is bounded from above by  $c$ .

For convenience, we say that a finite automaton  $M_1$  is *equivalent* (in computational power) to another finite automaton  $M_2$  over the same input alphabet if  $M_1$  agrees with  $M_2$  on all inputs. Here, we use a straightforward binary encoding  $\langle M \rangle$  of an  $n$ -state finite automaton  $M$  using  $O(n \log n)$  bits. A family  $\{M_n\}_{n \in \mathbb{N}}$  is said to be *L-uniform* if a deterministic Turing machine (or a DTM) produces from  $1^n$  an encoding  $\langle M_n \rangle$  of finite automaton  $M_n$  using space logarithmic in  $n$ .

**Proposition 1.** *Every L-uniform family of constant-branching  $O(n \log n)$ -state simple 2nfa’s can be converted into another L-uniform family of equivalent  $O(n^{1-c/\sqrt{\log n}})$ -narrow 2afa’s with  $n^{O(1)}$  states for a certain constant  $c > 0$ .*

**Theorem 2.** *The following three statements are logically equivalent.*

1. *The linear space hypothesis fails.*
2. *For any constants  $c > 0$  and  $k \geq 1$ , there exists a constant  $\varepsilon \in [0, 1)$  such that every L-uniform family of constant-branching simple 2nfa’s with at most  $cn \log^k n$  states can be converted into another L-uniform family of equivalent  $O(n^\varepsilon)$ -narrow 2afa’s with  $n^{O(1)}$  states.*
3. *For any constant  $c > 0$ , there exists a constant  $\varepsilon \in [0, 1)$  and a log-space computable function that, on every input of an encoding of  $c$ -branching simple  $n$ -state 2nfa, produces another encoding of equivalent  $O(n^\varepsilon)$ -narrow 2afa with  $n^{O(1)}$  states.*

In addition to the original linear space hypothesis, it is possible to discuss its *non-uniform version*, which asserts that  $(2\text{SAT}_3, m_{\text{ver}})$  does not belong to a non-uniform version of PsubLIN, succinctly denoted by PsubLIN/poly.

The state complexity class 2linN consists of all families  $\{L_n\}_{n \in \mathbb{N}}$  of languages, each  $L_n$  of which is recognized by a certain  $c$ -branching simple  $O(n \log^k n)$ -state 2nfa on all inputs for appropriate constants  $c, k \in \mathbb{N}^+$ . Moreover,  $2A_{\text{narrow}(f(n))}$  is composed of language families  $\{L_n\}_{n \in \mathbb{N}}$  recognized by  $O(f(n))$ -narrow 2afa’s of  $n^{O(1)}$  states on all inputs.

**Theorem 3.** *The following three statements are logically equivalent.*

1. *The non-uniform linear space hypothesis fails.*

---

<sup>2</sup> A 2nfa with a tape head that sweeps a circular tape is called “rotating” in [9].

2. For any constant  $c > 0$ , there exists a constant  $\varepsilon \in [0, 1)$  such that every  $c$ -branching simple  $n$ -state 2nfa can be converted into an equivalent  $O(n^\varepsilon)$ -narrow 2afa with at most  $n^{O(1)}$  states.
3.  $2\text{linN} \subseteq \bigcup_{\varepsilon \in [0, 1)} 2A_{\text{narrow}(n^\varepsilon)}$ .

It is open whether  $2\text{linN}$  in Theorem 3(3) can be replaced by  $2N$  or even  $2N/\text{poly}$ . This is related to the question of whether we can replace  $2\text{SAT}_3$  in the definition of LSH by  $2\text{SAT}$  [11].

In contrast, if we focus our attention on “unary” finite automata, then we obtain a slightly weaker implication to the failure of LSH.

**Theorem 4.** *Each one of the following statements implies the failure of the linear space hypothesis.*

1. For any constants  $c > 0$  and  $k \geq 1$ , there exists a constant  $\varepsilon \in [0, 1)$  such that every  $L$ -uniform family of constant-branching simple unary 2nfa’s with at most  $cn^4 \log^k n$  states can be converted into an  $L$ -uniform family of equivalent  $O(n^\varepsilon)$ -narrow unary 2afa’s with  $n^{O(1)}$  states.
2. For any constants  $c > 0$  and  $k \geq 1$ , there exist a constant  $\varepsilon \in [0, 1)$  and a log-space computable function that, on every input of an encoding of  $c$ -branching simple unary 2nfa with at most  $cn^4 \log^k n$  states, produces another encoding of equivalent  $O(n^\varepsilon)$ -narrow unary 2afa having  $n^{O(1)}$  states.

Theorems 2–3 will be proven in Sect. 3 after we establish basic properties of  $\text{PsubLIN}$  and  $3\text{DSTCON}$  in Sect. 2. Theorem 4 will be shown in Sect. 4.

## 2 Two Basic Characterizations

Since Theorems 2–3 are concerned with  $3\text{DSTCON}$  and  $\text{PsubLIN}$ , we want to look into their basic properties. In what follows, we will present two state complexity characterizations of the complexity class  $\text{PsubLIN}$  and the language  $3\text{DSTCON}$ .

