

LNCS 10792

Shmuel Tomi Klein
Carlos Martín-Vide
Dana Shapira (Eds.)

Language and Automata Theory and Applications

12th International Conference, LATA 2018
Ramat Gan, Israel, April 9–11, 2018
Proceedings



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>

Shmuel Tomi Klein · Carlos Martín-Vide
Dana Shapira (Eds.)

Language and Automata Theory and Applications

12th International Conference, LATA 2018
Ramat Gan, Israel, April 9–11, 2018
Proceedings

Editors

Shmuel Tomi Klein 

Bar-Ilan University

Ramat Gan

Israel

Dana Shapira 

Ariel University

Ariel

Israel

Carlos Martín-Vide 

Rovira i Virgili University

Tarragona

Spain

ISSN 0302-9743

ISSN 1611-3349 (electronic)

Lecture Notes in Computer Science

ISBN 978-3-319-77312-4

ISBN 978-3-319-77313-1 (eBook)

<https://doi.org/10.1007/978-3-319-77313-1>

Library of Congress Control Number: 2018934364

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

© Springer International Publishing AG, part of Springer Nature 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

These proceedings contain the papers that were presented at the 12th International Conference on Language and Automata Theory and Applications (LATA 2018), held at Bar-Ilan University near Tel Aviv, Israel, during April 9–11, 2018.

The scope of LATA is rather broad, including: algebraic language theory, algorithms for semi-structured data mining, algorithms on automata and words, automata and logic, automata for system analysis and program verification, automata networks, automatic structures, codes, combinatorics on words, computational complexity, concurrency and Petri nets, data and image compression, descriptional complexity, foundations of finite-state technology, foundations of XML, grammars (Chomsky hierarchy, contextual, unification, categorial, etc.), grammatical inference and algorithmic learning, graphs and graph transformation, language varieties and semigroups, language-based cryptography, mathematical and logical foundations of programming methodologies, parallel and regulated rewriting, parsing, patterns, power series, string processing algorithms, symbolic dynamics, term rewriting, transducers, trees, tree languages and tree automata, weighted automata.

LATA 2018 received 58 submissions. Every paper was reviewed by three Programme Committee members. There were also a few external experts consulted. After a thorough and vivid discussion phase, the committee decided to accept 20 papers (which represents a competitive acceptance rate of about 34%). The conference program included five invited talks as well.

The excellent facilities provided by the EasyChair conference management system allowed us to deal with the submissions successfully and handle the preparation of these proceedings in time.

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

January 2018

Shmuel Tomi Klein
Carlos Martín-Vide
Dana Shapira

Organization

Program Committee

Christel Baier	Technical University of Dresden, Germany
Pablo Barceló	University of Chile, Chile
Francine Blanchet-Sadri	University of North Carolina, Greensboro, USA
Alexander Clark	King's College London, UK
Frank Drewes	Umeå University, Sweden
Manfred Droste	University of Leipzig, Germany
Dora Giammarresi	University of Rome Tor Vergata, Italy
Erich Grädel	RWTH Aachen University, Germany
Peter Habermehl	Paris Diderot University, France
Jeffrey Heinz	Stony Brook University, USA
Pedro Rangel Henriques	University of Minho, Portugal
Hendrik Jan Hoogeboom	Leiden University, The Netherlands
Christian Honer zu Siederdissen	University of Leipzig, Germany
David Janin	University of Bordeaux, France
Marcin Jurdziński	University of Warwick, UK
Makoto Kanazawa	National Institute of Informatics, Japan
Jarkko Kari	University of Turku, Finland
Shmuel Tomi Klein	Bar-Ilan University, Israel
Salvatore La Torre	University of Salerno, Italy
Salvador Lucas	Polytechnic University of Valencia, Spain
Stuart W. Margolis	Bar-Ilan University, Israel
Carlos Martín-Vide (Chair)	Rovira i Virgili University, Spain
Fernando Orejas	Polytechnic University of Catalonia, Spain
Paritosh K. Pandya	Tata Institute of Fundamental Research, India
Gennaro Parlato	University of Southampton, UK
Dominique Perrin	University Paris-Est Marne-la-Vallée, France
Detlef Plump	University of York, UK
Matteo Pradella	Polytechnic University of Milan, Italy
Arend Rensink	University of Twente, The Netherlands
Kai Salomaa	Queen's University, Canada
Sven Schewe	University of Liverpool, UK
Helmut Seidl	Technical University of Munich, Germany
William F. Smyth	McMaster University, Canada
Jiri Srba	Aalborg University, Denmark
Benjamin Steinberg	City College of New York, USA
Frank Stephan	National University of Singapore, Singapore
K. G. Subramanian	University Sains Malaysia, Malaysia

Klaus Sutner	Carnegie Mellon University, USA
Menno van Zaanen	Tilburg University, The Netherlands
Margus Veannes	Microsoft Research, USA
Eric Villemonte de la Clergerie	Inria, France
Mahesh Viswanathan	University of Illinois, Urbana-Champaign, USA
Mikhail Volkov	Ural State University, Russia

Additional Reviewers

Baldan, Paolo	Löding, Christof
Basset, Nicolas	Maletti, Andreas
Berglund, Martin	Mandrioli, Dino
Björklund, Johanna	Manea, Florin
Blair, Dakota	Mardare, Radu
Blumensath, Achim	Margolis, Stuart
Carmona, Josep	Minas, Mark
Carreras, Xavier	Mitrana, Victor
Case, John	Morzenti, Angelo
Ciobanu, Laura	Nolte, Dennis
Crespi Reghizzi, Stefano	Peltomäki, Jarkko
Daviaud, Laure	Rossi, Matteo
Enea, Constantin	Ryzhikov, Andrew
Fici, Gabriele	Salo, Ville
Gawrychowski, Paweł	Sears, David
Julian-Iranzo, Pascual	Sharan, Shambhu
Kobele, Greg	Shur, Arseny
Koppitz, Jörg	Steffinlongo, Enrico
Krishna, S.	Tao, Jim
König, Barbara	Vatter, Vincent
Lee, Edmond	Vogler, Walter
Lohrey, Markus	

Contents

Constraint Satisfaction Problems: Complexity and Algorithms	1
<i>Andrei A. Bulatov</i>	
Sliding Window Algorithms for Regular Languages	26
<i>Moses Ganardi, Danny Hucke, and Markus Lohrey</i>	
Underlying Principles and Recurring Ideas of Formal Grammars	36
<i>Alexander Okhotin</i>	
Reshaping the Context-Free Model: Linguistic and Algorithmic Aspects	60
<i>Eli Shamir</i>	
Formal Languages over GF(2)	68
<i>Ekaterina Bakinova, Artem Basharin, Igor Batmanov, Konstantin Lyubort, Alexander Okhotin, and Elizaveta Sazhneva</i>	
Event-Clock Nested Automata	80
<i>Laura Bozzelli, Aniello Murano, and Adriano Peron</i>	
On the Synchronization of Planar Automata	93
<i>J. Andres Montoya and Christian Nolasco</i>	
Descriptional and Computational Complexity of the Circuit Representation of Finite Automata	105
<i>Māris Valdats</i>	
Disturbance Decoupling in Finite Automata	118
<i>Alexey Zhirabok and Alexey Shumsky</i>	
Default Logic and Bounded Treewidth	130
<i>Johannes K. Fichte, Markus Hecher, and Irina Schindler</i>	
A General Class of Monoids Supporting Canonisation and Minimisation of (Sub)sequential Transducers	143
<i>Stefan Gerdjikov</i>	
Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy	156
<i>Demen Güler, Andreas Krebs, Klaus-Jörn Lange, and Petra Wolf</i>	
Learners Based on Transducers	169
<i>Sanjay Jain, Shao Ning Kuek, Eric Martin, and Frank Stephan</i>	

Model Learning as a Satisfiability Modulo Theories Problem	182
<i>Rick Smetsers, Paul Fiterău-Broștean, and Frits Vaandrager</i>	
Analytic Combinatorics of Lattice Paths with Forbidden Patterns: Enumerative Aspects	195
<i>Andrei Asinowski, Axel Bacher, Cyril Banderier, and Bernhard Gittenberger</i>	
Bubble-Flip—A New Generation Algorithm for Prefix Normal Words	207
<i>Ferdinando Cicalese, Zsuzsanna Lipták, and Massimiliano Rossi</i>	
Permutations Sorted by a Finite and an Infinite Stack in Series	220
<i>Murray Elder and Yoong Kuan Goh</i>	
On Periodicity Lemma for Partial Words	232
<i>Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń</i>	
Measuring Closeness Between Cayley Automatic Groups and Automatic Groups	245
<i>Dmitry Berdinsky and Phongpitak Trakuldit</i>	
Pomsets and Unfolding of Reset Petri Nets	258
<i>Thomas Chatain, Maurice Comlan, David Delfieu, Loïg Jezequel, and Olivier H. Roux</i>	
Timed Comparisons of Semi-Markov Processes	271
<i>Mathias R. Pedersen, Nathanaël Fijalkow, Giorgio Bacci, Kim G. Larsen, and Radu Mardare</i>	
Handling Ties Correctly and Efficiently in Viterbi Training Using the Viterbi Semiring	284
<i>Markus Saers and Dekai Wu</i>	
Over-Approximative Petri Net Synthesis for Restricted Subclasses of Nets	296
<i>Uli Schlachter</i>	
Efficient Translation with Linear Bimorphisms	308
<i>Christoph Teichmann, Antoine Venant, and Alexander Koller</i>	
Author Index	321



Constraint Satisfaction Problems: Complexity and Algorithms

Andrei A. Bulatov^(✉)

School of Computing Science, Simon Fraser University,
8888 University Drive, Burnaby, Canada
abulatov@sfu.ca

Abstract. In this paper we briefly survey the history of the Dichotomy Conjecture for the Constraint Satisfaction problem, that was posed 25 years ago by Feder and Vardi. We outline some of the approaches to this conjecture, and then describe an algorithm that yields an answer to the conjecture.

1 Constraint Satisfaction Problem

We begin with definitions, examples and brief historical remarks on the Constraint Satisfaction Problem.

1.1 The Problem

The archetypal example of the Constraint Satisfaction Problem is a Sudoku puzzle, see, Fig. 1: One needs to assign values to every cell of the puzzle so that the assignment satisfies certain constraints, such as the values in every row, column, and smaller block are different. This example can be naturally generalized in the following way. In the definition below tuples of elements are denoted in boldface, say, \mathbf{a} , and the i th component of \mathbf{a} is referred to as $\mathbf{a}[i]$.

Definition 1. Let A_1, \dots, A_n be finite sets. An instance \mathcal{I} of the Constraint Satisfaction Problem (CSP for short) over A_1, \dots, A_n consists of a finite set of variables V such that each $v \in V$ is assigned a domain A_{i_v} , $i_v \in \{1, \dots, n\}$, and a finite set of constraints \mathcal{C} . Each constraint is a pair $\langle \mathbf{s}, R \rangle$ where R is a relation over A_1, \dots, A_n (say, k -ary), often called the constraint relation, and \mathbf{s} is a k -tuple of variables from V , called the constraint scope. Let $\sigma : V \rightarrow A = A_1 \cup \dots \cup A_n$ with $\sigma(v) \in A_{i_v}$; we write $\sigma(\mathbf{s})$, for $(\sigma(\mathbf{s}[1]), \dots, \sigma(\mathbf{s}[k]))$. A solution of \mathcal{I} is a mapping $\sigma : V \rightarrow A$ such that for every constraint $\langle \mathbf{a}, R \rangle \in \mathcal{C}$ we have $\sigma(\mathbf{s}) \in R$. The objective in the CSP is to decide whether or not a solution of a given instance \mathcal{I} exists.

This research was supported by an NSERC Discovery grant.

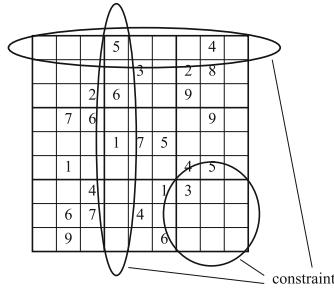


Fig. 1. A Sudoku puzzle as a CSP. The ellipses indicate some of the constraints

Since its inception in the early 70s [59], the CSP has become a very popular and powerful framework, widely used to model computational problems first in artificial intelligence, [34] and later in many other areas.

Modeling a specific computational problem usually gives rise to a *restricted CSP*. Such restrictions can be imposed either on the type of the allowed constraint relations, or on the way the constraint scopes interact, or both. Restrictions of the first kind—the main subject of this paper—are usually given by specifying a constraint language, that is, a set of relations Γ over a set, or a collection of sets such that every constraint relation has to belong to Γ . More formally, let A_1, \dots, A_ℓ be finite sets and Γ a set (finite or infinite) of relations over A_1, \dots, A_ℓ , called a *constraint language*. Then $\text{CSP}(\Gamma)$ is the class of all instances \mathcal{I} of the CSP such that $R \in \Gamma$ for every constraint $\langle s, R \rangle$ from \mathcal{I} . The following examples are just a few of the problems representable as $\text{CSP}(\Gamma)$.

***k*-COL** The standard *k*-Coloring problem has the form $\text{CSP}(\Gamma_{k\text{-COL}})$, where $\Gamma_{k\text{-COL}} = \{\neq_k\}$ and \neq_k is the disequality relation on a *k*-element set (of colours).

3-SAT An instance of the 3-SAT problem is a propositional logic formula in CNF each clause of which contains 3 literals, and asking if it has a satisfying assignment. Thus, 3-SAT is equivalent to $\text{CSP}(\Gamma_{3\text{SAT}})$, where $\Gamma_{3\text{SAT}}$ is the constraint language on $\{0, 1\}$ and containing relations R_1, \dots, R_8 , which are the 8 ternary relations that can be expressed by a 3-clause.

LIN Let F be a finite field and let $3\text{LIN}(F)$ be the problem of deciding the consistency of a system of linear equations over F each of which contains at most 3 variables. Then $3\text{LIN}(F)$ is equivalent to $\text{CSP}(\Gamma_{3\text{LIN}(F)})$, where $\Gamma_{3\text{LIN}(F)}$ is the constraint language over F whose relations are given by an equation with at most 3 variables.

MONEQ Let M be a monoid (or a semigroup). An equation over M is an expression of the form $t = s$, where t and s are words that involve indeterminates and constants from M . A solution of $t = s$ is an assignment of elements from M to the indeterminates such that t and s evaluate to the same element of M . In the problem $\text{MONEQ}(M)$ we are given

a system of equations over monoid M , and the objective is to decide, whether or not there exists an assignment to the indeterminates that is a solution for each of the given equations. Similar to 3LIN, $\text{MONEQ}(M)$ is the problem $\text{CSP}(\Gamma_{\text{MONEQ}(M)})$, where $\Gamma_{\text{MONEQ}(M)}$ is the constraint language consisting of all relations representable by an equation over M . Note that $\Gamma_{\text{MONEQ}(M)}$ is infinite in general.

1.2 Logic and Databases

The next step in the CSP research was motivated by its applications in the theory of relational databases. The **QUERY EVALUATION** problem can be thought of as deciding whether a first order sentence in the vocabulary of a database is true in that database (that is, whether or not the query has an answer). The **QUERY CONTAINMENT** problem asks, given two queries Φ and Ψ , whether $\Phi \rightarrow \Psi$ is true in any database with the appropriate vocabulary. The former problem is of course the main problem relational databases are needed for, while the latter is routinely used in various query optimization techniques. It turns out that both problems have intimate connections to the CSP, if the CSP is properly reformulated. We need some terminology from model theory.

A *vocabulary* is a finite set of relational symbols R_1, \dots, R_n each of which has a fixed arity $\text{ar}(R_i)$. A *relational structure* over vocabulary R_1, \dots, R_n is a tuple $\mathcal{H} = (H; R_1^{\mathcal{H}}, \dots, R_n^{\mathcal{H}})$ such that H is a non-empty set, called the *universe* of \mathcal{H} , and each $R_i^{\mathcal{H}}$ is a relation over H having the same arity as the symbol R_i . A sentence is said to be a *conjunctive query* if it only uses existential quantifiers and its quantifier-free part is a conjunction of atomic formulas.

Definition 2. *An instance of the CSP is a pair (Φ, \mathcal{H}) , where \mathcal{H} is a relational structure in a certain vocabulary, and Φ is a conjunctive sentence in the same vocabulary. The objective is to decide whether Φ is true in \mathcal{H} .*

To see that the definition above is equivalent to the original definition of the CSP, we consider its special case, k -COLOURING. The vocabulary corresponding to the problem contains just one binary predicate R_{\neq} . Let \mathcal{H}_k be the relational structure with universe $[k] = \{1, \dots, k\}$ in the vocabulary $\{R_{\neq}\}$, where $R_{\neq}^{\mathcal{H}_k}$ is interpreted as the disequality relation on the set $[k]$. (In the future we will tend to omit the superscripts indicating an interpretation, whenever it does not lead to a confusion.) Then an instance $G = (V, E)$ of k -COLOURING is equivalent to testing whether conjunctive sentence $\bigwedge_{(u,v) \in E} R_{\neq}(u, v)$ (we omit the quantifier prefix) is true in \mathcal{H} .

The **QUERY EVALUATION** problem is thus just the CSP, when restricted to conjunctive queries. A database is then cosidered as the input relational structure. The Chandra-Merlin Theorem [29] shows that the **QUERY CONTAINMENT** problem is also equivalent to the CSP.

Relational database theory also massively contributed to the CSP research, most notably by techniques related to local propagation algorithms and the logic language Datalog. We will return to this subject in Sect. 3.1.

1.3 Homomorphisms and Dichotomy

The complexity of the CSP and its solution algorithms have been a major theme since the problem was introduced. The general CSP is NP-complete, as it can be easily shown. However, various restrictions of the CSP may result in more tractable problems. Papers [36, 37] by Feder and Vardi marked the beginning of a systematic research of the complexity of the CSP. Among the numerous insights of this paper, it introduced a new definition of the CSP.

Let \mathcal{G} and \mathcal{H} be relational structures over the same vocabulary. A *homomorphism* from \mathcal{G} to \mathcal{H} is a mapping $\varphi: G \rightarrow H$ from the universe G of \mathcal{G} (the *instance*) to the universe H of \mathcal{H} (the *template*) such that, for every relation $R^{\mathcal{G}}$ of \mathcal{G} and every tuple $\mathbf{a} \in R^{\mathcal{G}}$, we have $\varphi(\mathbf{a}) \in R^{\mathcal{H}}$.

Definition 3. An instance of the CSP is a pair of relational structures \mathcal{G}, \mathcal{H} over the same vocabulary. The objective is to decide whether or not there exists a homomorphism from \mathcal{G} to \mathcal{H} .

The homomorphic definition of the CSP makes its restricted version very elegant. Let \mathcal{H} be a relational structure. An instance of the *nonuniform* constraint satisfaction problem $\text{CSP}(\mathcal{H})$ is a structure \mathcal{G} over the same vocabulary as \mathcal{H} , and the question is whether there is a homomorphism from \mathcal{G} to \mathcal{H} .

We again illustrate the correspondence between the definition above and Definition 1 with an example. Consider again the k -COLOURING problem, and let \mathcal{H}_k denote the relational structure with universe $[k]$ over vocabulary $\{R_{\neq}\}$ and $R_{\neq}^{\mathcal{H}_k}$ is interpreted as the disequality relation. In other words, $\mathcal{H}_k = K_k$ is a complete graph with k vertices. Then a homomorphism from a given graph $G = (V, E)$ to K_k exists if and only if it is possible to assign vertices of K_k (colours) to vertices of G in such a way that for any $(u, v) \in E$ the vertices u and v are assigned different colours. The latter is just a proper k -colouring of G .

Using the homomorphism framework the k -COLOURING problem can be generalized to the H -COLOURING problem, where H is a graph or digraph: Given a (di)graph G , decide whether or not there is a homomorphism from G to H . Using the CSP notation the H -COLOURING is $\text{CSP}(E_H)$, where E_H denotes the edge relation of H . The H -COLOURING problem has received much attention in graph theory, see, e.g. [48, 49].

Feder and Vardi in [36, 37] also initiated the line of research that is central for this paper, the study of the complexity of nonuniform CSPs. They observed that in all known cases a nonuniform CSP either can be solved in polynomial time, e.g. $\text{CSP}(\Gamma_{3\text{LIN}})$ or 2-COLOURING, or is NP-complete, e.g., 3-SAT or k -COLOURING for $k > 2$. Two results were quite suggestive at that point. The first one is the classification of the complexity of $\text{CSP}(\mathcal{H})$ for 2-element structures (or the GENERALIZED SATISFIABILITY problem, as it was referred to) by Schaefer [68]; who proved that every such problem is either solvable in polynomial time, or is NP-complete. The second result by Hell and Nešetřil [49] establishes that the H -COLOURING problem, where H is a graph, follows the same pattern: The H -COLOURING problem can be solved in polynomial time if H is bipartite or

has a loop, and it is NP-complete otherwise. This allowed Feder and Vardi to pose the following

Conjecture 4 (The Dichotomy Conjecture). For every finite relational structure \mathcal{H} the problem $\text{CSP}(\mathcal{H})$ is either solvable in polynomial time or is NP-complete.

Most of the remaining part of this paper is devoted to resolving the Dichotomy Conjecture.

1.4 The Other Side and Other Types

In nonuniform CSPs we restrict a constraint language or a template relational structure. Clearly, other kinds of restrictions are also possible. For instance, in database theory one cannot assume any restrictions on the possible content of a database—which is a template structure in the CONJUNCTIVE QUERY EVALUATION problem—but some restrictions on the possible form of queries make much sense. If a CSP is viewed as in Definition 1, the constraint scopes of an instance \mathcal{I} form a hypergraph on the set of variables. In a series of works [40, 43, 44, 46, 47] it has been shown that if this hypergraph allows some sort of decomposition, or is tree-like, then the CSP can be solved in polynomial time. The tree-likeness of a hypergraph is usually formalized as having bounded treewidth, or bounded hypertree width, or bounded fractional hypertree width. This line of work culminated in [64], in which Marx gave an almost tight description of classes of hypergraphs that give rise to a CSP solvable in polynomial time. *Hybrid* restrictions are also possible, although research in this direction has been more limited, see, [30, 38, 39] as an example.

Along with the decision version of the CSP, other versions of the problem have been intensively studied, see, [31] for definitions and early results on many of them. These include the Quantified CSP, which is the problem of checking whether or nor a conjunctive sentence allowing both universal and existential quantifiers is true in a given relational structure [11]. In the MaxCSP one needs to maximize the number of satisfied constraints. Note that often constraints in MaxCSP are considered weighted and the the problem is to maximize the total weight of satisfied constraints. Another variation of this problem is Valued CSP, in which the constraints are replaced by functions that give a weight to each assignment. The problem is to minimize (or maximize) the weight of an assignment. This problem has been considered for both exact optimization [56, 57, 69] and approximation algorithms [5, 33, 66]. The Counting CSP has received much attention, particularly due to its connections to statistical physics. In this problem the goal is to count the number of solutions of a CSP (the unweighted version) or to evaluate the total weight of the assignments (the weighted version). The complexity of exact counting is well understood [18, 28, 35], while approximate counting remains a largely open area [54].

2 Algebraic Approach

The most successful approach to tackling the Dichotomy Conjecture turned out to be the algebraic one. In this section we introduce the algebraic approach to the

CSP and show how it can be used to determine the complexity of nonuniform CSPs. A keen reader can find more details on the algebraic approach, its applications, and the underlying algebraic facts from the following books [45, 50], surveys [6, 7, 26, 27], and research papers [2–4, 9, 16, 17, 19, 24, 25, 51].

2.1 Primitive Positive Definitions

Let Γ be a set of relations (predicates) over a finite set A . A relation R over A is said to be *primitive-positive (pp-) definable* in Γ if $R(\mathbf{x}) = \exists \mathbf{y} \Phi(\mathbf{x}, \mathbf{y})$, where Φ is a conjunction that involves predicates from Γ and equality relations. The formula above is then called a *pp-definition* of R in Γ . A constraint language Δ is pp-definable in Γ if so is every relation from Δ . In a similar way pp-definability can be introduced for relational structures.

Example 5. Let $K_3 = ([3], E)$ be a 3-element complete graph. Its edge relation is the binary disequality relation on $[3]$. Then the pp-formula

$$\begin{aligned} Q(x, y, z) = \exists t, u, v, w (E(t, x) \wedge E(t, y) \wedge E(t, z) \wedge E(u, v) \wedge E(v, w) \\ \wedge E(w, u) \wedge E(u, x) \wedge E(v, y) \wedge E(w, z)) \end{aligned}$$

defines the relation Q that consists of all triples containing exactly 2 different elements from $[3]$.

A link between pp-definitions and reducibility between nonuniform CSPs was first observed in [52].

Theorem 6 ([52]). *Let Γ and Δ be constraint languages and Δ finite. If Δ is pp-definable in Γ then $\text{CSP}(\Delta)$ is polynomial time reducible¹ to $\text{CSP}(\Gamma)$.*

It was later shown that pp-definability in Theorem 6 can be replaced with a more general notion of *pp-constructibility* [7, 8].

2.2 Polymorphisms and Invariants

Primitive positive definability can be concisely characterized using polymorphisms. An operation $f : A^k \rightarrow A$ is said to be a *polymorphism* of a relation $R \subseteq A^n$ if for any $\mathbf{a}_1, \dots, \mathbf{a}_k \in R$ the tuple $f(\mathbf{a}_1, \dots, \mathbf{a}_k)$ also belongs to R , where $f(\mathbf{a}_1, \dots, \mathbf{a}_k)$ stands for $(f(\mathbf{a}_1[1], \dots, \mathbf{a}_k[1]), \dots, f(\mathbf{a}_1[n], \dots, \mathbf{a}_k[n]))$. Operation f is a polymorphism of a constraint language Γ if it is a polymorphism of every relation from Γ . Similarly, operation f is a polymorphism of a relational structure \mathcal{H} if it is a polymorphism of every relation of \mathcal{H} . The set of all polymorphisms of language Γ or relational structure \mathcal{H} is denoted by $\text{Pol}(\Gamma)$, $\text{Pol}(\mathcal{H})$. If F is a set of operations, $\text{Inv}(F)$ denotes the set of all relations R such that every operation from F is a polymorphism of R .

¹ In fact, due to the result of [67] this reduction can be made log-space.

Example 7. Let R be an affine relation, that is, R is the solution space of a system of linear equations over a field F . Then the operation $f(x, y, z) = x - y + z$ is a polymorphism of R . Indeed, let $A \cdot \mathbf{x} = \mathbf{b}$ be the system defining R , and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$. Then

$$A \cdot f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A \cdot (\mathbf{x} - \mathbf{y} + \mathbf{z}) = A \cdot \mathbf{x} - A \cdot \mathbf{y} + A \cdot \mathbf{z} = \mathbf{b}.$$

In fact, the converse can also be shown: if R is invariant under f , where f is defined in a certain finite field F then R is the solution space of some system of linear equations over F .

Example 8. In [55] it was shown that $\text{MONEQ}(M)$ for a monoid M can be solved in polynomial time if and only if M is commutative and is the union of its subgroups. If this is the case then the operation $t(x, y, z) = xy^{\omega-1}z$ is a polymorphism of $\Gamma_{\text{MONEQ}(M)}$ (see also [58]). Here x^ω denotes the power of x such that x^ω is an idempotent of M .

Several other useful polymorphisms are the following

Example 9 ([24, 52, 53]). (1) A binary semilattice operation.

(2) A k -ary operation g on A is called a *near-unanimity* operation, or NU if

$$g(y, x, \dots, x) = g(x, y, x, \dots, x) = \dots = g(x, \dots, x, y) = x$$

for any $x, y \in A$. A ternary NU is also referred to as a *majority* operation.

(3) A k -ary operation g on A is called a *weak near-unanimity* operation, or WNU if it satisfies all the equations of an NU except for the last one

$$g(y, x, \dots, x) = g(x, y, x, \dots, x) = \dots = g(x, \dots, x, y).$$

(4) A ternary operation h on A is called *Mal'tsev* if

$$h(x, y, y) = h(y, y, x) = x$$

for any $x, y \in A$. As we saw in Example 7 any structure whose relations can be represented by linear equations has the Mal'tsev polymorphism $x - y + z$ where $+$ and $-$ are the operations of the underlying field. Note that the operation $xy^{\omega-1}z$ from Example 8 is not necessarily Mal'tsev.

- (5) If every polymorphism f of a relational structure \mathcal{H} is such that $f(x_1, \dots, x_n) = x_i$ for some i and all $x_1, \dots, x_n \in H$, then $\text{CSP}(\mathcal{H})$ is NP-complete.
- (6) Schaefer's Theorem [68] can be stated in terms of polymorphisms. Let \mathcal{H} be a 2-element relational structure (we assume its universe to be $\{0, 1\}$). The problem $\text{CSP}(\mathcal{H})$ is solvable in polynomial time if and only if one of the following operations is a polymorphism of \mathcal{H} : the constant operations 0 or 1, the semilattice operations of conjunction and disjunction, the majority operation on $\{0, 1\}$ (there is only one such operation), or the Mal'tsev operation $x - y + z$ where $+$ and $-$ are modulo 2. Otherwise $\text{CSP}(\mathcal{H})$ is NP-complete.

A link between polymorphisms and pp-definability of relations is given by *Galois connection*.

Theorem 10 (Galois connection, [10, 42]). *Let Γ be a constraint language on A , and let $R \subseteq A^n$ be a non-empty relation. Then R is preserved by all polymorphisms of Γ if and only if R is pp-definable in Γ .*

2.3 Algebras and the CSP

Recall that a (*universal*) *algebra* is an ordered pair $\mathbb{A} = (A, F)$ where A is a non-empty set, called the *universe* of \mathbb{A} , and F is a family of finitary operations on A , called the *basic operations* of \mathbb{A} . Operations that can be obtained from F by means of composition are said to be *term* operations of the algebra. Every constraint language on a set A can be associated with an algebra $\text{Alg}(\Gamma) = (A, \text{Pol}(\Gamma))$. In a similar way any relational structure \mathcal{A} (with universe A) can be paired up with the algebra $\text{Alg}(\mathcal{A}) = (A, \text{Pol}(\mathcal{A}))$. On the other hand, an algebra $\mathbb{A} = (A, F)$, can be associated with the constraint language $\text{Inv}(F)$ or the class $\text{Str}(\mathbb{A})$ of structures $\mathcal{A} = (A, R_1, \dots, R_k)$ such that $R_1, \dots, R_k \in \text{Inv}(F)$.

This correspondence can be extended to CSPs: For an algebra \mathbb{A} by $\text{CSP}(\mathbb{A})$ we denote the class of problems $\text{CSP}(\mathcal{A})$, $\mathcal{A} \in \text{Str}(\mathbb{A})$. Equivalently, $\text{CSP}(\mathbb{A})$ can be thought of as $\text{CSP}(\text{Inv}(F))$ for the infinite constraint language $\text{Inv}(F)$. Note, however, that there is a subtle difference in the notion of polynomial time solvability in these two cases that we will address next.

We say that algebra \mathbb{A} is *tractable* if every $\text{CSP}(\mathcal{A})$, $\mathcal{A} \in \text{Str}(\mathbb{A})$, is solvable in polynomial time. Observe that this does not guarantee that there is a single solution algorithm for all such problems, nor it guarantees that there is any uniformity among those algorithms. In general, it is plausible that for a tractable algebra $\mathbb{A} = (A, F)$ the problem $\text{CSP}(\text{Inv}(F))$ is NP-hard. If the problem $\text{CSP}(\text{Inv}(F))$ is solvable in polynomial time, we call \mathbb{A} *globally tractable*. Algebra \mathbb{A} is called *NP-complete* if some $\text{CSP}(\mathcal{A})$, $\mathcal{A} \in \text{Str}(\mathbb{A})$ is NP-complete. Algebra \mathbb{A} is *globally NP-complete* if $\text{CSP}(\text{Inv}(F))$ is NP-complete.

Using the algebraic terminology we can pose a stronger version of the Dichotomy Conjecture.

Conjecture 11 (Dichotomy Conjecture+). Every finite algebra is either globally tractable or NP-complete (in the local sense).

Our next goal is to make Conjecture 11 more precise. We achieve this goal in Sect. 2.4, while now we observe that the standard algebraic constructions behave quite well with respect to reducibility between CSPs.

Theorem 12 ([25]). *Let $\mathbb{A} = (A; F)$ be a finite algebra. Then*

- (1) *if \mathbb{A} is tractable then so is every subalgebra, homomorphic image, and direct power of \mathbb{A} .*
- (2) *if \mathbb{A} has a NP-hard subalgebra, homomorphic image, or direct power, then \mathbb{A} is NP-hard.*

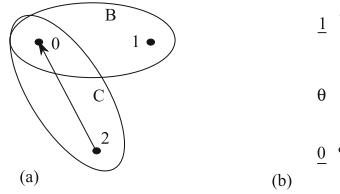


Fig. 2. (a) Algebra \mathbb{A}_M . (b) The congruence lattice of \mathbb{A}_M

More reducibility properties related to term operations of an algebra can be proved. Recall that an operation f on a set A is said to be *idempotent* if the equality $f(x, \dots, x) = x$ holds for all $x \in A$. An algebra all of whose term operations are idempotent is said to be *idempotent*.

Theorem 13 ([25]). *For any finite algebra \mathbb{A} there is an idempotent finite algebra \mathbb{B} such that:*

- \mathbb{A} is globally tractable if and only if \mathbb{B} is globally tractable;
- \mathbb{A} is NP-complete if and only if \mathbb{B} is NP-complete.

Theorem 13 reduces the Dichotomy Conjecture+ 11 to idempotent algebras.

Example 14. The next example will be our running example throughout the paper. Let $A = \{0, 1, 2\}$, and let \mathbb{A}_M be the algebra with universe A and two basic operations: a binary operation r such that $r(0, 0) = r(0, 1) = r(2, 0) = r(0, 2) = r(2, 1) = 0$, $r(1, 1) = r(1, 0) = r(1, 2) = 1$, $r(2, 2) = 2$; and a ternary operation t such that $t(x, y, z) = x - y + z$ if $x, y, z \in \{0, 1\}$, where $+, -$ are the operations of \mathbb{Z}_2 , $t(2, 2, 2) = 2$, and otherwise $t(x, y, z) = t(x', y', z')$, where $x' = x$ if $x \in \{0, 1\}$ and $x' = 0$ if $x = 2$; the values y', z' are obtained from y, z by the same rule. It is an easy exercise to verify the following facts: (a) $\mathbb{B} = (\{0, 1\}, r|_{\{0,1\}}, t|_{\{0,1\}})$ and $\mathbb{C} = (\{0, 2\}, r|_{\{0,2\}}, t|_{\{0,2\}})$ are subalgebras of \mathbb{A}_M , (b) the partition $\{0, 1\}, \{2\}$ is a congruence of \mathbb{A}_M , let us denote it θ , (c) algebra \mathbb{C} is basically a semilattice, that is, a set with a semilattice operation, see Fig. 2(a).