A function  $m : \Sigma^* \rightarrow \mathbb{N}^+$  is called a *log-space size parameter* if there exists a DTM  $M$  that, on any input  $x$ , produces  $m(x)$  in binary on its output tape using only  $O(\log |x|)$  work space. A function  $f : \mathbb{N} \rightarrow \mathbb{N}$  is *log-space computable* (resp.,  *$t(n)$ -time space constructible* for a given function  $t$ ) if there exists a DTM with a write-only output tape such that, for each given length  $n \in \mathbb{N}$ , when  $M$  takes  $1^n$  and then produces  $1^{f(n)}$  using  $O(\log n)$  work space (resp., within  $O(t(n))$  steps using no more than  $f(n)$  cells).

### 2.1 Automata Characterizations of $\text{PsubLIN}$

Let us give a precise characterization of  $\text{PsubLIN}$  in terms of the state complexity of narrow 2afa’s because the narrowness of 2afa’s directly corresponds to the space usage of DTMs. What we intend to prove in this section is, in fact, slightly more general than what we actually need for proving Theorems 2–3.

Take two functions  $s : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}^+$  and  $t : \mathbb{N} \rightarrow \mathbb{N}^+$ , and let  $m$  denote any log-space size parameter, where  $\mathbb{N}^+ = \mathbb{N} - \{0\}$ . We define  $\text{TIME}, \text{SPACE}(t(x), s(x, m(x)))$  (where  $x$  expresses a symbolic input) to be the collection of all parameterized decision problems  $(L, m)$  recognized by DTMs (each of which is equipped with a read-only input tape and a semi-infinite rewritable work tape) within time  $c_1 t(x)$  using space at most  $c_2 s(x, m(x))$  on every input  $x$  for certain absolute constants  $c_1, c_2 > 0$ .

Our proof of Proposition 5 is a fine-grained analysis of the well-known transformation of *alternating Turing machines* (or ATMs) to DTMs and vice versa. In what follows, we freely identify a language with its *characteristic function*.

**Proposition 5.** *Let  $t, \ell : \mathbb{N} \rightarrow \mathbb{N}^+$  be log-space computable and  $O(t(n))$ -time space constructible, respectively. Consider a language  $L$  and a log-space size parameter  $m$ .*

1. *If  $(L, m) \in \text{TIME}, \text{SPACE}(t(|x|), \ell(m(x)))$ , then there are two constants  $c_1, c_2 > 0$  and an L-uniform family  $\{M_{n,l}\}_{n,l \in \mathbb{N}}$  of  $c_2 \ell(m(x))$ -narrow 2afa's such that each  $M_{n,|x|}$  has at most  $c_1 t(|x|) \ell(m(x))$  states and computes  $L(x)$  on all inputs  $x$  satisfying  $m(x) = n$ .*
2. *If there are constants  $c_1, c_2 > 0$  and an L-uniform family  $\{M_{n,l}\}_{n,l \in \mathbb{N}}$  of  $c_2 \ell(m(x))$ -narrow 2afa's such that each  $M_{n,|x|}$  has at most  $c_1 t(|x|)$  states and computes  $L(x)$  on all inputs  $x$  satisfying  $m(x) = n$ , then  $(L, m)$  belongs to  $\text{TIME}, \text{SPACE}(t(|x|) \ell(m(x)), \ell(m(x)) + \log t(|x|) + \log |x|)$ .*

**Proof Sketch.** (1) Given a parameterized decision problem  $(L, m)$ , let us consider a DTM  $N$  that solves  $(L, m)$  in time at most  $c_1 t(|x|)$  using space at most  $c_2 \ell(m(x))$  for certain constants  $c_1, c_2 > 0$ . We first modify  $N$  so that it halts in scanning both  $\dagger$  on the input tape and the blank symbol  $B$  at the *start cell* (i.e., cell 0) of the work tape. Moreover, we make it halt in exactly  $c_1 t(|x|)$  steps. Now, we want to simulate  $N$  by 2afa's  $M_{n,l}$  of the desired type. Let  $x$  be any instance to  $L$ . Let us consider *surface configurations*  $(q, j, k, w)$  of  $N$  on  $x$ , which indicates that  $N$  is in state  $q$ , scanning both the  $j$ th cell of the input tape and the  $k$ th cell of the work tape containing  $w$ . We want to trace down these surface configurations using an alternating series of  $\forall$ -states and  $\exists$ -states of  $M_{n,|x|}$ .

Since each move of  $N$  affects at most 3 consecutive cells of the input tape and the work tape, it suffices to focus our attention on these local cells. Our idea is to define  $M_{n,|x|}$ 's surface configuration  $((q, i, k', u), j)$  to represent  $N$ 's surface configuration  $(q, j, k, w)$  at time  $i$  in such a way that  $u$  indicates either the  $k'$ -th cell content or the content of its neighboring 3 cells. In particular, when  $k = k'$ ,  $u$  carries extra information (by changing tape symbol  $\sigma$  to  $\hat{\sigma}$ ) that a tape head is at the  $k'$ th cell. For example, an initial surface configuration of  $M_{n,|x|}$  on  $x$  is  $((q_{acc}, c_1 t(|x|), 0, \hat{B}), 0)$ , which corresponds to the final accepting surface configuration of  $N$  on  $x$ , where  $q_{acc}$  is a unique accepting state of  $N$ . Inductively, we generate the next surface configuration of  $M_{n,|x|}$  roughly in the following way. In an  $\exists$ -state,  $M_{n,|x|}$  guesses (i.e., nondeterministically chooses) the content of 3 consecutive cells in the current configuration of  $N$  on  $x$ . In a  $\forall$ -state,  $M_{n,|x|}$  checks whether the guessed content is actually correct by



branching out 3 computation paths, each of which selects one of the 3 cells chosen in the  $\exists$ -state. The  $O(\ell(m(x)))$ -narrowness comes from the space bound of  $N$ .

(2) Let  $k \geq 1$  and  $\mathcal{M} = \{M_{n,l}\}_{n,l \in \mathbb{N}}$  be a family given for  $L$  by the premise of (2). In particular, each  $M_{n,l}$  is a  $c_2 \ell(m(x))$ -narrow 2afa having at most  $c_1 t(|x|)$  states for constants  $c_1, c_2 > 0$ . We simulate  $\mathcal{M}$  by the following DTM. On input  $x$ , compute  $n = m(x)$ , and generate  $\langle M_{n,|x|} \rangle$  using  $O(\log |x|)$  space. Consider a computation graph of  $M_{n,|x|}$  on input  $x$ . Using a breadth-first search technique, we check whether there is an accepting computation subgraph of  $M_{n,|x|}$  on  $x$  by trimming all encountered branches that lead to rejecting states. It is possible to carry out this procedure using space  $O(\log t(|x|)) + O(\ell(m(x))) + O(\log |x|)$  since  $M_{n,|x|}$  is  $c_2 \ell(m(x))$ -narrow and  $O(\log t(|x|))$  bits are needed to describe each state. The running time of this DTM is at most  $O(t(|x|)\ell(m(x)))$ .  $\square$

Similarly, we can obtain a non-uniform version of Proposition 5 by making use of “advice” instead of the uniformity condition. In the uniform case, we have used a DTM to produce  $\langle M_n \rangle$  from  $1^n$ ; in the non-uniform case, by contrast, we must generate  $\langle M_n \rangle$  from information given by the advice.

A Karp-Lipton non-uniform version of TIME,  $\text{SPACE}(t(x), \ell(x, m(x)))$ , which is denoted by  $\text{TIME, SPACE}(t(x), \ell(x, m(x)))/O(s(|x|))$ , is defined by supplementing external information known as “advice” to underlying Turing machines that characterize TIME,  $\text{SPACE}(t(x), \ell(x, m(x)))$ . Each of such underlying machines is equipped with an additional read-only *advice tape*, to which we provide exactly one string (called an *advice string*) of length  $O(s(n))$  surrounded by two endmarkers for all input instances of length  $n$ .

**Proposition 6.** *Let  $t : \mathbb{N} \rightarrow \mathbb{N}^+$  be log-space computable and let  $s, \ell : \mathbb{N} \rightarrow \mathbb{N}^+$  be  $O(t(n))$ -time space constructible. Let  $L$  and  $m$  be a language and a log-space size parameter, respectively. Assume that there is a function  $h$  satisfying  $|x| \leq h(m(x))$  for all  $x$ .*

1. *If  $(L, m) \in \text{TIME, SPACE}(t(|x|), \ell(m(x)))/O(s(|x|))$ , then there is a non-uniform family  $\{M_{n,l}\}_{n,l \in \mathbb{N}}$  of  $O(\ell(m(x)))$ -narrow 2afa’s such that, for each  $n \in \mathbb{N}$ ,  $M_{n,|x|}$  has  $O(t(|x|)\ell(m(x))s(|x|))$  states and computes  $L(x)$  on all inputs  $x$  satisfying  $m(x) = n$ .*
2. *If there is a non-uniform family  $\{M_{n,l}\}_{n,l \in \mathbb{N}}$  of  $O(\ell(m(x)))$ -narrow 2afa’s such that each  $M_{n,|x|}$  has  $O(t(|x|))$  states and computes  $L(x)$  on all inputs  $x$  satisfying  $m(x) = n$ , then  $(L, m)$  belongs to  $\text{TIME, SPACE}(t(|x|)\ell(m(x)), \ell(m(x)) + \log t(|x|) + \log |x|)/O(h(m(x))t(|x|)^2 \log t(|x|))$ .*

## 2.2 Automata Characterizations of 3DSTCON

The proofs of Theorems 2 and 3 require a characterization of 3DSTCON in terms of 2nfa’s. Kapoutsis [8] and Kapoutsis and Pighizzini [9] earlier gave 2nfa-characterizations of DSTCON; however, 3DSTCON needs a slightly different characterization.