The classes of congruence θ are $0^\theta = \{0, 1\}, 2^\theta = \{2\}$. Then the quotient algebra \mathbb{A}_M/θ is also basically a semilattice, as $r/\theta(0^\theta, 0^\theta) = r/\theta(0^\theta, 2^\theta) = r/\theta(2^\theta, 0^\theta) = 0^\theta$ and $r/\theta(2^\theta, 2^\theta) = 2^\theta$. \diamond

2.4 The CSP and Omitting Types

In the 1980s Hobby and McKenzie developed tame congruence theory that studies the local structure of algebras [50]. They discovered that the local structure of universal algebras is surprisingly well behaved and can be classified into just five types. Each type is associated with a certain basic algebra, and if an algebra admits a type, it means that its local structure resembles that of the corresponding basic algebra. The five basic algebras and corresponding types are:

1. A *unary* algebra whose basic operations are permutations (*unary type*);
2. A one-dimensional vector space over some finite field (*affine type*);
3. A 2-element boolean algebra whose basic operations include conjunction, disjunction, and negation (*boolean type*);
4. A 2-element lattice whose basic operations include conjunction and disjunction (*lattice type*);
5. A 2-element semilattice whose basic operations include a semilattice operation (*semilattice type*).

Omitting or admitting types is strongly related to the complexity of the CSP. Theorem 5 from [25] claims that if a relational structure \mathcal{A} is such that $\text{Alg}(\mathcal{A})$ is idempotent and admits the unary type then $\text{CSP}(\mathcal{A})$ is NP-complete. Combined with Theorem 12 this allows for a more precise Dichotomy Conjecture.

Conjecture 15. If a relational structure \mathcal{A} is such that $\text{Alg}(\mathcal{A})$ is idempotent, then $\text{CSP}(\mathcal{A})$ is solvable in polynomial time if and only if no subalgebra of $\text{Alg}(\mathcal{A})$ admits the unary type. Otherwise it is NP-complete.

Or in the stronger algebraic version.

Conjecture 16 (Dichotomy Conjecture ++). An idempotent algebra \mathbb{A} is globally tractable if and only if none of its subalgebras admits the unary type. Otherwise it is NP-complete.

The results [63] imply that the latter condition in Conjecture 16 is also equivalent to the existence of a weak near-unanimity term operation in \mathbb{A} .

Conjecture 16 has been confirmed in a number of special cases.

- Schaefer’s classification of 2-element structures [68] with respect to complexity can be easily extended to 2-element algebras. Then it claims that an idempotent 2-element algebra is globally tractable if and only if it has one of the following term operations: a semilattice operation, a majority operation, or the affine operation $x - y + z$. By [65] this is equivalent to having a term weak near-unanimity operation.
- Let H be a graph, $\mathbb{A} = \text{Alg}(H)$, and let \mathbb{B} the idempotent algebra constructed from \mathbb{A} as in Theorem 13. If H is bipartite then \mathbb{B} is 2-element and has a majority term operation. Otherwise \mathbb{B} admits the unary type [15]. Thus the classification from [49] matches the Dichotomy Conjecture++.
- The Dichotomy Conjecture++ was confirmed for 3-element algebras in [12, 16], and for 4-element algebras in [61].
- It was shown in [13, 17] that the Dichotomy Conjecture++ holds for *conservative* algebras, that is, algebras in which every subset of the universe is a subalgebra. These results have also been simplified in [1, 19].
- Finally, Zhuk in [70, 71] proved the conjecture for 5- and 7-element algebras.

In the rest of this paper we show that Conjecture 16 is true. The hardness part of the conjecture follows from the mentioned result of [25]; so we focus on the algorithmic part. The algorithm presented here is based on [23] (a full version can be found in [22]). Note that the conjecture was also independently proved by Zhuk [72].

3 CSP Algorithms

It would be natural to expect a wide variety of algorithms solving the CSP in those cases in which it can be solved in polynomial time. However, surprisingly, only two types of such algorithms are known, and for each type there is ‘the most general’ algorithm, which means that basically only two CSP algorithms exist.

3.1 Local Propagation Algorithms

The first type can be described as local propagation algorithms. We describe one such algorithm, applicable whenever any other propagation algorithm solves the problem.

Let $R \subseteq A^n$ be a relation, $\mathbf{a} \in A^n$, and $J = \{i_1, \dots, i_k\} \subseteq [n]$. Let $\text{pr}_J \mathbf{a} = (\mathbf{a}[i_1], \dots, \mathbf{a}[i_k])$ and $\text{pr}_J R = \{\text{pr}_J \mathbf{a} : \mathbf{a} \in R\}$. Often we will use sets of CSP variables to index entries of tuples and relations. Projections in such cases are defined in a similar way. Let $\mathcal{I} = (V, \mathcal{C})$ be a CSP instance. For $W \subseteq V$ by \mathcal{I}_W we denote the *restriction* of \mathcal{I} onto W , that is, the instance (W, \mathcal{C}_W) , where for each $C = \langle \mathbf{s}, R \rangle \in \mathcal{C}$, the set \mathcal{C}_W includes the constraint $C_W = \langle \mathbf{s} \cap W, \text{pr}_{\mathbf{s} \cap W} R \rangle$. The set of solutions of \mathcal{I}_W will be denoted by \mathcal{S}_W .

Unary solutions, that is, when $|W| = 1$ play a special role. As is easily seen, for $v \in V$ the set \mathcal{S}_v is just the intersections of unary projections $\text{pr}_v R$ of constraints whose scope contains v . Instance \mathcal{I} is said to be *1-minimal* if for every $v \in V$ and every constraint $C = \langle \mathbf{s}, R \rangle \in \mathcal{C}$ such that $v \in \mathbf{s}$, it holds $\text{pr}_v R = \mathcal{S}_v$. For a 1-minimal instance one may always assume that allowed values for a variable $v \in V$ is the set \mathcal{S}_v . We call this set the *domain* of v and assume that CSP instances may have different domains, which nevertheless are always subalgebras or quotient algebras of the original algebra \mathbb{A} . It will be convenient to denote the domain of v by \mathbb{A}_v . The domain \mathbb{A}_v may change as a result of transformations of the instance.

Instance \mathcal{I} is said to be *(2,3)-minimal* if it satisfies the following condition: – for every $X = \{u, v\} \subseteq V$, any $w \in V - X$, and any $(a, b) \in \mathcal{S}_X$, there is $c \in \mathbb{A}_w$ such that $(a, c) \in \mathcal{S}_{\{u, w\}}$ and $(b, c) \in \mathcal{S}_{\{v, w\}}$.

For $k \in \mathbb{N}$, $(k, k + 1)$ -minimality is defined in a similar way using $k, k + 1$.

Instance \mathcal{I} is said to be *minimal* (or *globally minimal*) if for every $C = \langle \mathbf{s}, R \rangle \in \mathcal{C}$ and every $\mathbf{a} \in R$ there is a solution φ such that $\varphi(\mathbf{s}) = \mathbf{a}$. Similarly, \mathcal{I} is said to be *globally 1-minimal* if for every $v \in V$ and $a \in \mathbb{A}_v$ there is a solution φ with $\varphi(v) = a$.

Any instance can be transformed to a 1-minimal or (2,3)-minimal instance in polynomial time using the standard constraint propagation algorithms (see, e.g. [34]). These algorithms work by changing the constraint relations and the domains of the variables eliminating some tuples and elements from them. We call such a process *tightening* the instance. It is important to notice that if the original instance belongs to $\text{CSP}(\mathbb{A})$ for some algebra \mathbb{A} , that is, all its constraint relations are invariant under the basic operations of \mathbb{A} , the constraint relations

obtained by propagation algorithms are also invariant under the basic operations of \mathbb{A} , and so the resulting instance also belongs to $\text{CSP}(\mathbb{A})$. Establishing minimality amounts to solving the problem and so not always can be easily done.

If a constraint propagation algorithm solves a CSP, the problem is said to be of bounded width. More precisely, $\text{CSP}(\Gamma)$ (or $\text{CSP}(\mathbb{A})$) is said to have *bounded width* if for some k every $(k, k+1)$ -minimal instance from $\text{CSP}(\Gamma)$ (or $\text{CSP}(\mathbb{A})$) has a solution (we also say that $\text{CSP}(\Gamma)$ has width k in this case). Problems of bounded width are well studied, see the older survey [26] and more recent [2].

Theorem 17 ([2, 14, 21, 60]). *For an idempotent algebra \mathbb{A} the following are equivalent:*

- (1) $\text{CSP}(\mathbb{A})$ has bounded width;
- (2) every $(2,3)$ -minimal instance from $\text{CSP}(\mathbb{A})$ has a solution;
- (3) \mathbb{A} has a weak near-unanimity term of arity k for every $k \geq 3$;
- (4) every quotient algebra of a subalgebra of \mathbb{A} has a nontrivial operation, and none of them is equivalent to a module (in a certain precise sense).

Example 18. (1) The 2-SAT problem has bounded width, namely, width 2.

- (2) The H -COLOURING problem has width 2 when graph H is bipartite, and NP-complete otherwise.
- (3) The HORN-SAT is the SATISFIABILITY problem restricted to Horn clauses, i.e., clauses of the form $x_1 \wedge \dots \wedge x_k \rightarrow y$. Let $\Gamma_{k\text{-HORN}}$ be the constraint language consisting of relations expressible by a Horn clause with at most k premises. The problem k -HORN-SAT is equivalent to $\text{CSP}(\Gamma_{k\text{-HORN}})$ and has width k .

3.2 Gaussian Elimination and Few Subpowers

The simplest algorithm of the second type is known from basic linear algebra—Gaussian elimination. While propagation algorithms cannot solve the LIN problem, it is solvable by Gaussian elimination. A similar algorithm solving group constraints, defined in terms of finite groups, was suggested in [37].

Algebraic techniques make it possible to generalize the Gaussian elimination algorithm. The algorithm from [24] solving $\text{CSP}(\mathcal{A})$ for a relational structure \mathcal{A} with a Mal'tsev polymorphism can be viewed as a generalization of Gaussian elimination in the following sense. Similar to the output of Gaussian elimination it constructs some sort of a basis or a compact representation of the set of all solutions of a CSP.

It is thought that the property of relations to have a compact representation, where compactness is understood as having size polynomial in the arity of the relation, is the right generalization of linear algebra problems where Gaussian elimination can be used. Let $\mathbb{A} = (A, F)$ be an algebra. It is said to be an *algebra with few subpowers* if every relation over A invariant under F admits a compact representation [9, 51]. The term ‘few subpowers’ comes from the observation that every relation invariant under F is a subalgebra of a direct power of \mathbb{A} , and if

the size of compact representation is bounded by a polynomial $p(n)$ then at most $2^{p(n)}$ n -ary relations can be represented, while the total number of such relations can be as large as $2^{|A|^n}$. Algebras with few subpowers are completely characterized by Idziak et al. [9, 51]. A minor generalization of the algorithm from [32] solves $\text{CSP}(\mathbb{A})$, where \mathbb{A} has few subpowers.

Here the few subpowers algorithm is used in the context of semilattice edges. A pair of elements $a, b \in \mathbb{A}$ is said to be a *semilattice edge* if there is a binary term operation f of \mathbb{A} such that $f(a, a) = a$ and $f(a, b) = f(b, a) = f(b, b) = b$, that is, f is a semilattice operation on $\{a, b\}$. For example, the set $\{0, 2\}$ from Example 14 is a semilattice edge, and the operation r of \mathbb{A}_M witnesses that.

Proposition 19 ([21]). *If an idempotent algebra \mathbb{A} has no semilattice edges, it has few subpowers, and therefore $\text{CSP}(\mathbb{A})$ is solvable in polynomial time.*

Semilattice edges have other useful properties including the following one that we use for reducing a CSP to smaller problems.

Lemma 20 ([20]). *For any idempotent algebra \mathbb{A} there is a term operation xy (think multiplication) such that xy is a semilattice operation on any semilattice edge and for any $a, b \in \mathbb{A}$ either $ab = a$ or $\{a, ab\}$ is a semilattice edge.*

Note that any semilattice operation satisfies the conditions of Lemma 20. The operation r of the algebra \mathbb{A}_M from Example 14 is not a semilattice operation (it is not commutative), but it satisfies the conditions of Lemma 20.

4 Congruence Separation and Centralizers

We now move on to describe the algorithm resolving the Dichotomy Conjecture. In this section we introduce two of the key ingredients of our algorithm.

4.1 Separating Congruences

Unlike the vast majority of the literature on the algebraic approach to the CSP we use not only term operations, but also polynomial operations of an algebra. It should be noted however that the first to use polynomials for CSP algorithms was Maroti in [62]. We make use of some ideas from that paper in the next section. Let $f(x_1, \dots, x_k, y_1, \dots, y_\ell)$ be a $k + \ell$ -ary term operation of an algebra \mathbb{A} and $b_1, \dots, b_\ell \in \mathbb{A}$. The operation $g(x_1, \dots, x_k) = f(x_1, \dots, x_k, b_1, \dots, b_\ell)$ is called a *polynomial* of \mathbb{A} . A polynomial for which $k = 1$ is said to be a *unary* polynomial. If α is a congruence, and f is a unary polynomial, by $f(\alpha)$ we denote the set of pairs $\{(f(a), f(b)) \mid (a, b) \in \alpha\}$.

Let \mathbb{A} be an algebra and let $\text{Con}(\mathbb{A})$ denote its congruence lattice. For $\alpha, \beta \in \text{Con}(\mathbb{A})$ we write $\alpha \prec \beta$ if $\alpha < \beta$ (that is, $\alpha \subset \beta$ as sets of pairs) and $\alpha \leq \gamma \leq \beta$ in $\text{Con}(\mathbb{A})$ implies $\gamma = \alpha$ or $\gamma = \beta$. If this is the case we call (α, β) a *prime interval* in $\text{Con}(\mathbb{A})$. Let $\alpha \prec \beta$ and $\gamma \prec \delta$ be prime intervals in $\text{Con}(\mathbb{A})$. We say that $\alpha \prec \beta$ can be *separated* from $\gamma \prec \delta$ if there is a unary polynomial f of \mathbb{A} such that $f(\beta) \not\subseteq \alpha$, but $f(\delta) \subseteq \gamma$. The polynomial f in this case is said to *separate* $\alpha \prec \beta$ from $\gamma \prec \delta$.

Example 21. The unary polynomials of the algebra \mathbb{A}_M from Example 14 include the following unary operations (these are the polynomials we will use, there are more unary polynomials of \mathbb{A}_M):

$$\begin{aligned} h_1(x) &= r(x, 0) = r(x, 1), \text{ such that } h_1(0) = h_1(2) = 0, h_1(1) = 1; \\ h_2(x) &= r(2, x), \text{ such that } h_2(0) = h_2(1) = 0, h_2(2) = 2; \\ h_3(x) &= r(0, x) = 0. \end{aligned}$$

The lattice $\text{Con}(\mathbb{A}_M)$ has two prime intervals $\underline{0} \prec \theta$ and $\theta \prec \underline{1}$ (see Example 14 and Fig. 2(b)). As is easily seen, $h_3(\underline{1}) \subseteq \underline{0}$, therefore h_3 collapses both prime intervals. Since $h_1(\theta) \not\subseteq \underline{0}$, but $h_1(\underline{1}) \subseteq \theta$, polynomial h_1 separates $(\underline{0}, \theta)$ from $(\theta, \underline{1})$. Similarly, the polynomial h_2 separates $(\theta, \underline{1})$ from $(\underline{0}, \theta)$, because $h_2(\underline{1}) \not\subseteq \theta$, while $h_2(\theta) \subseteq \underline{0}$. \diamond

In a similar way separation can be defined for prime intervals in different coordinate positions of a relation. Let R be a subdirect product of $\mathbb{A}_1 \times \cdots \times \mathbb{A}_n$, that is, $\text{pr}_i R = \mathbb{A}_i$ for $i \in [n]$. Then R can also be viewed as an algebra with operations acting component-wise, and polynomials of R can be defined in the same way. Since every basic operation acts on R component-wise, its unary polynomials also act component-wise. Therefore, for a unary polynomial f of R it makes sense to consider $f(a)$, where $a \in \mathbb{A}_i$, $i \in [n]$. Let $i, j \in [n]$ and let $\alpha \prec \beta$, $\gamma \prec \delta$ be prime intervals in $\text{Con}(\mathbb{A}_i)$ and $\text{Con}(\mathbb{A}_j)$, respectively. Interval $\alpha \prec \beta$ can be separated from $\gamma \prec \delta$ if there is a unary polynomial f of R such that $f(\beta) \not\subseteq \alpha$ but $f(\delta) \subseteq \gamma$. The binary relation ‘cannot be separated’ on the set of prime intervals of an algebra or factors of a relation is easily seen to be reflexive and transitive. We will say that $\alpha \prec \beta$, $\gamma \prec \delta$ cannot be separated if $\alpha \prec \beta$ and $\gamma \prec \delta$ cannot be separated from each other.

Example 22. Let R be a ternary relation over \mathbb{A}_M invariant under r, t , given by

$$R = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 2 \end{pmatrix},$$

where triples, the elements of the relation are written vertically. It will be convenient to distinguish congruences in the three factors of R , so we denote them by $\underline{0}_i, \theta_i, \underline{1}_i$ for the i th factor. Since $\text{pr}_{12} R$ is the congruence θ , any unary polynomial h of R acts identically modulo θ on the first and the second coordinate positions. In particular, the prime interval $(\theta_1, \underline{1}_1)$ cannot be separated from the prime interval $(\theta_2, \underline{1}_2)$. Consider the polynomial $h(x)$ of R given by

$$h(x) = r \left(\begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}, x \right) = \begin{pmatrix} r(2, x) \\ r(2, x) \\ r(0, x) \end{pmatrix} = \begin{pmatrix} h_2(x) \\ h_2(x) \\ h_3(x) \end{pmatrix},$$

it is a polynomial of R because $(2, 2, 0) \in R$. Since $h_2(\underline{1}) \not\subseteq \theta$, but $h_3(\underline{1}) \subseteq \theta$ and $h_3(\theta) \subseteq \underline{0}$, the prime interval $(\theta_2, \underline{1}_2)$ can be separated from $(\underline{0}_3, \theta_3)$ and $(\theta_3, \underline{1}_3)$. Similarly, the interval $(\theta_3, \underline{1}_3)$ can be separated from $(\underline{0}_1, \theta_1), (\underline{0}_2, \theta_2)$.

Through a slightly more involved argument it can be shown that $(\theta_3, \underline{1}_3)$ cannot be separated from $(\theta_1, \underline{1}_1)$, $(\theta_2, \underline{1}_2)$. In the next section we explain why the prime intervals $(\underline{0}_i, \theta_i)$, $(\underline{0}_j, \theta_j)$ cannot be separated from each other. \diamond

4.2 Quasi-Centralizers

The second ingredient we will use here is the notion of quasi-centralizer of a pair of congruences. It is similar to the centralizer as it is defined in commutator theory [41], albeit the exact relationship between the two concepts is not quite clear, and so we name it differently for safety.

For an algebra \mathbb{A} , a term operation $f(x, y_1, \dots, y_k)$, and $\mathbf{a} \in \mathbb{A}^k$, let $f^\mathbf{a}(x) = f(x, \mathbf{a})$; it is a unary polynomial of \mathbb{A} . Let $\alpha, \beta \in \text{Con}(\mathbb{A})$, and let $\zeta(\alpha, \beta) \subseteq \mathbb{A}^2$ denote the following binary relation: $(a, b) \in \zeta(\alpha, \beta)$ if and only if, for any term operation $f(x, y_1, \dots, y_k)$, any $i \in [k]$, and any $\mathbf{a}, \mathbf{b} \in \mathbb{A}^k$ such that $\mathbf{a}[i] = a$, $\mathbf{b}[i] = b$, and $\mathbf{a}[j] = \mathbf{b}[j]$ for $j \neq i$, it holds $f^\mathbf{a}(\beta) \subseteq \alpha$ if and only if $f^\mathbf{b}(\beta) \subseteq \alpha$. (Polynomials of the form $f^\mathbf{a}, f^\mathbf{b}$ are sometimes called *twin polynomials*.) The relation $\zeta(\alpha, \beta)$ is always a congruence of \mathbb{A} . Next we show how it is related to the structure of algebra \mathbb{A} and the corresponding CSP.

Example 23. In the algebra \mathbb{A}_M , see Example 14, the quasi-centralizer acts as follows: $\zeta(\underline{0}, \theta) = \underline{1}$ and $\zeta(\theta, \underline{1}) = \theta$. We start with the second centralizer. Since every polynomial preserves congruences, for any term operation $h(x, y_1, \dots, y_k)$ and any $\mathbf{a}, \mathbf{b} \in \mathbb{A}_M^k$ such that $(\mathbf{a}[i], \mathbf{b}[i]) \in \theta$ for $i \in [k]$, we have $(h^\mathbf{a}(x), h^\mathbf{b}(x)) \in \theta$ for any x . This of course implies $\zeta(\theta, \underline{1}) \geq \theta$. On the other hand, let $f(x, y) = r(y, x)$. Then as we saw before, $f^0(x) = f(x, 0) = r(0, x) = h_3(x)$ and $f^2(x) = f(x, 2) = r(2, x) = h_2(x)$, and $f^0(\underline{1}) \subseteq \theta$, while $f^2(\underline{1}) \not\subseteq \theta$. This means that $(0, 2) \notin \zeta(\theta, \underline{1})$ and so $\zeta(\theta, \underline{1}) \subset \underline{1}$. For the first centralizer it suffices to demonstrate that the condition in the definition of quasi-centralizer is satisfied for pairs of twin polynomials produced by r, t of the form $(r(a, x), r(b, x))$, $(r(x, a), r(x, b))$, $(t(x, a_1, a_2), t(x, b_1, b_2))$, $(t(a_1, x, a_2), t(b_1, x, b_2))$, $(t(a_1, a_2, x), t(b_1, b_2, x))$, which can be verified directly.

Note that the equality $\zeta(\underline{0}, \theta) = \underline{1}$ explains why prime intervals $(\underline{0}_i, \theta_i)$, $(\underline{0}_j, \theta_j)$ in Example 22 cannot be separated. For that the relation $\text{pr}_{ij}R$ has to contain tuples $(a, b), (c, d)$ such that $(a, c) \in \zeta(\underline{0}_i, \theta_i)$ while $(b, d) \notin \zeta(\underline{0}_j, \theta_j)$, which is impossible. \diamond

5 The Algorithm

In this section we introduce the reductions used in the algorithm, and then explain the algorithm itself.

5.1 Decomposition of CSPs

Let R be a binary relation, a subdirect product of $\mathbb{A} \times \mathbb{B}$, and $\alpha \in \text{Con}(\mathbb{A})$, $\gamma \in \text{Con}(\mathbb{B})$. Relation R is said to be $\alpha\gamma$ -aligned if, for any $(a, c), (b, d) \in R$,

$(a, b) \in \alpha$ if and only if $(c, d) \in \gamma$. This means that if A_1, \dots, A_k are the α -blocks of \mathbb{A} , then there are also k γ -blocks of \mathbb{B} and they can be labeled B_1, \dots, B_k in such a way that

$$R = (R \cap (A_1 \times B_1)) \cup \dots \cup (R \cap (A_k \times B_k)).$$

Lemma 24. *Let $R, \mathbb{A}, \mathbb{B}$ be as above and $\alpha, \beta \in \text{Con}(\mathbb{A})$, $\gamma, \delta \in \text{Con}(\mathbb{B})$, with $\alpha \prec \beta$, $\gamma \prec \delta$. If (α, β) and (γ, δ) cannot be separated, then R is $\zeta(\alpha, \beta)\zeta(\gamma, \delta)$ -aligned.*

Lemma 24 provides a way to decompose CSP instances. Let $\mathcal{I} = (V, \mathcal{C})$ be a (2,3)-minimal instance from $\text{CSP}(\mathbb{A})$. We will always assume that a (2,3)-minimal instance has a constraint $C^X = \langle X, R^X \rangle$ for every $X \subseteq V$, $|X| = 2$, where $R^X = \mathcal{S}_X$. Recall that \mathbb{A}_v denotes the domain of $v \in V$. Also, let $W \subseteq V$ and congruences $\alpha_v, \beta_v \in \text{Con}(\mathbb{A}_v)$ for $v \in W$ be such that $\alpha_v \prec \beta_v$, and for any $v, w \in W$ the intervals (α_v, β_v) and (α_w, β_w) cannot be separated in $R^{\{v, w\}}$.

Denoting $\zeta_v = \zeta(\alpha_v, \beta_v)$ for $v \in W$ we see that there is a one-to-one correspondence between ζ_v - and ζ_w -blocks of \mathbb{A}_v and \mathbb{A}_w , $v, w \in W$. Moreover, by (2,3)-minimality these correspondences are consistent, that is, if $u, v, w \in W$ and B_u, B_v, B_w are ζ_u -, ζ_v - and ζ_w -blocks, respectively, such that $R^{\{u, v\}} \cap (B_u \times B_v) \neq \emptyset$ and $R^{\{v, w\}} \cap (B_v \times B_w) \neq \emptyset$, then $R^{\{u, w\}} \cap (B_u \times B_w) \neq \emptyset$. This means that \mathcal{I}_W can be split into several instances, whose domains are ζ_v -blocks.

Lemma 25. *Let $\mathcal{I}, W, \alpha_v, \beta_v$ for each $v \in W$, be as above. Then \mathcal{I}_W can be decomposed into a collection of instances $\mathcal{I}_1, \dots, \mathcal{I}_k$, k constant, $\mathcal{I}_i = (W, \mathcal{C}_i)$ such that every solution of \mathcal{I}_W is a solution of one of the \mathcal{I}_i and for every $v \in W$ its domain in \mathcal{I}_i is a ζ_v -block.*

Example 26. Consider the following simple CSP instance from $\text{CSP}(\mathbb{A}_M)$, where \mathbb{A}_M is the algebra introduced in Example 14, and R is the relation introduced in Example 22: $\mathcal{I} = (V = \{v_1, v_2, v_3, v_4, v_5\}, \{C^1 = \langle s_1 = (v_1, v_2, v_3), R_1 \rangle, C^2 = \langle s_2 = (v_2, v_4, v_5), R_2 \rangle\})$, where $R_1 = R_2 = R$. To make the instance (2,3)-minimal we run the appropriate local propagation algorithm on it. First, such an algorithm adds new binary constraints $C^{\{v_i, v_j\}} = \langle (v_i, v_j), R^{\{v_i, v_j\}} \rangle$ for $i, j \in [5]$ starting with $R^{\{v_i, v_j\}} = \mathbb{A}_M \times \mathbb{A}_M$. It then iteratively removes pairs from these relations that do not satisfy the (2,3)-minimality condition. Similarly, it tightens the original constraint relations if they violate the conditions of (2,3)-minimality. This algorithm does not change constraints C^1, C^2 , and the new binary relations are as follows: $R^{\{v_1, v_2\}} = R^{\{v_2, v_4\}} = R^{\{v_1, v_4\}} = \theta$, $R^{\{v_1, v_3\}} = R^{\{v_2, v_3\}} = R^{\{v_2, v_5\}} = R^{\{v_4, v_5\}} = R^{\{v_1, v_5\}} = R^{\{v_3, v_4\}} = Q$, and $R^{\{v_3, v_5\}} = S$, where

$$Q = \text{pr}_{13}R = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 & 0 & 2 \end{pmatrix}, \quad S = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 2 & 2 \\ 0 & 1 & 0 & 1 & 2 & 0 & 2 \end{pmatrix}.$$

For convenience let the domain of v_i be denoted by \mathbb{A}_i , its elements by $0_i, 1_i, 2_i$, and the congruences of \mathbb{A}_i by $0_i, \theta_i, 1_i$.

Let $W = \{v_1, v_2, v_4\}$, $\alpha_i = \theta_i$, $\beta_i = \underline{1}_i$ for $v_i \in W$. We have $\zeta_i = \zeta(\alpha_i, \beta_i) = \theta_i = \alpha_i$. Then, as was observed in Example 23, the prime interval (α_i, β_i) cannot be separated from (α_j, β_j) for $v_i, v_j \in W$. Therefore by Lemma 25 the instance $\mathcal{I}_W = (\{v_1, v_2, v_4\}, \{C_W^1 = \langle(v_1, v_2), \text{pr}_{v_1 v_2} R_1\rangle, C_W^2 = \langle(v_2, v_4), \text{pr}_{v_2 v_4} R_2\rangle\})$ can be decomposed into a disjoint union of two instances:

$$\begin{aligned}\mathcal{I}_1 &= (\{v_1, v_2, v_4\}, \{\langle(v_1, v_2), Q_1\rangle, \langle(v_2, v_4), Q_2\rangle\}), \\ \mathcal{I}_2 &= (\{v_1, v_2, v_4\}, \{\langle(v_1, v_2), \{(2_1, 2_2)\}\rangle, \langle(v_2, v_4), \{(2_2, 2_4)\}\rangle\}),\end{aligned}$$

where $Q_1 = \{0_1, 1_1\} \times \{0_2, 1_2\}$, $Q_2 = \{0_2, 1_2\} \times \{0_4, 1_4\}$. \diamond

5.2 Block-Minimality

In order to formulate the algorithm properly we need one more transformation of algebras. An algebra \mathbb{A} is said to be *subdirectly irreducible* if the intersection of all its nontrivial (different from the equality relation) congruences is nontrivial. This smallest nontrivial congruence $\mu_{\mathbb{A}}$ is called the *monolith* of \mathbb{A} . For instance, the algebra \mathbb{A}_M from Example 14 is subdirectly irreducible, because it has the smallest nontrivial congruence, θ . It is a folklore observation that any CSP instance can be transformed in polynomial time to an instance, in which the domain of every variable is a subdirectly irreducible algebra. We will assume this property of all the instances we consider.

Lemma 25 allows us to use a new type of consistency of a CSP instance, block-minimality, which is key for our algorithm. In a certain sense it is similar to the standard local consistency, as it is also defined through a family of relations that have to be consistent in a certain way. However, block-minimality is not quite local, and is more difficult to establish, as it involves solving smaller CSP instances recursively. The definitions below are designed to allow for an efficient procedure to establish block-minimality. This is achieved either by allowing for decomposing a subinstance into instances over smaller domains as in Lemma 25, or by replacing large domains with their quotient algebras.

Let $\mathcal{I} = (V, \mathcal{C}) \in \text{CSP}(\mathbb{A})$ and α_v be a congruence of \mathbb{A}_v for $v \in V$. By $\mathcal{I}/\bar{\alpha}$ we denote the instance $(V, \mathcal{C}_{\bar{\alpha}})$ constructed as follows: the domain of $v \in V$ is \mathbb{A}_v/α_v ; for every constraint $C = \langle \mathbf{s}, R \rangle \in \mathcal{C}$, $\mathbf{s} = (v_1, \dots, v_k)$, the set $\mathcal{C}_{\bar{\alpha}}$ includes the constraint $\langle \mathbf{s}, R/\bar{\alpha} \rangle$, where $R/\bar{\alpha} = \{(\mathbf{a}[v_1]^{\alpha_{v_1}}, \dots, \mathbf{a}[v_k]^{\alpha_{v_k}}) \mid \mathbf{a} \in R\}$.

We start with several definitions. Let $\mathcal{I} = (V, \mathcal{C})$ be a (2,3)-minimal instance and let $\{R^X \mid X \subseteq V, |X| = 2\}$ be the relations introduced after Lemma 24. Let $\mathcal{U}^{\mathcal{I}}$ denote the set of triples (v, α, β) such that $v \in V$, $\alpha, \beta \in \text{Con}(\mathbb{A}_v)$, and $\alpha \prec \beta$. For every $(v, \alpha, \beta) \in \mathcal{U}^{\mathcal{I}}$, let $W_{v, \alpha\beta}$ denote the set of all variables $w \in V$ such that (α, β) and (γ, δ) cannot be separated in $R^{\{v, w\}}$ for some $\gamma, \delta \in \text{Con}(\mathbb{A}_w)$ with $(w, \gamma, \delta) \in \mathcal{U}^{\mathcal{I}}$. Sets of the form $W_{v, \alpha\beta}$ are called *coherent sets*. Let $\mathcal{Z}^{\mathcal{I}}$ denote the set of triples $(v, \alpha, \beta) \in \mathcal{U}^{\mathcal{I}}$, for which $\zeta(\alpha, \beta)$ is the full relation.

We say that algebra \mathbb{A}_v is *semilattice free* if it does not contain semilattice edges. Let $\text{size}(\mathcal{I})$ denote the maximal size of domains of \mathcal{I} that are not semilattice free and $\text{MAX}(\mathcal{I})$ be the set of variables $v \in V$ with $|\mathbb{A}_v| = \text{size}(\mathcal{I})$ and \mathbb{A}_v

is not semilattice free. For instances $\mathcal{I}, \mathcal{I}'$ we say that \mathcal{I}' is *strictly smaller* than \mathcal{I} if $\text{size}(\mathcal{I}') < \text{size}(\mathcal{I})$. For $Y \subseteq V$ let $\mu_v^Y = \mu_v$ if $v \in Y$ and $\mu_v^Y = \underline{0}_v$ otherwise.

Instance \mathcal{I} is said to be *block-minimal* if for every $(v, \alpha, \beta) \in \mathcal{U}^\mathcal{I}$ the following conditions hold:

- (B1) if $(v, \alpha, \beta) \notin \mathcal{Z}^\mathcal{I}$, the problem $\mathcal{I}_{W_{v,\alpha\beta}}$ is minimal;
- (B2) if $(v, \alpha, \beta) \in \mathcal{Z}^\mathcal{I}$, for every $C = \langle \mathbf{s}, R \rangle \in \mathcal{C}$ the problem $\mathcal{I}_{W_{v,\alpha\beta}/\bar{\mu}^Y}$, where $Y = \text{MAX}(\mathcal{I}) - \mathbf{s}$, is minimal;
- (B3) if $(v, \alpha, \beta) \in \mathcal{Z}^\mathcal{I}$, then for every $(w, \gamma, \delta) \in \mathcal{U}^\mathcal{I} - \mathcal{Z}^\mathcal{I}$ the problem $\mathcal{I}_{W_{v,\alpha\beta}/\bar{\mu}^Y}$, where $Y = \text{MAX}(\mathcal{I}) - (W_{v,\alpha\beta} \cap W_{w,\gamma\delta})$ is minimal.