First, we re-formulate the parameterized decision problem  $(3\text{DSTCON}, m_{\text{ver}})$  as a family  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$  of decision problems, each  $3\text{DSTCON}_n$  of which

is limited to directed graphs of vertex size exactly  $n$ . To express instances to  $3\text{DSTCON}_n$ , we also need to define an appropriate binary encoding of degree-bounded directed graphs. Formally, let  $K_n = (V, E)$  denote a complete directed graph with  $V = \{0, 1, \dots, n - 1\}$  and  $E = V \times V$  and let  $G = (V, E)$  be a degree-3 subgraph of  $K_n$ . We express this graph  $G$  as the form of an *adjacency list*, which is represented by an  $n \times 3$  matrix whose rows are indexed by  $i \in [n]$  and columns are indexed by  $j \in \{1, 2, 3\}$ . If there is no  $j$ th edge leaving from vertex  $i$ , then the  $(i, j)$ th entry of this list is the designated symbol  $\perp$ . We further encode this list into a single binary string, denoted by  $\langle G \rangle$ , of size  $O(n \log n)$ . Here, we demand that we can easily check whether a given string is an binary encoding of a certain directed graph.

**Lemma 7.** *There exists an L-uniform family  $\{N_n\}_{n \in \mathbb{N}}$  of  $O(n \log n)$ -state simple 2dfa's, each  $N_n$  of which checks whether any given input  $x$  is an encoding  $\langle G \rangle$  of a certain subgraph  $G$  of  $K_n$ .*

The language family  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$  is defined as follows.

Degree-3 Directed  $s$ - $t$  Connectivity Problem for Size  $n$  ( $3\text{DSTCON}_n$ ):

- Instance: an encoding  $\langle G \rangle$  of a subgraph  $G$  of the complete directed graph  $K_n$  with vertices of degree (i.e., indegree plus outdegree) at most 3.
- Output: YES if there is a path from vertex 0 to vertex  $n - 1$ ; NO otherwise.

Notice that each instance  $x$  belonging to  $3\text{DSTCON}_n$  must satisfy  $m_{\text{ver}}(x) = n$ . Clearly, the family  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$  corresponds to  $(3\text{DSTCON}, m_{\text{ver}})$ , and thus we freely identify  $(3\text{DSTCON}, m_{\text{ver}})$  with the family  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$ .

**Lemma 8.** *There is an absolute constant  $c > 0$  such that  $m_{\text{ver}}(x) \leq |x| \leq cm_{\text{ver}}(x) \log m_{\text{ver}}(x)$  for all inputs  $x$  to  $3\text{DSTCON}$ .*

Next, we want to build a uniform family of constant-branching simple 2nfa's that solve  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$ . Let  $\Sigma_n$  denote the set of all valid encodings of input graphs given to  $3\text{DSTCON}_n$ .

**Lemma 9.** *There exists an  $O(\log n)$ -space computable function  $g$  for which  $g$  produces from each  $1^n$  a description of 3-branching simple 2nfa  $N_n$  of  $O(n \log n)$  states that solves  $3\text{DSTCON}_n$  on inputs in  $\Sigma_n$  in time  $O(n^2)$ . Moreover,  $N_n$  can reject all inputs outside of  $\Sigma_n$ .*

**Proof Sketch.** Our 2nfa has a circular tape and moves its tape head only to the right. Choose any input  $x = \langle G \rangle$  to  $3\text{DSTCON}_n$ , where  $G = (V, E)$  is a degree-3 subgraph of  $K_n$  with  $V = \{0, 1, \dots, n - 1\}$ .

We design  $M_n$  so that it works *round by round* in the following way. At the first round, we assign vertex 0 in  $G$  to  $v_0$  and move the tape head rightward from  $\phi$  to  $\$$ . Now, assume by induction hypothesis that, at round  $i$  ( $\geq 0$ ), we have already chosen vertex  $v_i$  and have moved the tape head to  $\$$ . Nondeterministically, we select an index  $j \in \{1, 2, 3\}$  while scanning  $\$$ , and then deterministically search

for a row indexed  $i$  in an adjacency list of  $G$  by moving the tape head only from left to right along the circular tape. We then read the content of the  $(i, j)$ -entry of the list. If it is  $\perp$ , then reject immediately. Assuming otherwise, if  $v_{i+1}$  is the  $(i, j)$ -entry, then we update the current vertex from  $v_i$  to  $v_{i+1}$ . Whenever we reach vertex  $n - 1$ , we immediately accept  $x$  and halt. If  $M_n$  visits more than  $n$  vertices, we surely know that  $M_n$  cannot accept  $x$ .  $\square$

Let us consider the converse of Lemma 9.