Example 27. Let us consider again the instance \mathcal{I} from Example 26. There we found all its binary solutions, and now we use them to find coherent sets and to verify that this instance is block-minimal. For the instance \mathcal{I} we have $\mathcal{U}^\mathcal{I} = \{(v_i, \underline{0}_i, \theta_i), (v_i, \theta_i, \underline{1}_i) \mid i \in [5]\}$ and $\mathcal{Z}^\mathcal{I} = \{(v_i, \underline{0}_i, \theta_i) \mid i \in [5]\}$. As we noticed in Example 22, interval $(\underline{0}_i, \theta_i)$ cannot be separated from $(\underline{0}_j, \theta_j)$ for any $i, j \in [5]$. Therefore, for each $i \in [5]$ we have $W_{v_i, \underline{0}_i, \theta_i} = V$. Also, it was shown in Example 22 that $(\theta_i, \underline{1}_i)$ cannot be separated from $(\theta_j, \underline{1}_j)$ for $\{i, j\} = \{1, 2\}$ and $\{i, j\} = \{2, 4\}$, while $\{\theta_i, \underline{1}_i\}$ can be separated from $(\theta_j, \underline{1}_j)$ and $(\underline{0}_j, \theta_j)$ for $i \in \{1, 2, 4\}$ and $j \in \{3, 5\}$. Therefore, for $i \in \{1, 2, 4\}$ we have $W_{v_i, \theta_i, \underline{1}_i} = \{v_1, v_2, v_4\}$. Finally, $(\theta_3, \underline{1}_3)$ can be separated from $(\underline{0}_5, \theta_5), (\theta_5, \underline{1}_5)$ by considering the relation S from Example 26, and $(\underline{0}_i, \theta_i)$, $i \in \{1, 2, 4\}$ can be separated from $(\theta_3, \underline{1}_3)$ by considering the relation Q . Therefore, $W_{v_i, \theta_i, \underline{1}_i} = \{v_i\}$ for $i \in \{3, 5\}$.

Now we check the conditions (B1)–(B3) for \mathcal{I} . Since $\zeta(\theta_i, \underline{1}_i) = \theta_i$, $i \in [5]$, for the coherent sets $W_{v_i, \theta_i, \underline{1}_i}$ we need to check condition (B1). If $i = 3, 5$ this condition is trivially true, as the set of solutions of \mathcal{I} on every 1-element set of variables is \mathbb{A}_M . Consider $W_{v_1, \theta_1, \underline{1}_1} = \{v_1, v_2, v_4\}$; as is easily seen, a triple (a_1, a_2, a_4) is a solution of $\mathcal{I}_{\{v_1, v_2, v_4\}}$ if and only if $(a_1, a_2), (a_1, a_4), (a_2, a_4) \in \theta$. Condition (B1) amounts to saying that for any constraint of \mathcal{I} , say, C^1 , and any tuple \mathbf{a} from its constraint relation R_1 , the projection $\text{pr}_{v_1 v_2} \mathbf{a}$ can be extended to a solution of $\mathcal{I}_{\{v_1, v_2, v_4\}}$. Since $\text{pr}_{v_1 v_2} \mathbf{a} \in \theta$, this can always be done. For other constraints (B1) is verified in a similar way.

Next consider $W_{v_1, \underline{0}_1, \theta_1} = V$. As $\zeta(\underline{0}_1, \theta_1) = \underline{1}_1$, we have to verify conditions (B2), (B3). We consider condition (B2) for constraint C^1 , the remaining cases are similar. The monolith of \mathbb{A}_M is θ , therefore in the first case $Y = \{v_4, v_5\}$ and $\mu_{v_i}^Y$ is the equality relation for $i \in \{1, 2, 3\}$ and $\mu_{v_4}^Y = \theta_4, \mu_{v_5}^Y = \theta_5$. The instance $\mathcal{I}/\bar{\mu}^Y$ is as follows: $\mathcal{I}/\bar{\mu}^Y = (V, \{C'^1 = \langle \mathbf{s}_1, R_1 \rangle, C'^2 = \langle \mathbf{s}_2, R_2/\bar{\mu} \rangle\})$. The constraint relation, of C'^1 equals R_1 , as $\mu_{v_i}^Y = \underline{0}_i$ for $i \in \{1, 2, 3\}$. The constraint relation of C'^2 then equals $R'_2 = R_2/\bar{\mu}^Y = \{(0, 0^\theta, 0^\theta), (1, 0^\theta, 0^\theta), (2, 2^\theta, 0^\theta), (2, 2^\theta, 2^\theta)\}$. Now, for every tuple $\mathbf{a} \in R_1$, and for every tuple $\mathbf{b} \in R'_2$ we need to find solutions φ, ψ of $\mathcal{I}/\bar{\mu}^Y$ such that $\varphi(v_i) = \mathbf{a}[v_i]$ for $i \in \{1, 2, 3\}$ and $\psi(v_i) = \mathbf{b}[v_i]$ for $i \in \{2, 4, 5\}$. If $\mathbf{a}[v_2] \in \{0, 1\}$ ($\mathbf{b}[v_2] \in \{0, 1\}$) then extending \mathbf{a} by $\varphi(v_4) = \varphi(v_5) = 0^\theta$ (extending \mathbf{b} by $\psi(v_1) = \psi(v_3) = 0$) gives solutions of $\mathcal{I}/\bar{\mu}^Y$. If $\mathbf{a}[v_2] = 2$ ($\mathbf{b}[v_2] = 2$), then tuples \mathbf{a}, \mathbf{b} can be extended by $\varphi(v_4) = \varphi(v_5) = 2^\theta$ and by $\psi(v_1) = \psi(v_3) = 2$ to solutions of $\mathcal{I}/\bar{\mu}^Y$. \diamond

Next we observe that establishing block-minimality can be efficiently reduced to solving a polynomial number of strictly smaller instances. First, observe that $W_{v,\alpha\beta}$ can be large, even equal to V , as we saw in Example 27. However if $(v, \alpha, \beta) \notin \mathcal{Z}^{\mathcal{I}}$, by Lemma 25 the problem $\mathcal{I}_{W_{v,\alpha\beta}}$ splits into a union of disjoint problems over smaller domains, and so its minimality can be established by recursing to strictly smaller problems. On the other hand, if $(v, \alpha, \beta) \in \mathcal{Z}^{\mathcal{I}}$ then $\mathcal{I}_{W_{v,\alpha\beta}}$ may not split into such a union. Since we need an efficient procedure of establishing block-minimality, this explains the complications introduced in conditions (B2), (B3). In the case of (B2) $\mathcal{I}_{W_{v,\alpha\beta}}/\bar{\mu}^Y$ (see the definition of block-minimality) can be solved for each tuple $\mathbf{a} \in R$ by fixing the values from this tuple. Taking the quotient algebras of the remaining domains guarantees that we recurse to a strictly smaller instance. In the case of (B3) $\mathcal{I}_{W_{v,\alpha\beta} \cap W_{w,\gamma\delta}}/\bar{\mu}^Y$ splits into disjoint subproblems, and we branch on those strictly smaller subproblems.

Lemma 28. *Let $\mathcal{I} = (V, \mathcal{C})$ be a (2,3)-minimal instance. Then by solving a quadratic number of strictly smaller CSPs \mathcal{I} can be transformed to an equivalent block-minimal instance \mathcal{I}' .*

5.3 The Algorithm

In the algorithm we distinguish three cases depending on the presence of semi-lattice edges and quasi-centralizers of the domains of variables. In each case we employ different methods of solving or reducing the instance to a strictly smaller one. Algorithm 1 gives a more formal description of the solution algorithm.

Let $\mathcal{I} = (V, \mathcal{C})$ be a subdirectly irreducible, (2,3)-minimal instance. Let $\text{Center}(\mathcal{I})$ denote the set of variables $v \in V$ such that $\zeta(\underline{0}_v, \mu_v) = \underline{1}_v$. Let $\mu_v^* = \mu_v$ if $v \in \text{MAX}(\mathcal{I}) \cap \text{Center}(\mathcal{I})$ and $\mu_v^* = \underline{0}_v$ otherwise.

Semilattice Free Domains. If no domain of \mathcal{I} contains a semilattice edge then by Proposition 19 \mathcal{I} can be solved in polynomial time, using the few subalgebras algorithm, as shown in [21, 51].

Small Centralizers. If $\mu_v^* = \underline{0}_v$ for all $v \in V$, block-minimality guarantees the existence of a solution, as Theorem 29 shows, and we can use Lemma 28 to solve the instance.

Theorem 29. *If \mathcal{I} is subdirectly irreducible, (2,3)-minimal, block-minimal, and $\text{MAX}(\mathcal{I}) \cap \text{Center}(\mathcal{I}) = \emptyset$, then \mathcal{I} has a solution.*

Proof of Theorem 29 is the most technically involved part of our result.

Large Centralizers. Suppose that $\text{MAX}(\mathcal{I}) \cap \text{Center}(\mathcal{I}) \neq \emptyset$. In this case the algorithm proceeds in three steps.

Step 1. Consider the problem $\mathcal{I}/\bar{\mu}^*$. We establish the global 1-minimality of this problem. If it is tightened in the process, we start solving the new problem from scratch. To check global 1-minimality, for each $v \in V$ and every

$a \in \mathbb{A}_v/\mu_v^*$, we need to find a solution of the instance, or show it does not exists. To this end, add the constraint $\langle(v), \{a\}\rangle$ to $\mathcal{I}/\bar{\mu}^*$. The resulting problem belongs to $\text{CSP}(\mathbb{A})$, since \mathbb{A}_v is idempotent, and hence $\{a\}$ is a subalgebra of \mathbb{A}_v/μ_v^* . Then we establish (2,3)-minimality and block minimality of the resulting problem. Let us denote it \mathcal{I}' . There are two possibilities. First, if $\text{size}(\mathcal{I}') < \text{size}(\mathcal{I})$ then \mathcal{I}' is a problem strictly smaller than \mathcal{I} and can be solved by recursively calling Algorithm 1 on \mathcal{I}' . If $\text{size}(\mathcal{I}') = \text{size}(\mathcal{I})$ then, as all the domains \mathbb{A}_v of maximal size for $v \in \text{Center}(\mathcal{I})$ are replaced with their quotient algebras, there is $w \notin \text{Center}(\mathcal{I})$ such that $|\mathbb{A}_w| = \text{size}(\mathcal{I})$ and \mathbb{A}_w is not semilattice free. Therefore for every $u \in \text{Center}(\mathcal{I}')$, for the corresponding domain \mathbb{A}'_u we have $|\mathbb{A}'_u| < \text{size}(\mathcal{I}) = \text{size}(\mathcal{I}')$. Thus, $\text{MAX}(\mathcal{I}') \cap \text{Center}(\mathcal{I}') = \emptyset$, and \mathcal{I}' has a solution by Theorem 29.

Step 2. For every $v \in \text{Center}(\mathcal{I})$ we find a solution φ of $\mathcal{I}/\bar{\mu}^*$ satisfying the following condition: there is $a \in \mathbb{A}_v$ such that $\{a, \varphi(v)\}$ is a semilattice edge if $\mu_v^* = \underline{0}_v$, or, if $\mu_v^* = \mu_v$, there is $b \in \varphi(v)$ such that $\{a, b\}$ is a semilattice edge. Take $b \in \mathbb{A}_v/\mu_v^*$ such that $\{a, b\}$ is a semilattice edge in \mathbb{A}_v/μ_v^* for some $a \in \mathbb{A}_v/\mu_v^*$. Since $\mathcal{I}/\bar{\mu}^*$ is globally 1-minimal, there is a solution $\varphi_{v,b}$ such that $\varphi_{v,b}(v) = b$.

Step 3. We apply the transformation of \mathcal{I} suggested by Maroti in [62]. For a solution φ of $\mathcal{I}/\bar{\mu}^*$ by $\mathcal{I} \cdot \varphi$ we denote the instance (V, \mathcal{C}_φ) given by the rule: for every $C = \langle \mathbf{s}, R \rangle \in \mathcal{C}$ the set \mathcal{C}_φ contains a constraint $\langle \mathbf{s}, R \cdot \varphi \rangle$. To construct $R \cdot \varphi$ choose a tuple $\mathbf{b} \in R$ such that $\mathbf{b}[v]^{\mu_v^*} = \varphi(v)$ for all $v \in \mathbf{s}$; this is possible because φ is a solution of $\mathcal{I}/\bar{\mu}^*$. Then set $R \cdot \varphi = \{\mathbf{a} \cdot \mathbf{b} \mid \mathbf{a} \in R\}$. By the results of [62] it can be shown that the instance $\mathcal{I} \cdot \varphi$ has a solution if and only if \mathcal{I} does. Let $\mathcal{I}' = (\dots(\mathcal{I} \cdot \varphi_{v_1,b_1}) \cdot \dots) \cdot \varphi_{v_\ell,b_\ell}$, where $\varphi_{v_1,b_1}, \dots, \varphi_{v_\ell,b_\ell}$ are the solutions chosen in Step 2. We have $\text{size}(\mathcal{I}') < \text{size}(\mathcal{I})$.

This last case can be summarized as the following

Theorem 30. *If $\mathcal{I}/\bar{\mu}^*$ is globally 1-minimal, then \mathcal{I} can be reduced in polynomial time to a strictly smaller instance over an algebra satisfying the conditions of the Dichotomy Conjecture.*

Example 31. We illustrate the algorithm `SolveCSP` on the instance from Example 26. Recall that the domain of each variable is \mathbb{A}_M , its monolith is θ , and $\zeta(\underline{0}, \theta)$ is the full relation. This means that $\text{size}(\mathcal{I}) = 3$, $\text{MAX}(\mathcal{I}) = V$ and $\text{Center}(\mathcal{I}) = V$, as well. Therefore we are in the case of large centralizers. Set $\mu_{v_i}^* = \theta_i$ for each $i \in [5]$ and consider the problem $\mathcal{I}/\bar{\mu}^* = (V, \{C_1^* = \langle \mathbf{s}_1, R_1^* \rangle, C_2^* = \langle \mathbf{s}_2, R_2^* \rangle\})$, where $R^* = \{(0^\theta, 0^\theta, 0^\theta), (2^\theta, 2^\theta, 0^\theta), (2^\theta, 2^\theta, 2^\theta)\}$. It is an easy excercise to show that this instance is globally 1-minimal (every value 0^θ can be extended to the all- 0^θ solution, and every value 2^θ can be extended to the all- 2^θ solution). This completes *Step 1*. For every variable v_i we choose $b \in \mathbb{A}_M/\theta$ such that for some $a \in \mathbb{A}_M/\theta$ the pair $\{a, b\}$ is a semilattice edge. Since \mathbb{A}_M/θ is a 2-element semilattice, setting $b = 0^\theta$ and $a = 2^\theta$ is the only choice. Therefore all solutions $\varphi_{v_i,0^\theta}$ in our case can be chosen to be φ , where $\varphi(v_i) = 0^\theta$; and *Step 2* is completed. For *Step 3* first note that in \mathbb{A}_M the operation r plays

Algorithm 1. Procedure SolveCSP

Require: A CSP instance $\mathcal{I} = (V, \mathcal{C})$ from $\text{CSP}(\mathbb{A})$

Ensure: A solution of \mathcal{I} if one exists, ‘NO’ otherwise

- 1: **if** all the domains are semilattice free **then**
- 2: Solve \mathcal{I} using the few subpowers algorithm and RETURN the answer
- 3: **end if**
- 4: Transform \mathcal{I} to a subdirectly irreducible, block-minimal, and (2,3)-minimal instance
- 5: $\mu_v^* = \mu_v$ for $v \in \text{MAX}(\mathcal{I}) \cap \text{Center}(\mathcal{I})$ and $\mu_v^* = \underline{0}_v$ otherwise
- 6: $\mathcal{I}^* = \mathcal{I}/\underline{\mu}^*$
- 7: *%% Check the 1-minimality of \mathcal{I}^**
- 8: **for** every $v \in V$ and $a \in \mathbb{A}_v/\mu_v^*$ **do**
- 9: $\mathcal{I}' = \mathcal{I}_{(v,a)}^*$ *%% Add the constraint $\langle(v), \{a\}\rangle$ fixing the value of v to a*
- 10: Transform \mathcal{I}' to a subdirectly irreducible, (2,3)-minimal instance \mathcal{I}''
- 11: **if** $\text{size}(\mathcal{I}'') < \text{size}(\mathcal{I})$ **then**
- 12: Call SolveCSP on \mathcal{I}'' and flag a if \mathcal{I}'' has no solution
- 13: **else**
- 14: Establish block-minimality of \mathcal{I}'' ; if the problem changes, return to Step 10
- 15: If the resulting instance is empty, flag element a
- 16: **end if**
- 17: **end for**
- 18: If there are flagged values, tighten the instance by removing the flagged elements and start over
- 19: Use Theorem 30 to reduce \mathcal{I} to an instance \mathcal{I}' with $\text{size}(\mathcal{I}') < \text{size}(\mathcal{I})$
- 20: Call SolveCSP on \mathcal{I}' and RETURN the answer

the role of multiplication \cdot defined in Lemma 20. Then for each of the constraints C^1, C^2 choose a representative $\mathbf{a}_1 \in R_1 \cap (\varphi(v_1) \times \varphi(v_2) \times \varphi(v_3)) = R_1 \cap \{0,1\}^3$, $\mathbf{a}_2 \in R_2 \cap (\varphi(v_2) \times \varphi(v_4) \times \varphi(v_5)) = R_2 \cap \{0,1\}^3$, and set $\mathcal{I}' = (\{v_1, \dots, v_5\}, \{C'_1 = \langle(v_1, v_2, v_3), R'_1\rangle, C'_2 = \langle(v_2, v_4, v_5), R'_2\rangle\})$, where $R'_1 = r(R_1, \mathbf{a})$, $R'_2 = r(R_2, \mathbf{b})$. Since $r(2,0) = r(2,1) = 0$, regardless of the choice of \mathbf{a}, \mathbf{b} in our case $R'_1 \subseteq R_1$, $R'_2 \subseteq R_2$, and are invariant with respect to the affine operation of \mathbb{Z}_2 . Therefore the instance \mathcal{I}' can be viewed as a system of linear equations over \mathbb{Z}_2 (this system is actually empty in our case), and can be easily solved. \diamond

Using Lemma 28 and Theorems 29, 30 it is not difficult to see that the algorithm runs in polynomial time. Indeed, every time it makes a recursive call it calls on a problem whose non-semilattice free domains of maximal cardinality have strictly smaller size, and therefore the depth of recursion is bounded by $|\mathbb{A}|$ if we are dealing with $\text{CSP}(\mathbb{A})$.