**Lemma 10.** *Let  $c \in \mathbb{N}^+$  be a constant. There exists a function  $g$  such that, for every  $c$ -branching simple 2nfa  $M$  with  $n$  states,  $g$  takes input  $\langle M \rangle \# x$  and outputs an encoding  $\langle G_x \rangle$  of a subgraph  $G_x$  of  $K_{2n+3}$  of degree at most  $2(c + 1)$  satisfying that  $M$  accepts  $x$  if and only if  $G_x \in \text{3DSTCON}_{2n+3}$ . Moreover,  $g$  is computed by a certain  $n^{O(1)}$ -state simple 2dfa with a write-only output tape.*

### 3 Proofs of Theorems 2 and 3

#### 3.1 Generalizations to PTIME,SPACE( $\cdot$ )

Theorems 2 and 3(1)–(2) are concerned with PsubLIN. Nonetheless, it is possible to prove slightly more general theorems, shown below as Theorems 11 and 12, for a complexity class PTIME,SPACE( $s(x, m(x))$ ), defined in [11], which is the union of all TIME,SPACE( $p(|x|), s(x, m(x))$ ) for any positive polynomial  $p$ .

**Theorem 11.** *Let  $\mathcal{F}$  denote an arbitrary nonempty set of functions  $\ell : \mathbb{N} \rightarrow \mathbb{N}^+$  such that, for every  $\ell \in \mathcal{F}$  and every  $c > 0$  and  $k \in \mathbb{N}^+$ , there are functions  $\ell', \ell'' \in \mathcal{F}$  such that  $\ell(cn \log^k n) \leq \ell'(n)$  and  $\ell(n) + \log n^k \leq \ell''(n)$  for all  $n \in \mathbb{N}$ . Assume that, for each  $\ell \in \mathcal{F}$ ,  $\bigcup_m \text{PTIME,SPACE}(\ell(m(x)))$  is closed under short L- $m$ -reductions (see [11]), where  $m$  ranges over all log-space size parameters. The following three statements are logically equivalent.*

1. *There exists a function  $\ell \in \mathcal{F}$  such that  $(\text{3DSTCON}, m_{\text{ver}})$  is in  $\bigcup_m \text{PTIME,SPACE}(\ell(m(x)))$ .*
2. *There are a function  $\ell \in \mathcal{F}$  and two constants  $c > 0$  and  $k \in \mathbb{N}^+$  such that every L-uniform family of constant-branching simple 2nfa's with at most  $cn \log^k n$  states is converted into another L-family of  $O(\ell(n))$ -narrow 2afa's with  $n^{O(1)}$  states that agree with them on all inputs.*
3. *There are a function  $\ell \in \mathcal{F}$  and a constant  $\varepsilon \in [0, 1)$  satisfying the following: for each constant  $c \in \mathbb{N}^+$ , there exists a log-space computable function  $f$  such that  $f$  takes an input of the form  $\langle M \rangle$  for any  $c$ -branching  $n$ -state simple 2nfa  $M$  and  $f$  produces another encoding of  $O(\ell(n))$ -narrow 2afa with  $n^{O(1)}$  states that agree with  $M$  on all inputs.*

**Theorem 12.** *Let  $\mathcal{F}$  denote an arbitrary nonempty set of functions  $\ell : \mathbb{N} \rightarrow \mathbb{N}^+$  such that, for every  $\ell \in \mathcal{F}$  and every  $c > 0$  and  $k \in \mathbb{N}^+$ , there are functions  $\ell', \ell'' \in \mathcal{F}$  such that  $\ell(cn \log^k n) \leq \ell'(n)$  and  $\ell(n) + \log n^k \leq \ell''(n)$  for all  $n \in \mathbb{N}$ .*

$\mathbb{N}$ . Assume that  $\bigcup_m \text{PTIME}, \text{SPACE}(\ell(m(x)))/\text{poly}$  is closed under short L- $m$ -reductions, where  $m$  is any log-space size parameter. There is an  $\ell \in \mathcal{F}$  such that  $(3\text{DSTCON}, m_{\text{ver}})$  is in  $\bigcup_m \text{PTIME}, \text{SPACE}(\ell(m(x)))/\text{poly}$  iff, for each constant  $e \in \mathbb{N}^+$ , there are an  $\ell \in \mathcal{F}$  and a constant  $\varepsilon \in [0, 1)$  such that any  $n$ -state  $e$ -branching simple 2nfa can be converted into another  $n^{O(1)}$ -state  $O(\ell(n))$ -narrow 2afa that agrees with it on all inputs.

**Proof of Theorems 2 and 3(1)–(2).** These results follow from the fact that Theorems 2 and 3(1)–(2) are special cases of Theorems 11 and 12, respectively, where  $\ell(n)$  equals  $n^\varepsilon$  for a certain constant  $\varepsilon \in [0, 1)$ . □

Now, we return to Theorem 11 and describe its proof.

**Proof Sketch of Theorem 11.** For convenience, given a function  $\ell$ , we write  $\mathcal{C}_\ell$  for  $\bigcup_m \text{PTIME}, \text{SPACE}(\ell(m(x)))$  regarding all log-space size parameters  $m$ .

[1  $\Rightarrow$  3] Assume that  $(3\text{DSTCON}, m_{\text{ver}}) \in \mathcal{C}_\ell$  for a certain function  $\ell \in \mathcal{F}$ . Let  $c > 0$  be a constant. By applying Proposition 5(1), we obtain a constant  $k \geq 1$  and an L-uniform family  $\{N_{n,l}\}_{n,l \in \mathbb{N}}$  of  $O(\ell(m_{\text{ver}}(x)))$ -narrow 2afa's having  $O(|x|^k \cdot m_{\text{ver}}(x))$  states that agree with  $3\text{DSTCON}(x)$  on all inputs  $x$  satisfying  $m_{\text{ver}}(x) = n$ . Take a log-space computable function  $g$  that produces  $\langle N_{n,l} \rangle$  from input  $1^n \# 1^l$ . For simplicity, let  $d = 2n + 3$ . By Lemma 10, there is a function  $g$  that transforms  $\langle M \rangle \# x$  to the encoding  $\langle G_x \rangle$  of a subgraph  $G_x$  of  $K_n$  satisfying that  $M$  accepts  $x$  exactly when  $G_x \in 3\text{DSTCON}_d$ . Note that  $g$  is computed by a certain simple 2dfa with  $n^{O(1)}$  states.

Next, we want to design a log-space computable function  $f$  that transforms every  $c$ -branching  $n$ -state simple 2nfa  $M$  to another 2afa  $N$  of the desired type. We define  $f$  so that, given an encoding  $\langle M \rangle$  of a  $c$ -branching simple 2nfa  $M$  with  $n$  states, it produces an appropriate 2afa  $N$  that works as follows. On input  $x$ , generate  $\langle G_x \rangle$  from  $\langle M \rangle \# x$  by applying  $g$  and compute  $e = |\langle G_x \rangle|$ , which is  $O(n \log n)$ . Produce  $N_{d,e}$  and run it on the input  $\langle G_x \rangle$ . Note that we cannot actually write down  $\langle G_x \rangle$  onto a tape; however, since  $g$  is computed by a simple 2dfa, we can produce every bit of  $\langle G_x \rangle$  separately.

[3  $\Rightarrow$  2] Assuming (3), we obtain a log-space computable function  $g$  that, from any  $c$ -branching  $n$ -state simple 2nfa, produces an  $\ell(n)$ -narrow 2afa that agrees with it on all inputs. Let us take any L-uniform family  $\{M_n\}_{n \in \mathbb{N}}$  of simple 2nfa's, each  $M_n$  of which has at most  $cn \log^k n$  states for absolute constants  $c > 0$  and  $k \in \mathbb{N}^+$ . By the L-uniformity of  $\{M_n\}_{n \in \mathbb{N}}$ , we choose a log-space DTM  $N$  that produces  $\langle M_n \rangle$  from  $1^n$  for each  $n \in \mathbb{N}$ . By (3), we obtain an  $\ell(cn \log^k n)$ -narrow 2afa  $\langle N_n \rangle$  from  $\langle M_n \rangle$  in polynomial time using log space. Hence,  $\{N_n\}_{n \in \mathbb{N}}$  is L-uniform. Moreover, by our assumption, there is a function  $\ell' \in \mathcal{F}$  such that  $\ell(cn \log^k n) \leq \ell'(n)$  for all  $n \in \mathbb{N}$ . It thus follows that  $N_n$  is  $\ell'(n)$ -narrow.

[2  $\Rightarrow$  1] Let  $c \geq 1$ . Assume that we can convert any L-uniform family of  $c$ -branching  $cn \log^k n$ -state simple 2nfa's into another L-uniform family of  $n^{O(1)}$ -state  $O(\ell(n))$ -narrow 2afa that agrees with it on all inputs, where  $\ell \in \mathcal{F}$ . Let us consider  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$ . By Lemma 9, we obtain an L-uniform family of 3-branching simple 2nfa's  $N_n$  with  $cn \log n$  states that solve  $3\text{DSTCON}_n$  within

$cn^2$  steps on all inputs  $x$  with  $m_{ver}(x) = n$ , where  $c \geq 3$  is an appropriate constant.

Since  $m_{ver}(x) \leq |x| \leq em_{ver}(x) \log m_{ver}(x)$  for a certain constant  $e > 0$  by Lemma 8, we obtain  $|x| \leq en \log n$ . Apply (2), and we obtain 2afa's, which have  $n^{O(1)}$  states and are  $O(\ell(cn \log n))$ -narrow, solving  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$  on all inputs, including all strings  $x$  satisfying  $m_{ver}(x) = n$ . Take an  $\ell' \in \mathcal{F}$  such that  $\ell'(n) \geq \ell(cn \log n)$  for all  $n \in \mathbb{N}$ . By Proposition 5(2), we conclude that  $(3\text{DSTCON}, m_{ver})$  belongs to  $\text{TIME}, \text{SPACE}(|x|^{O(1)}, \ell'(m_{ver}(x)))$ , which is included in  $\mathcal{C}_{\ell'}$ .  $\square$

The proof of Theorem 12 is in essence similar to that of Theorem 11 except for the treatment of advice strings.