References

1. Barto, L.: The dichotomy for conservative constraint satisfaction problems revisited. In: LICS, pp. 301–310 (2011)
2. Barto, L.: The collapse of the bounded width hierarchy. *J. Logic Comput.* **26**(3), 923–943 (2014)
3. Barto, L., Kozik, M.: Absorbing subalgebras, cyclic terms, and the constraint satisfaction problem. *Log. Methods Comput. Sci.* **8**(1), 1–26 (2012)
4. Barto, L., Kozik, M.: Constraint satisfaction problems solvable by local consistency methods. *J. ACM* **61**(1), 3:1–3:19 (2014)
5. Barto, L., Kozik, M.: Robustly solvable constraint satisfaction problems. *SIAM J. Comput.* **45**(4), 1646–1669 (2016)
6. Barto, L., Kozik, M.: Absorption in universal algebra and CSP. In: The Constraint Satisfaction Problem: Complexity and Approximability, pp. 45–77 (2017)
7. Barto, L., Krokhin, A.A., Willard, R.: Polymorphisms, and how to use them. In: The Constraint Satisfaction Problem: Complexity and Approximability, pp. 1–44 (2017)
8. Barto, L., Pinsker, M., Opršal, J.: The wonderland of reflections. *Israel J. Math.* (2018, to appear)
9. Berman, J., Idziak, P., Marković, P., McKenzie, R., Valeriote, M., Willard, R.: Varieties with few subalgebras of powers. *Trans. Amer. Math. Soc.* **362**(3), 1445–1473 (2010)
10. Bodnarchuk, V., Kaluzhnin, L., Kotov, V., Romov, B.: Galois theory for post algebras. I. *Kibernetika* **3**, 1–10 (1969)
11. Börner, F., Bulatov, A.A., Chen, H., Jeavons, P., Krokhin, A.A.: The complexity of constraint satisfaction games and QCSP. *Inf. Comput.* **207**(9), 923–944 (2009)
12. Bulatov, A.: A dichotomy theorem for constraints on a three-element set. In: FOCS, pp. 649–658 (2002)
13. Bulatov, A.A.: Tractable conservative constraint satisfaction problems. In: LICS, pp. 321–330 (2003)
14. Bulatov, A.A.: A graph of a relational structure and constraint satisfaction problems. In: LICS, pp. 448–457 (2004)
15. Bulatov, A.A.: H-coloring dichotomy revisited. *Theor. Comp. Sci.* **349**(1), 31–39 (2005)
16. Bulatov, A.A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM* **53**(1), 66–120 (2006)
17. Bulatov, A.A.: Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.* **12**(4), 24 (2011)
18. Bulatov, A.A.: The complexity of the counting constraint satisfaction problem. *J. ACM* **60**(5), 34:1–34:41 (2013)
19. Bulatov, A.A.: Conservative constraint satisfaction re-revisited. *J. Comput. Syst. Sci.* **82**(2), 347–356 (2016)
20. Bulatov, A.A.: Graphs of finite algebras, edges, and connectivity. *CoRR.* abs/1601.07403 (2016)
21. Bulatov, A.A.: Graphs of relational structures: restricted types. In: LICS, pp. 642–651 (2016)
22. Bulatov, A.A.: A dichotomy theorem for nonuniform CSPs. *CoRR.* abs/1703.03021 (2017)
23. Bulatov, A.A.: A dichotomy theorem for nonuniform CSPs. In: FOCS, pp. 319–330 (2017)

24. Bulatov, A.A., Dalmau, V.: A simple algorithm for Mal'tsev constraints. *SIAM J. Comput.* **36**(1), 16–27 (2006)
25. Bulatov, A.A., Jeavons, P., Krokhin, A.A.: Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.* **34**(3), 720–742 (2005)
26. Bulatov, A.A., Krokhin, A., Larose, B.: Dualities for constraint satisfaction problems. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) *Complexity of Constraints – An Overview of Current Research Themes [Result of a Dagstuhl Seminar]*. LNCS, vol. 5250, pp. 93–124. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92800-3_5
27. Bulatov, A.A., Valerio, M.A.: Recent results on the algebraic approach to the CSP. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) *Complexity of Constraints – An Overview of Current Research Themes [Result of a Dagstuhl Seminar]*. LNCS, vol. 5250, pp. 68–92. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92800-3_4
28. Cai, J., Chen, X.: Complexity of counting CSP with complex weights. In: STOC, pp. 909–920 (2012)
29. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: STOC, pp. 77–90 (1977)
30. Cooper, M.C., Zivny, S.: Hybrid tractable classes of constraint problems. In: *The Constraint Satisfaction Problem: Complexity and Approximability*, pp. 113–135 (2017)
31. Creignou, N., Khanna, S., Sudan, M.: *Complexity Classifications of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications, vol. 7. SIAM (2001)
32. Dalmau, V.: Generalized majority-minority operations are tractable. *Log. Methods Comput. Sci.* **2**(4), 1–15 (2006)
33. Dalmau, V., Kozik, M., Krokhin, A.A., Makarychev, K., Makarychev, Y., Oprsal, J.: Robust algorithms with polynomial loss for near-unanimity CSPs. In: SODA, pp. 340–357 (2017)
34. Dechter, R.: *Constraint Processing*. Morgan Kaufmann Publishers, Burlington (2003)
35. Dyer, M.E., Greenhill, C.S.: The complexity of counting graph homomorphisms. *Random Struct. Algorithms* **17**(3–4), 260–289 (2000)
36. Feder, T., Vardi, M.: Monotone monadic SNP and constraint satisfaction. In: STOC, pp. 612–622 (1993)
37. Feder, T., Vardi, M.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. *SIAM J. Comput.* **28**, 57–104 (1998)
38. Feder, T., Hell, P., Klein, S., Motwani, R.: List partitions. *SIAM J. Discrete Math.* **16**(3), 449–478 (2003)
39. Feder, T., Hell, P., Tucker-Nally, K.: Digraph matrix partitions and trigraph homomorphisms. *Discr. Appl. Math.* **154**(17), 2458–2469 (2006)
40. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. *J. ACM* **49**(6), 716–752 (2002)
41. Freese, R., McKenzie, R.: *Commutator Theory for Congruence Modular Varieties*. London Mathematical Society Lecture Note Series, vol. 125. Cambridge University Press, Cambridge (1987)
42. Geiger, D.: Closed systems of function and predicates. *Pacific J. Math.* **27**(1), 95–100 (1968)
43. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. *Artif. Intell.* **124**(2), 243–282 (2000)

44. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.* **64**(3), 579–627 (2002)
45. Grätzer, G.: Universal Algebra, 2nd edn. Springer, New York (2008). <https://doi.org/10.1007/978-0-387-77487-9>
46. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM* **54**(1), 1:1–1:24 (2007)
47. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. *ACM Trans. Algorithms* **11**(1), 4:1–4:20 (2014)
48. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford Lecture Series in Mathematics and its Applications, vol. 28. Oxford University Press, Oxford (2004)
49. Hell, P., Nešetřil, J.: On the complexity of H -coloring. *J. Comb. Theory Ser. B* **48**, 92–110 (1990)
50. Hobby, D., McKenzie, R.: The Structure of Finite Algebras. Contemporary Mathematics, vol. 76. American Mathematical Society, Providence (1988)
51. Idziak, P.M., Markovic, P., McKenzie, R., Valeriote, M., Willard, R.: Tractability and learnability arising from algebras with few subpowers. *SIAM J. Comput.* **39**(7), 3023–3037 (2010)
52. Jeavons, P., Cohen, D.A., Gyssens, M.: Closure properties of constraints. *J. ACM* **44**(4), 527–548 (1997)
53. Jeavons, P., Cohen, D., Cooper, M.: Constraints, consistency and closure. *Artif. Intell.* **101**(1–2), 251–265 (1998)
54. Jerrum, M.: Counting constraint satisfaction problems. In: The Constraint Satisfaction Problem: Complexity and Approximability, pp. 205–231 (2017)
55. Klíma, O., Tesson, P., Thérien, D.: Dichotomies in the complexity of solving systems of equations over finite semigroups. *Theory Comput. Syst.* **40**(3), 263–297 (2007)
56. Kolmogorov, V., Krokhin, A.A., Rolínek, M.: The complexity of general-valued CSPs. *SIAM J. Comput.* **46**(3), 1087–1110 (2017)
57. Krokhin, A.A., Zivny, S.: The complexity of valued CSPs. In: The Constraint Satisfaction Problem: Complexity and Approximability, pp. 233–266 (2017)
58. Larose, B., Zádori, L.: Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *IJAC* **16**(3), 563–582 (2006)
59. Mackworth, A.: Consistency in networks of relations. *Artif. Intell.* **8**, 99–118 (1977)
60. Kozik, M., Krokhin, A., Valeriote, M., Willard, R.: Characterizations of several Maltsev conditions. *Algebra Univers.* **73**(3–4), 205–224 (2015)
61. Markovic, P.: The complexity of CSPs on a 4-element set. Oral Communication (2011)
62. Maróti, M.: Tree on top of Malcev (2011). <http://www.math.u-szeged.hu/~mmaroti/pdf/200x%20Tree%20on%20top%20of%20Maltsev.pdf>
63. Maróti, M., McKenzie, R.: Existence theorems for weakly symmetric operations. *Algebra Univers.* **59**(3–4), 463–489 (2008)
64. Marx, D.: Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM* **60**(6), 42:1–42:51 (2013)
65. Post, E.: The Two-Valued Iterative Systems of Mathematical Logic. Annals Mathematical Studies, vol. 5. Princeton University Press, Princeton (1941)
66. Raghavendra, P.: Optimal algorithms and inapproximability results for every CSP? In: STOC, pp. 245–254 (2008)
67. Reingold, O.: Undirected connectivity in log-space. *J. ACM* **55**(4), 17:1–17:24 (2008)
68. Schaefer, T.: The complexity of satisfiability problems. In: STOC, pp. 216–226 (1978)

69. Thapper, J., Zivny, S.: The complexity of finite-valued CSPs. *J. ACM* **63**(4), 37:1–37:33 (2016)
70. Zhuk, D.: On CSP dichotomy conjecture. In: *Arbeitstagung Allgemeine Algebra, AAA 1992*, p. 32 (2016)
71. Zhuk, D.: The proof of CSP dichotomy conjecture for 5-element domain. In: *Arbeitstagung Allgemeine Algebra AAA 1991* (2016)
72. Zhuk, D.: A proof of CSP dichotomy conjecture. In: *FOCS*, pp. 331–342 (2017)



Sliding Window Algorithms for Regular Languages

Moses Ganardi, Danny Hucke, and Markus Lohrey^(✉)

Department für Elektrotechnik und Informatik, Universität Siegen,
Hölderlinstrasse 3, 57076 Siegen, Germany
{ganardi,hucke,lohrey}@eti.uni-siegen.de

Abstract. This paper gives a survey on recent results for sliding window streaming algorithms for regular languages. Details can be found in the recent papers [18, 19].

Keywords: Automata theory · Streaming algorithms
Sliding window algorithms · Regular languages

1 Introduction

Streaming algorithms. Streaming algorithms [1] process an input sequence $a_1a_2 \cdots a_m$ of data values from left to right. Random access to the input is not allowed, and at time instant t the algorithm has only direct access to the current data value a_t and its goal is to compute an output value $f(a_1a_2 \cdots a_t)$ for a certain function f . During this process it is quite often infeasible and in many settings also not necessary to store the whole history $a_1a_2 \cdots a_t$. Such a scenario arises for instance when searching in large databases (e.g., genome databases or web databases), analyzing internet traffic (e.g. click stream analysis), and monitoring networks. Ideally, a streaming algorithm works in constant space, in which case the algorithm is a deterministic finite automaton (DFA), but polylogarithmic space with respect to the input length might be acceptable, too.

The first papers on streaming algorithms as we know them today are usually attributed to Munro and Paterson [24] and Flajolet and Martin [16], although the principle idea goes back to the work on online machines by Hartmanis et al. from the 1960's [23, 27]. Extremely influential for the area of streaming algorithms was the paper of Alon et al. [2] on computing frequency moments in the streaming model.

The sliding window model. The streaming model sketched above is also known as the *standard streaming model*. One missing aspect of the standard model is the fact that data items are usually no longer important after a certain time. For instance, in the analysis of a time series as it may arise in medical monitoring, web tracking, or financial monitoring, data items are usually outdated after a certain time. The *sliding window model* is an alternative streaming model that can capture this aspect. Two variants of the sliding window model can be found in the literature; see e.g. [3]:

- *Fixed-size model:* In this model the algorithm works on a sliding window of a certain fixed length n . While reading the input word $w = a_1 a_2 \cdots a_m$ symbol by symbol from left to right it has to output at every time instant $n \leq t \leq m$ a value $f(a_{t-n+1} \cdots a_t)$ that depends on the n last symbols. The number n is also called the *window size*.
- *Variable-size model:* Here, the sliding window $a_{t-n+1} a_{t-n+2} \cdots a_t$ is determined by an adversary. At every time instant the adversary can either remove the first data value from the sliding window (expiration of a value), or add a new data value at the right end (arrival of a new value).

In the seminal paper of Datar et al. [15], where the (fixed-size) sliding window model was introduced, the authors show how to maintain the number of 1's in a sliding window of size n over the alphabet $\{0, 1\}$ in space $\frac{1}{\varepsilon} \cdot \log^2 n$ if one allows a multiplicative error of $1 \pm \varepsilon$. A matching lower bound is proved as well in [15]. Following the work of Datar et al., a large number of papers that deal with the approximation of statistical data over sliding windows followed. Let us mention the work on computation of the variance and k -median [4], quantiles [3], and entropy [8] over sliding windows. Other computational problems that have been considered for the sliding window model include optimal sampling [9], various pattern matching problems [10–13], database querying (e.g. processing of join queries [20]) and graph problems (e.g. checking for connectivity and computation of matchings, spanners, and spanning trees [14]). Further references on the sliding window model can be found in the surveys [1, Chapter 8] and [7].

Language recognition in the streaming model. A natural problem that has been surprisingly neglected for the streaming model (in particular the sliding window model) is language recognition. The goal is to check whether an input string belongs to a given language L . Let us quote Magniez et al. [22]: “Few applications [of streaming] have been made in the context of formal languages, which may have impact on massive data such as DNA sequences and large XML files. For instance, in the context of databases, properties decidable by streaming algorithm have been studied [25, 26], but only in the restricted case of deterministic and constant memory space algorithms.” For Magniez et al. this was the starting point to study language recognition in the streaming model. Thereby they restricted their attention to the above mentioned standard streaming model. Note that in the standard model the membership problem for a regular language is trivial to solve: One simply has to simulate a DFA on the stream and thereby only store the current state of the DFA. In [22] the authors presented a randomized streaming algorithm for the (non-regular) Dyck language D_s with s pairs of parenthesis that works in space $\mathcal{O}(\sqrt{n} \log n)$ and time $\text{polylog}(n)$ per symbol. The algorithm has a one-sided error: it accepts with small probability also words that do not belong to D_s . An almost matching lower bound of $\Omega(\sqrt{n} \log n)$ for two-sided errors is proved in [22] as well. Further investigations on streaming language recognition for various subclasses of context-free languages can be found in [5, 6, 17, 21].

Let us emphasize that all the papers cited in the last paragraph exclusively deal with the standard streaming model. Language recognition problems for the

sliding window model have been completely neglected so far. This is surprising, since even for regular languages the membership problem becomes non-trivial in the sliding window model. This was the starting point for our work on streaming [18, 19] that mainly deals with the membership problem for regular languages in the sliding window model. Before we explain the results from [18, 19] in Sect. 4, we formally define the various streaming models in the next section.

2 Streaming Algorithms as Automata

We use standard definitions from automata theory. A *nondeterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ where Q is a finite set of states, Σ is an alphabet, $I \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of final states. A *deterministic finite automaton* (DFA) $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ has a single initial state $q_0 \in Q$ instead of I and a transition function $\delta: Q \times \Sigma \rightarrow Q$ instead of the transition relation Δ . A *deterministic automaton* has the same format as a DFA, except that the state set Q is not required to be finite. If \mathcal{A} is deterministic, the transition function δ is extended to a function $\delta: Q \times \Sigma^* \rightarrow Q$ in the usual way and we define $\mathcal{A}(x) = \delta(q_0, x)$ for $x \in \Sigma^*$. The language accepted by \mathcal{A} is $L(\mathcal{A}) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$.

Recall from the introduction that a streaming algorithm reads an input word $w = a_1 a_2 \cdots a_m$ from left to right and computes at every time instant $0 \leq t \leq m$ a value $f(a_1 a_2 \cdots a_t)$ for some target function f . In this paper we make two restrictions:

- The data values a_i are from some finite alphabet Σ . This rules out streaming algorithms that read for instance a natural number in each time unit.
- The target function is boolean-valued, i.e., $f : \Sigma^* \rightarrow \{0, 1\}$.

These two restrictions imply that a streaming algorithm can be seen as a deterministic automaton, possibly with an infinite state set. Moreover, in order to make statements about the space complexity of a streaming algorithm, we also have to fix an encoding of the automaton states by bit strings. Formally, a *streaming algorithm* over Σ is a deterministic (possibly infinite) automaton $\mathcal{A} = (S, \Sigma, s_0, \delta, F)$, where the states are encoded by bit strings. We describe this encoding by an injective function $\text{enc}: S \rightarrow \{0, 1\}^*$. The *space function* $\text{space}(\mathcal{A}, \cdot): \Sigma^* \rightarrow \mathbb{N}$ specifies the space used by \mathcal{A} on a certain input: For $w \in \Sigma^*$ let $\text{space}(\mathcal{A}, w) = \max\{\|\text{enc}(\mathcal{A}(u))\| : u \in \text{Pref}(w)\}$, where $\text{Pref}(w)$ denotes the set of prefixes of w . We also say that \mathcal{A} is a *streaming algorithm* for the accepted language $L(\mathcal{A})$.

3 Sliding Window Streaming Models

In the above streaming model, the output value of the streaming algorithm at time t depends on the whole past $a_1 a_2 \cdots a_t$ of the data stream. However, in many practical applications one is only interested in the relevant part of the past. Two formalizations of “relevant past” can be found in the literature:

- Only the suffix of $a_1a_2 \cdots a_t$ of length n is relevant. Here, n is a fixed constant. This streaming model is called the *fixed-size sliding window model*.
- The relevant suffix of $a_1a_2 \cdots a_t$ is determined by an adversary. In this model, at every time instant the adversary can either remove the first symbol from the active window (expiration of a data value), or add a new symbol at the right end (arrival of a new data value). This streaming model is also called the *variable-size sliding window model*.

In the following two subsections, we formally define these two models.

3.1 Fixed-Size Sliding Windows

Given a word $w = a_1a_2 \cdots a_m \in \Sigma^*$ and a window length $n \geq 0$, we define $\text{last}_n(w) \in \Sigma^n$ by

$$\text{last}_n(w) = \begin{cases} a_{m-n+1}a_{m-n+2} \cdots a_m, & \text{if } n \leq m, \\ \square^{n-m}a_1 \cdots a_m, & \text{if } n > m, \end{cases}$$

which is called the *active window*. Here $\square \in \Sigma$ is an arbitrary symbol, which fills the initial window. A sequence $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$ is a *fixed-size sliding window algorithm* for a language $L \subseteq \Sigma^*$ if each \mathcal{A}_n is a streaming algorithm for the language

$$L_n := \{w \in \Sigma^* : \text{last}_n(w) \in L\}.$$

Its *space complexity* is the function $f_{\mathcal{A}}: \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\}$ where $f_{\mathcal{A}}(n)$ is the maximal encoding length of a state in \mathcal{A}_n . Note that for every language L and every n the language L_n is regular, which ensures that \mathcal{A}_n can be chosen to be a DFA and hence $f_{\mathcal{A}}(n) < \infty$ for all $n \geq 0$.

A trivial fixed-size sliding window algorithm $\mathcal{B} = (\mathcal{B}_n)_{n \geq 0}$ for L is obtained by taking the DFAs $\mathcal{B}_n = (\Sigma^n, \Sigma, \square^n, \delta_n, \Sigma^n \cap L)$ with the transition function $\delta_n(au, b) = ub$ for $a, b \in \Sigma$, $u \in \Sigma^{n-1}$. It stores the active window explicitly in the state. States of \mathcal{B}_n can be encoded with $\mathcal{O}(\log |\Sigma| \cdot n)$ bits. By minimizing each \mathcal{B}_n , we obtain an *optimal fixed-size sliding window algorithm* \mathcal{A}_L for L . Finally, we define $F_L(n) = f_{\mathcal{A}_L}(n)$. Thus, F_L is the space complexity of an optimal fixed-size sliding window algorithm for L . Notice that F_L is not necessarily monotonic. For instance, take $L = \{au : u \in \{a, b\}^*, |u| \text{ odd}\}$. Then, we have $F_L(2n) \in \Theta(n)$ (see Example 5 below) and $F_L(2n + 1) \in \mathcal{O}(1)$. The above trivial algorithm \mathcal{B} yields $F_L(n) \in \mathcal{O}(n)$ for every language L .

Note that the fixed-size sliding window model is a *non-uniform* model: for every window size we have a separate streaming algorithm and these algorithms do not have to follow a common pattern. Working with a non-uniform model makes lower bounds stronger. In contrast, the variable-size sliding window model that we discuss next is a uniform model in the sense that there is a single streaming algorithm that works for every window length.

3.2 Variable-Size Sliding Windows

For an alphabet Σ we define the extended alphabet $\overline{\Sigma} = \Sigma \cup \{\downarrow\}$. In the variable-size model the *active window* $\text{wnd}(u) \in \Sigma^*$ for a stream $u \in \overline{\Sigma}^*$ is defined as follows:

- $\text{wnd}(\varepsilon) = \varepsilon$
- $\text{wnd}(ua) = \text{wnd}(u) a$ for $a \in \Sigma$
- $\text{wnd}(u\downarrow) = \varepsilon$ if $\text{wnd}(u) = \varepsilon$
- $\text{wnd}(u\downarrow) = v$ if $\text{wnd}(u) = av$ for $a \in \Sigma$

A *variable-size sliding window algorithm* for a language $L \subseteq \Sigma^*$ is a streaming algorithm \mathcal{A} for $\{w \in \overline{\Sigma}^* : \text{wnd}(w) \in L\}$. Its *space complexity* is the function $v_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\}$ mapping each window length n to the maximum number of bits used by \mathcal{A} on inputs producing an active window of size at most n . Formally, it is the function

$$v_{\mathcal{A}}(n) = \max\{\text{space}(\mathcal{A}, u) : u \in \overline{\Sigma}^*, |\text{wnd}(v)| \leq n \text{ for all } v \in \text{Pref}(u)\}.$$

Note that $v_{\mathcal{A}}$ is a monotonic function. It is not completely obvious that every language L has an optimal variable-size sliding window algorithm:

Lemma 1. *For every language $L \subseteq \Sigma^*$ there exists a variable-size sliding window algorithm \mathcal{A} such that $v_{\mathcal{A}}(n) \leq v_{\mathcal{B}}(n)$ for every variable-size sliding window algorithm \mathcal{B} for L and every n .*

We define $V_L(n) = v_{\mathcal{A}}(n)$, where \mathcal{A} is a space *optimal variable-size sliding window algorithm* for L from Lemma 1. Since any algorithm in the variable-size model yields an algorithm in the fixed-size model, we have $F_L(n) \leq V_L(n)$.

It is not hard to show that any variable-size sliding window algorithm for a non-trivial language has to store enough information to recover the length of the active window. Hence, we have:

Lemma 2. *For every language $L \subseteq \Sigma^*$ such that $\emptyset \neq L \neq \Sigma^*$ we have $V_L(n) \in \Omega(\log n)$.*

4 Sliding Window Algorithms for Regular Languages

4.1 Space Trichotomy for Regular Languages

The main result from [19] is a space trichotomy for regular languages with respect to the two sliding window models: For every regular language, the space is either constant, logarithmic or linear.

Theorem 3 (space trichotomy [19]). *For every regular language L , exactly one of the following three cases holds:*

1. $F_L(n) \in \mathcal{O}(1)$
2. $F_L(n) \in \mathcal{O}(\log n) \setminus o(\log n)$ and $V_L(n) \in \Theta(\log n)$
3. $F_L(n) \in \mathcal{O}(n) \setminus o(n)$ and $V_L(n) \in \Theta(n)$

Note in particular that the class of regular languages that need logarithmic space (resp., linear space) is the same for the fixed-size model and the variable-size model. This is not true for constant space: For instance, for the language $L_1 = \{a, b\}^*a$ we have $F_{L_1}(n) \in \mathcal{O}(1)$, whereas Lemma 2 implies that $V_{L_1}(n) \in \Omega(\log n)$. Here are two further examples:

Example 4. For the language $L_2 = \{a, b\}^*a\{a, b\}^*$ we have $F_{L_2}(n) \in \Theta(\log n)$ as well as $V_{L_2}(n) \in \Theta(\log n)$. A variable-size sliding window algorithm for L_2 stores (i) the length of the active window and (ii) the position of the right-most a in the active window (or ∞ if the active window does not contain an a). For this, $\mathcal{O}(\log n)$ bits are sufficient. To see that $V_{L_2}(n) \in \Omega(\log n)$ consider a fixed-size sliding window algorithm $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$ for L_2 . Consider the window length n and all strings $w_i = b^{i-1}ab^{n-i}$ for $1 \leq i \leq n$. Then a standard fooling argument shows that for $1 \leq i < j \leq n$ the words w_i and w_j must lead to different states in \mathcal{A}_n . Hence, \mathcal{A}_n has at least n states, which implies that $f_{\mathcal{A}}(n) \in \Omega(\log n)$.

Example 5. For the language $L_3 = a\{a, b\}^*$ we have $F_{L_3}(n) \in \Theta(n)$ as well as $V_{L_3}(n) \in \Theta(n)$. It suffices to show the lower bound $F_{L_3}(n) \in \Omega(n)$. This follows from a fooling argument similar to the one from Example 4: Consider a fixed-size sliding window algorithm $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$ for L_3 . Consider the window length n . Then, all words from Σ^n have to lead to different states of \mathcal{A}_n , i.e., \mathcal{A}_n has at least $|\Sigma|^n$ states which implies that $f_{\mathcal{A}}(n) \in \Omega(n)$.

The reader might wonder why we write $F_L(n) \in \mathcal{O}(\log n) \setminus o(\log n)$ (resp., $F_L(n) \in \mathcal{O}(n) \setminus o(n)$) instead of $F_L(n) \in \Theta(\log n)$ (resp., $F_L(n) \in \Theta(n)$) in point 2 (resp., point 3) of Theorem 3. To see that this is indeed necessary, consider again the language $L = \{au: u \in \{a, b\}^*, |u| \text{ odd}\}$. Then, we have $F_L(2n) \in \mathcal{O}(n) \setminus o(n)$, but $F_L(n) \notin \Omega(n)$, since $F_L(2n+1) \in \mathcal{O}(1)$. On the other hand, for the variable-size model, we can make the stronger statement $V_L(n) \in \Theta(\log n)$ (resp., $V_L(n) \in \Theta(n)$) due to the monotonicity of $V_L(n)$.

4.2 Characterizations of the Space Classes

We use the following notation for the three classes from Theorem 3:

- $\text{Reg}(1)$ is the class of all regular languages for which point 1 from Theorem 3 holds.
- $\text{Reg}(\log n)$ is the class of all regular languages for which point 2 from Theorem 3 holds.
- $\text{Reg}(n)$ is the class of all regular languages for which point 3 from Theorem 3 holds.

Theorem 3 does not give language theoretical characterizations of the above three classes. Such characterizations were provided in [18]. We need the following definitions.

A language $L \subseteq \Sigma^*$ is called *k-suffix testable* if for all $x, y \in \Sigma^*$ and $z \in \Sigma^k$ we have: $xz \in L$ if and only if $yz \in L$. Equivalently, L is a Boolean combination

of languages of the form Σ^*w where $w \in \Sigma^{\leq k}$. We call L *suffix testable* if it is k -suffix testable for some $k \geq 0$. Clearly, every finite language is suffix testable: if $L \subseteq \Sigma^{\leq k}$ then L is $(k+1)$ -suffix testable. Moreover, every suffix testable language is regular. A language $L \subseteq \Sigma^*$ is called a *length language* if for all $n \in \mathbb{N}$, either $\Sigma^n \subseteq L$ or $L \cap \Sigma^n = \emptyset$.

Theorem 6 ([18]). *Reg(1) is the class of all finite Boolean combinations of suffix testable languages and regular length languages.*

In order to characterize the class $\text{Reg}(\log n)$ we need the following definition: A language $L \subseteq \Sigma^*$ is called a *left ideal* if $\Sigma^*L \subseteq L$.

Theorem 7 ([18]). *Reg($\log n$) is the class of all finite Boolean combinations of regular left ideals and regular length languages.*

The class $\text{Reg}(\log n)$ has a useful characterization in terms of automata as well. A *strongly connected component* (SCC for short) of a DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is an inclusion-maximal subset $C \subseteq Q$ such that for all $p, q \in C$ there exist words $u, v \in \Sigma^*$ such that $\delta(p, u) = q$ and $\delta(q, v) = p$. A singleton SCC $\{q\}$ is called trivial if $\delta(q, u) \neq q$ for all non-empty words u (i.e., q is not on a cycle). An SCC $C \subseteq Q$ is *well-behaved* if for all $q \in C$ and $u, v \in \Sigma^*$ with $|u| = |v|$ and $\delta(q, u), \delta(q, v) \in C$ we have: $\delta(q, u) \in F$ if and only if $\delta(q, v) \in F$. Clearly, every trivial SCC is well-behaved. If every SCC in \mathcal{A} which is reachable from q_0 is well-behaved, then \mathcal{A} is called *well-behaved*. Figure 1 shows an example of a well-behaved DFA. Its SCCs are $\{0\}$ (which is trivial), $\{1\}$, $\{2, 3\}$, and $\{4\}$.

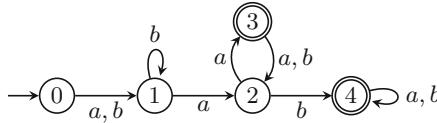


Fig. 1. A well-behaved DFA.

For a word $w = a_1a_2 \cdots a_n$, let $w^{\text{rev}} = a_n \cdots a_2a_1$ be the reversed word.

Theorem 8 ([18]). *A regular language L belongs to $\text{Reg}(\log n)$ if and only if the reversed language $L^{\text{rev}} = \{w^{\text{rev}} : w \in L\}$ is accepted by a well-behaved DFA. Moreover, this holds if and only if every DFA for L^{rev} is well-behaved.*

Let us sketch the logspace (variable-size) sliding-window algorithm for a regular language L such that L^{rev} is accepted by a well-behaved DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$. Let w be the current active window. Assume that we store for every state $q \in Q$ the run of \mathcal{A} on the word w^{rev} (i.e., the sequence of visited states) that starts in q . This information would allow to make the necessary updates and queries for the variable-size model (i.e., removing the left-most symbol from the window, adding a symbol on the right, and testing membership in L). But the space needed to

store these runs would be linear in the length w . The main observation for the logspace algorithm is that since \mathcal{A} is well-behaved it suffices to store for each of the above runs a so called path summary that is defined as follows: Let C_1, \dots, C_k be the sequence of pairwise different SCCs that are visited by the run in that order. The path summary of the run is the sequence $(q_1, \ell_1, q_2, \ell_2, \dots, q_k, \ell_k)$ where q_i is the first state in C_i visited by ρ , and $\ell_i \geq 0$ is the number of transitions from the first occurrence of q_i until the first state from C_{i+1} (or until the end for q_k).

The lower bound $\Omega(n)$ for the space complexity (with respect to the fixed-size model) in case L^{rev} is accepted by a DFA that is not well-behaved can be shown by a fooling argument similar to the one from Example 5.

4.3 Uniform Space Bounds

In the statements from Sects. 4.1 and 4.2 we always assume a fixed regular language. When, e.g., saying that $V_L \in \mathcal{O}(\log n)$ then the \mathcal{O} -constant depends on the automaton size. Using the path summaries mentioned in Sect. 4.2, one can show:

Theorem 9 ([18]). *Let \mathcal{A} be a DFA or NFA with m states such that $L = L(\mathcal{A}) \in \text{Reg}(\log n)$ is well-behaved. There are constants c_m, d_m that only depend on m such that the following holds:*

- If \mathcal{A} is a DFA then $V_L(n) \leq (2^m \cdot m + 1) \cdot \log n + c_m$ for n large enough.
- If \mathcal{A} is an NFA then $V_L(n) \leq (4^m + 1) \cdot \log n + d_m$ for n large enough.

The following theorem states a lower bound for the fixed-size model (and hence also for the variable-size model) that almost matches the space bound in Theorem 9:

Theorem 10 ([18]). *For all $k \geq 1$ there exists a language $L_k \subseteq \{0, \dots, k\}^*$ recognized by a DFA with $k + 3$ states such that $L_k \in \text{Reg}(\log n)$ and $F_{L_k}(n) \geq (2^k - 1) \cdot (\log n - k)$.*

It is open whether in Theorem 10 the alphabet $\{0, \dots, k\}$ can be replaced by a fixed (e.g. binary) alphabet without changing the lower bound.

4.4 Deciding the Space Classes

Theorem 8 leads to a decision algorithm for the class $\text{Reg}(\log n)$: Given a DFA (or NFA) for a regular language L , one first constructs a DFA \mathcal{A} for L^{rev} using standard automata constructions and then checks whether \mathcal{A} is well-behaved. But this algorithm is not very efficient since in general the size of a DFA for L^{rev} is exponential in the size of an automaton for L , even if the latter automaton is deterministic. In [18] we provided a more efficient algorithm for the class $\text{Reg}(\log n)$ as well as $\text{Reg}(1)$ in case the input automaton is deterministic.