An argument similar to that of  $[1 \Rightarrow 3]$  in the proof of Theorem 11 leads to Proposition 1 on top of the result of Barnes et al. [2] on  $(\text{DSTCON}, m_{ver})$ .

### 3.2 Relationships Among State Complexity Classes

To complete the proof of Theorem 3, nevertheless, we still need to show the logical equivalence between (1) and (3) of the theorem. To achieve this goal, we first show Proposition 13, in which we present a close relationship between  $\text{PsubLIN}$  and  $\bigcup_{\varepsilon \in [0,1]} 2A_{\text{narrow}(n^\varepsilon)}$ .

**Proposition 13.** *Given a parameterized decision problem  $(L, m)$  with a log-space size parameter  $m$ , define  $L_{n,l} = \{x \in L \cap \Sigma^l \mid m(x) = n\}$  and  $\bar{L}_{n,l} = \{x \in \bar{L} \cap \Sigma^l \mid m(x) = n\}$  for each pair  $n, l \in \mathbb{N}$ . We set  $\mathcal{L} = \{(L_{n,l}, \bar{L}_{n,l})\}_{n,l \in \mathbb{N}}$ . Assume that there are constants  $c_1, c_2 > 0$  and  $k \geq 1$  for which  $c_1 m(x) \leq |x| \leq c_2 m(x) \log^k m(x)$  for all  $x$  with  $|x| \geq 2$ . It then follows that  $(L, m) \in \text{PsubLIN}/\text{poly}$  iff  $\mathcal{L} \in \bigcup_{\varepsilon \in [0,1]} 2A_{\text{narrow}(n^\varepsilon)}$ .*

**Corollary 14.**  $(3\text{DSTCON}, m_{ver}) \in \text{PsubLIN}/\text{poly}$  if and only if  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}} \in \bigcup_{\varepsilon \in [0,1]} 2A_{\text{narrow}(n^\varepsilon)}$ .

**Proof Sketch of Theorem 3(3).** Write  $\mathcal{L}$  for  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$ . By Lemma 9, we obtain  $\mathcal{L} \in 2\text{linN}$ . It is possible to prove that  $(*) 2\text{linN} \subseteq \bigcup_{\varepsilon \in [0,1]} 2A_{\text{narrow}(n^\varepsilon)}$  iff  $\mathcal{L} \in \bigcup_{\varepsilon \in [0,1]} 2A_{\text{narrow}(n^\varepsilon)}$ . The equivalence between (1) and (3) of Theorem 3 follows directly from Corollary 14 and the above statement (\*).  $\square$

## 4 Case of Unary Finite Automata

The proof of Theorem 4 needs a unary version of  $\{3\text{DSTCON}_n\}_{n \in \mathbb{N}}$ . Hence, we first define a *unary encoding* of a degree-bounded subgraph of each complete directed graph  $K_n$ . Given a degree-3 subgraph  $G = (V, E)$  of  $K_n$  with  $V = \{0, 1, 2, \dots, n - 1\}$ , the unary encoding  $\langle G \rangle_{\text{unary}}$  of  $G$  is of the form  $1^e$  with  $e = \prod_{i=1}^k p_{(i,j)}$ , where  $E = \{(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)\} \subseteq V^2$  with  $k = |E|$  and each  $p_{(i,j)}$  denotes the  $(i \cdot n + j)$ -th prime number. Since  $G$  has degree at most 3, it follows that  $k \leq 3n$ . It is known that the  $r$ th prime number is at most

$cr \log r$  for a certain constant  $c > 0$ . Since  $i \cdot n + j \leq n^2$  for all pairs  $i, j \in V$ , we conclude that  $|\langle G \rangle_{\text{unary}}| = e \leq (cn^2 \log n)^{3n}$ . Let  $\{\text{unary3DSTCON}_n\}_{n \in \mathbb{N}}$  be defined as follows.

Unary 3DSTCON of Size  $n$  ( $\text{unary3DSTCON}_n$ ):

- Instance:  $\langle G \rangle_{\text{unary}}$  for a subgraph  $G$  of  $K_n$  with vertices of degree at most 3.
- Output: YES if there is a path from vertex 0 to vertex  $n - 1$ ; NO otherwise.

**Proof Sketch of Theorem 4.** (1) Assume that every L-uniform family of  $O(n^4 \log^k n)$ -state constant-branching simple unary 2nfa's can be converted into another L-uniform family of equivalent  $n^{O(1)}$ -state  $O(n^\varepsilon)$ -narrow unary 2afa's. We then take a function  $g$  that transforms  $\langle G \rangle$  to  $\langle G \rangle_{\text{unary}}$ . Note that  $g$  can be implemented by an appropriate L-uniform family of  $n^{O(1)}$ -state simple 2dfa's. We further take a constant  $c > 0$  and an L-uniform family  $\{M_n\}_{n \in \mathbb{N}}$  of  $c$ -branching simple 2nfa's of  $O(n^4 \log n)$  states, each  $M_n$  of which solves  $\text{unary3DSTCON}_n$  on all inputs. Our assumption guarantees the existence of an L-uniform family  $\{N_n\}_{n \in \mathbb{N}}$  of  $O(n^\varepsilon)$ -narrow 2afa's with  $n^{O(1)}$  states, each  $N_n$  of which agrees with  $M_n$  on all inputs for a suitable choice of constant  $\varepsilon \in [0, 1)$ .

We want to show that the condition of Theorem 2(2) is satisfied. Let  $\langle G \rangle$  be any input to  $\text{3DSTCON}_n$ . To this input  $\langle G \rangle$ , we apply  $g$  in order to produce  $\langle G \rangle_{\text{unary}}$  and then run  $N_n$  on  $\langle G \rangle_{\text{unary}}$ . This new 2afa has  $n^{O(1)}$  states and is  $O(n^\varepsilon)$ -narrow, as we expected. Thus, the desired condition holds.

(2) Assume that, given a  $k \in \mathbb{N}^+$ , there are a constant  $\varepsilon \in [0, 1)$  and a log-space computable function  $g$  for which, on each input  $\langle M \rangle$  of  $c$ -branching simple 2nfa  $M$  with  $O(n^4 \log^k n)$  states,  $g$  outputs its equivalent  $O(n^\varepsilon)$ -narrow 2afa  $N$ . It suffices to consider the following machine. On input  $\langle G \rangle$  to  $\text{3DSTCON}_n$ , we transform it to  $\langle G \rangle_{\text{unary}}$  and run  $M_{2n+3}$  on  $\langle G \rangle_{\text{unary}}$ . This makes the condition of Theorem 2(3) true.  $\square$

## References

1. Allender, E., Chen, S., Lou, T., Papakonstantinou, P.A., Tang, B.: Width-parameterized SAT: time-space tradeoffs. *Theory Comput.* **10**, 297–339 (2014)
2. Barnes, G., Buss, J.F., Ruzzo, W.L., Schieber, B.: A sublinear space, polynomial time algorithm for directed s-t connectivity. *SIAM J. Comput.* **27**, 1273–1282 (1998)
3. Berman, P., Lingas, A.: On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Science, Warsaw (1977)
4. Chakraborty, D., Tewari, R.: Simultaneous time-space upper bounds for red-blue path problem in planar DAGs. In: Rahman, M.S., Tomita, E. (eds.) *WALCOM 2015*. LNCS, vol. 8973, pp. 258–269. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-15612-5\\_23](https://doi.org/10.1007/978-3-319-15612-5_23)
5. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic automata into simpler automata. *Theoret. Comput. Sci.* **295**, 189–203 (2003)
6. Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space. *Inform. Comput.* **209**, 1016–1025 (2011)

7. Kapoutsis, C.A.: Minicomplexity. *J. Automat. Lang. Combin.* **17**, 205–224 (2012)
8. Kapoutsis, C.A.: Two-way automata versus logarithmic space. *Theory Comput. Syst.* **55**, 421–447 (2014)
9. Kapoutsis, C.A., Pighizzini, G.: Two-way automata characterizations of L/poly versus NL. *Theory Comput. Syst.* **56**, 662–685 (2015)
10. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: *STOC 1978*, pp. 275–286 (1978)
11. Yamakami, T.: The 2CNF Boolean formula satisfiability problem and the linear space hypothesis. In: *MFCS 2017, LIPIcs*, vol. 83, pp. 62:1–62:14 (2017). [arXiv:1709.10453](https://arxiv.org/abs/1709.10453)
12. Yamakami, T.: Parameterized graph connectivity and polynomial-time sub-linear-space short reductions (preliminary report). In: Hague, M., Potapov, I. (eds.) *RP 2017. LNCS*, vol. 10506, pp. 176–191. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-67089-8\\_13](https://doi.org/10.1007/978-3-319-67089-8_13)

## Author Index

- Beier, Simon 11  
Brough, Tara 24  
Brzozowski, Janusz A. 37
- Cho, Da-Jung 49
- Dando, Louis-Marie 62  
Davies, Sylvie 37, 75
- Ferreira, Miguel 88  
Formenti, Enrico 101
- Goldwurm, Massimiliano 114  
Guillon, Bruno 126
- Han, Yo-Sub 49  
Hetzl, Stefan 139  
Holzer, Markus 11, 151
- Ibarra, Oscar H. 163  
Ibrahimov, Rishat 175
- Jirásková, Galina 188
- Keeler, Chris 200  
Khadiev, Kamil 175  
Kutrib, Martin 212
- Lin, Jianyi 114  
Lombardy, Sylvain 62
- Malcher, Andreas 212  
Massazza, Paolo 101  
McQuillan, Ian 163  
Moreira, Nelma 88
- Okhotin, Alexander 188, 224
- Prigioniero, Luca 126  
Prūsis, Krišjānis 175
- Reis, Rogério 88
- Salomaa, Kai 49, 200, 224  
Schneider, Christian 212  
Smith, Taylor J. 49  
Staiger, Ludwig 1
- Vignati, Marco 114
- Wolfsteiner, Simon 139, 151
- Yakaryilmaz, Abuzer 175  
Yamakami, Tomoyuki 237