Theorem 11 ([18]). *Given a DFA for a regular language L , it is NL-complete to check whether $L \in \text{Reg}(\log n)$, respectively $L \in \text{Reg}(1)$.*

As one might expect, if the input automaton is nondeterministic than the complexity increases by one exponent:

Theorem 12 ([18]). *Given an NFA for a regular language L , it is PSPACE-complete to check whether $L \in \text{Reg}(\log n)$, respectively $L \in \text{Reg}(1)$.*

5 Future Work

The space trichotomy theorem for regular languages (Theorem 3) leads to several interesting research questions:

- Do similar results hold for context-free languages or subclasses like deterministic context-free languages or visibly pushdown languages?
- Is it possible to generalize Theorem 3 to a randomized setting? In fact, most papers on streaming algorithms deal with randomized streaming algorithms.

These topics will be the content of two forthcoming papers.

References

1. Aggarwal, C.C.: Data Streams - Models and Algorithms. Springer, Heidelberg (2007). <https://doi.org/10.1007/978-0-387-47534-9>
2. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.* **58**(1), 137–147 (1999)
3. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proceedings of PODS 2004, pp. 286–296. ACM (2004)
4. Babcock, B., Datar, M., Motwani, R., O’Callaghan, L.: Maintaining variance and k-medians over data stream windows. In: Proceedings of PODS 2003, pp. 234–243. ACM (2003)
5. Babu, A., Limaye, N., Radhakrishnan, J., Varma, G.: Streaming algorithms for language recognition problems. *Theor. Comput. Sci.* **494**, 13–23 (2013)
6. Babu, A., Limaye, N., Varma, G.: Streaming algorithms for some problems in log-space. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 94–104. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13562-0_10
7. Braverman, V.: Sliding window algorithms. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms, pp. 2006–2011. Springer, Heidelberg (2016). <https://doi.org/10.1007/978-1-4939-2864-4>
8. Braverman, V., Ostrovsky, R.: Smooth histograms for sliding windows. In: Proceedings of FOCS 2007, pp. 283–293. IEEE Computer Society (2007)
9. Braverman, V., Ostrovsky, R., Zaniolo, C.: Optimal sampling from sliding windows. *J. Comput. Syst. Sci.* **78**(1), 260–272 (2012)
10. Breslauer, D., Galil, Z.: Real-time streaming string-matching. *ACM Trans. Algorithms* **10**(4), 22:1–22:12 (2014)

11. Clifford, R., Fontaine, A., Porat, E., Sach, B., Starikovskaya, T.: Dictionary matching in a stream. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 361–372. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_31
12. Clifford, R., Fontaine, A., Porat, E., Sach, B., Starikovskaya, T.: The k-mismatch problem revisited. In: Proceedings of SODA 2016, pp. 2039–2052. SIAM (2016)
13. Clifford, R., Starikovskaya, T.: Approximate hamming distance in a stream. In: Proceedings of ICALP 2016. LIPIcs, vol. 55, pp. 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
14. Crouch, M.S., McGregor, A., Stubbs, D.: Dynamic graphs in the sliding-window model. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 337–348. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40450-4_29
15. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM J. Comput. **31**(6), 1794–1813 (2002)
16. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci. **31**(2), 182–209 (1985)
17. Fran ois, N., Magniez, F., de Rougemont, M., Serre, O.: Streaming property testing of visibly pushdown languages. In: Proceedings of ESA 2016. LIPIcs, vol. 57, pp. 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum f r Informatik (2016)
18. Ganardi, M., Hucke, D., K onig, D., Lohrey, M., Mamouras, K.: Automata theory on sliding windows. In: Proceedings of STACS 2018. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum f r Informatik (2018, to appear)
19. Ganardi, M., Hucke, D., Lohrey, M.: Querying regular languages over sliding windows. In: Proceedings of FSTTCS 2016. LIPIcs, vol. 65, pp. 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum f r Informatik (2016)
20. Golab, L.,  Ozs u, M.T.: Processing sliding window multi-joins in continuous queries over data streams. In: Proceedings of VLDB 2003, pp. 500–511. Morgan Kaufmann (2003)
21. Krebs, A., Limaye, N., Srinivasan, S.: Streaming algorithms for recognizing nearly well-parenthesized expressions. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 412–423. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22993-0_38
22. Magniez, F., Mathieu, C., Nayak, A.: Recognizing well-parenthesized expressions in the streaming model. SIAM J. Comput. **43**(6), 1880–1905 (2014)
23. Lewis II, P.M., Stearns, R.E., Hartmanis, J.: Memory bounds for recognition of context-free and context-sensitive languages. In: Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design, pp. 191–202. IEEE Computer Society (1965)
24. Munro, J.I., Paterson, M.: Selection and sorting with limited storage. Theor. Comput. Sci. **12**, 315–323 (1980)
25. Segoufin, L., Sirangelo, C.: Constant-memory validation of streaming XML documents against DTDs. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 299–313. Springer, Heidelberg (2006). https://doi.org/10.1007/11965893_21
26. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: Proceedings of PODS 2002, pp. 53–64. ACM (2002)
27. Stearns, R.E., Hartmanis, J., Lewis II, P.M.: Hierarchies of memory limited computations. In: Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design, pp. 179–190. IEEE Computer Society (1965)



Underlying Principles and Recurring Ideas of Formal Grammars

Alexander Okhotin^(✉)

St. Petersburg State University, 7/9 Universitetskaya nab.,
Saint Petersburg 199034, Russia
alexander.okhotin@spbu.ru

Abstract. The paper investigates some of the fundamental ideas of the context-free grammar theory, as they are applied to several extensions and subclasses of context-free grammars. For these grammar families, including multi-component grammars, tree-adjoining grammars, conjunctive grammars and Boolean grammars, a summary of the following properties is given: parse trees, language equations, closure under several operations, normal forms, parsing algorithms, representation in the FO(LFP) logic, representations by automata and by categorial grammars, homomorphic characterizations, hardest language theorems, pumping lemmata and other limitations, computational complexity.

1 Introduction

Formal grammars are both a classical subject, presented in all computer science curricula, and a topic of ongoing theoretical and applied research. Over half a century, the classroom presentation of grammars has stabilized to a certain collection of essential facts on context-free grammars, all established in the 1960s. However, the research on formal grammars did not end in the 1960s: many results on context-free grammars were established, and also quite a few new grammar models have been introduced over the years. Some of the new models did not work out well: indeed, when there are no examples of sensible language specifications by the proposed grammars, no efficient algorithms and even no theoretical results of any value, such a model deserves oblivion. On the other hand, a few models were found to share some useful properties of context-free grammars, which confirmed their value, and the research on such models has carried on.

The aim of this paper is to present several grammar families with good properties *together*, emphasizing their common underlying principles, and tracing what happens with the fundamental ideas of formal grammar theory, as they are applied to these families. Even putting these grammar families together already requires a certain reappraisal of foundations. At the dawn of computer science, the study of formal grammars was dominated by the string-rewriting approach in Chomsky's [17] early work. By now, the Chomsky hierarchy of rewriting systems has retained a purely historical value, and modern presentations of the basics of

formal grammars, such as the one in Sipser’s [78] textbook, omit it altogether. Rather than try fitting all kind of grammars into this framework, one should look for the actual common ground for the existing formal grammar families, and present them in light of the current state of knowledge.

This common ground is the understanding of formal grammars as a logic for describing the syntax, and definitions of grammars through inference rules. For instance, whereas the string-rewriting approach is to rewrite S into NP VP by a rule $S \rightarrow \text{NP VP}$, and then proceed with rewriting NP VP into, e.g., **Every man is mortal**, using inference rules, a proposition $S(\text{Every man is mortal})$ is inferred from $\text{NP}(\text{Every man})$ and $\text{VP}(\text{is mortal})$. This more modern understanding of grammars led to many developments in formal grammar theory which would not be possible under the string-rewriting approach. In particular, it is essential in the definitions of multi-component grammars of Seki et al. [77], it led to the important representation of formal grammars in the FO(LFP) logic, given by Rounds [74], and to a uniform description of parsing algorithms by Pereira and Warren [70].

Once the grammar families are uniformly defined, this paper aims to trace the main recurring ideas in the field, all originating from the context-free grammar theory, as they are applied to various new grammar models. It turns out that almost none of these ideas are exclusive to context-free grammars, and their applicability to other grammar families confirms that the formal grammar theory is larger than just the context-free grammar theory. As well put by Chytil [19], “A tour through the theory of context-free languages reveals an interesting feature of many celebrated results—something like ‘stability’ of their proofs, or ‘buried reserves’ contained in them”.

Before proceeding any further, there is a certain small detail to mention that will likely arouse some controversy. When Chomsky [17] proposed the term “context-free grammar”, he considered the idea of a phrase-structure rule applicable only in specified contexts, and attempted to implement this idea by a string-rewriting system. Thus, the ordinary kind of grammars got a name “context-free” to distinguish them from the attempted new model. However, it was soon found that context-sensitive string rewriting does not implement any sensible syntactic descriptions, and, as a formal grammar model, it makes no sense. As of today, even though the term “context-free” has been repeated for decades, *no major model in the theory of formal grammars uses contexts of any kind*. Thus, this name is no longer suitable for distinguishing the main model of syntax from other related models. Due to the significance of this model in computer science and its central position in the theory of formal grammars, the suggested alternative name is

an ordinary grammar.

This name is used throughout in this paper. All other grammar families are named after the feature that makes them different from the ordinary grammars: linear grammars, unambiguous grammars, conjunctive grammars, multi-component grammars, etc.

2 Grammar Families

A formal grammar is a mathematically precise syntactical description, which formalizes natural definitions, such as “*A noun phrase followed by a verb phrase is a sentence*” or “*An arithmetical expression enclosed in brackets is also an arithmetical expression*”. The following standard example of a grammar illustrates this kind of definitions.

Example 1. The set of well-nested strings of brackets over the alphabet $\Sigma = \{a, b\}$, known as the *Dyck language*, is defined by the following conditions.

- The empty string ε is well-nested.
- If w is a well-nested string, then so is awb .
- If u and v are well-nested strings, then so is uv .

In the notation of formal grammars, this definition is expressed as follows, where S is the syntactic category of well-nested strings.

$$S \rightarrow \varepsilon \mid aSb \mid SS$$

The S on the left-hand side, followed by an arrow, means that this is the definition of strings with the property S . The right-hand side defines the possible form of such strings, with alternative structure separated by vertical lines. Any occurrence of S on the right-hand sides stands for an arbitrary string with the property S .

Actually, there is a bizarre detail: a rule in a grammar means a logical implication from its right-hand side to its left-hand side, and therefore there are all reasons to write the arrow in the opposite direction, as in $S \leftarrow \varepsilon \mid aSb \mid SS$. This paper follows the traditional, incorrect direction of arrows.

The general form of this kind of syntactic descriptions is universally known.

Definition 1 (Chomsky [17]). An ordinary grammar (Chomsky’s “context-free”) is a quadruple $G = (\Sigma, N, R, S)$, where

- Σ is the alphabet of the language being defined;
- N is the set of syntactic categories defined in the grammar, which are typically called nonterminal symbols;
- R is the set of rules, each of the form $A \rightarrow u_0B_1u_1 \dots B_\ell u_\ell$, with $\ell \geq 0$, $u_0, u_1, \dots, u_\ell \in \Sigma^*$ and $B_1, \dots, B_\ell \in N$.
- the nonterminal symbol S represents the syntactic category of grammatically correct strings (“sentences” of the language).

Each rule $A \rightarrow u_0B_1u_1 \dots B_\ell u_\ell$ defines a possible structure of strings with the property A ; it means that if each string v_i has the property B_i , then the string $u_0v_1u_1 \dots v_\ell u_\ell$ has the property A . If there are multiple rules for A , this means that strings with the property A may be of any of the given forms.

All other grammar families are obtained by modifying some elements of this definition. For that reason, in order to see what kind of grammar families can there be, one should first take note, what is this definition comprised of.

Definition 2. A family of grammars is characterized by the following three elements.

Constituents, that is, fragments, from which a sentence is formed. Fragments are joined together to form larger fragments, until the entire sentence is ultimately obtained. In most grammar families, the constituents are substrings of the sentence.

Operations on constituents used to express other constituents.

Logical operations used to define syntactic conditions.

A grammar family is thus a specialized logic for reasoning about the properties of constituents.

In ordinary grammars, constituents are **substrings**, the only operation on constituents is **concatenation**, that is, writing two substrings one after another to form a longer substring. The only logical operation is **disjunction** of syntactical conditions.

How can one modify this model? One of its special cases, the *linear grammars*, is obtained by restricting the operations on constituents: instead of concatenation of arbitrary substrings, as in a rule $S \rightarrow SS$, linear grammars may only concatenate fixed symbols to a substring from both sides, as in a rule $S \rightarrow aSb$.

Another special case, the *unambiguous grammars*, restricts both the concatenation and the disjunction. In an unambiguous grammar, whenever a concatenation of two languages, K and L , is expressed, it is required that every string w has at most one partition $w = uv$ into a string u in K and a string v in L . Furthermore, whenever a disjunction is used in an unambiguous grammar, at most one alternative must be true.

Several grammar families are obtained by splitting the alphabet Σ into left brackets and right brackets, restricting the constituents to be **well-nested substrings**, and then restricting the concatenation, ensuring, in particular, that only well-nested strings could be expressed. Such models proposed by McNaughton [53] (“parenthesis grammars”), by Ginsburg and Harrison [30] (“bracketed grammars”) by Berstel and Boasson [9] (“balanced grammars”) and by Alur and Madhusudan [3] (“visibly pushdown grammars”).

One possible direction for extending ordinary grammars is to augment the set of allowed logical operations. Adding the conjunction of any syntactical conditions leads to *conjunctive grammars*, defined as follows.

Definition 3 [57, 65]. A conjunctive grammar is a quadruple $G = (\Sigma, N, R, S)$, where Σ , N and S are as in an ordinary grammar, and each rule in R is of the form $A \rightarrow \alpha_1 \& \dots \& \alpha_m$, where $m \geq 1$ and $\alpha_1, \dots, \alpha_m \in (\Sigma \cup N)^*$.

Such a rule asserts that if a string w can be represented according to each conjunct α_i , then w has the property A . More precisely, let $\alpha_i = u_{i,0}B_{i,1}u_{i,1}B_{i,2}\dots u_{i,\ell_i-1}B_{i,\ell_i}u_{i,\ell_i}$, with $B_{i,1}, \dots, B_{i,\ell_i} \in N$, $\ell_i \geq 0$ and $u_{i,0}, u_{i,1}, \dots, u_{i,\ell_i} \in \Sigma^*$. Then, if, for each $i \in \{1, \dots, m\}$, the string w is representable as $w = u_{i,0}v_{i,1}u_{i,1}v_{i,2}\dots u_{i,\ell_i-1}v_{i,\ell_i}u_{i,\ell_i}$, where each $v_{i,1}$ has the property B_i , then w has the property A .

Thus, conjunctive grammars use **substrings** as constituents, **concatenation** as the only operation on substrings, and unrestricted **disjunction** and **conjunction** as logical operations. Years before this model was studied by the author [57], exactly the same model was defined by Szabari [81] in his unpublished Master's thesis. Closely related models were considered by Boullier [13] and by Lange [50]. Clark et al. [20] and Yoshinaka [90] defined an equivalent grammar model for their learning algorithm.

Example 2. The following conjunctive grammar describes the language $\{a^n b^n c^n \mid n \geq 0\}$.

$$\begin{aligned} S &\rightarrow AB \& DC \\ A &\rightarrow aA \mid \varepsilon \\ B &\rightarrow bBc \mid \varepsilon \\ C &\rightarrow cC \mid \varepsilon \\ D &\rightarrow aDb \mid \varepsilon \end{aligned}$$

The rules for the nonterminal symbols A , B , C and D do not use conjunction, and have the same meaning as in an ordinary grammar. In particular, A and C define the languages a^* and c^* , B defines $\{b^n c^n \mid n \geq 0\}$ and D defines $\{a^k b^k \mid k \geq 0\}$. Then the rule for S represents the strings in $a^* b^* c^*$ that have the same number of symbols b and c (ensured by AB), as well as the same number of symbols a and b (specified by DC). These are exactly all strings of the form $a^n b^n c^n$.

Boolean grammars are a further extension of conjunctive grammars that additionally allows the negation.

Definition 4 [58, 65]. A Boolean grammar is a quadruple $G = (\Sigma, N, R, S)$, where each rule in R is of the form $A \rightarrow \alpha_1 \& \dots \& \alpha_m \& \beta_1 \& \dots \& \beta_n$, with $m, n \geq 0$, $m + n \geq 1$ and $\alpha_1, \dots, \alpha_m, \beta_1, \dots, \beta_n \in (\Sigma \cup N)^*$.

This rule asserts that if a string w can be represented according to each positive conjunct α_i , and cannot be represented according to any negative conjunct β_j , then w has the property A .

Restricting conjunctive grammars and Boolean grammars to use *linear concatenation* yields *linear conjunctive grammars* and *linear Boolean grammars*, which are known to be equivalent in power [59].

Extended grammar models of another kind feature more complicated constituents, such as *substrings with a gap*. A substring with a gap is a pair (u, v) , with $u, v \in \Sigma^*$, which stands for any substring of the form u -GAP- v , or uxv with $x \in \Sigma^*$. Using these constituents facilitates describing syntactical links between two remote parts of the same sentence. A more complicated type of constituents are **substrings with multiple gaps**, that is, k -tuples of the form (u_1, u_2, \dots, u_k) , representing substrings u_1 -GAP- u_2 -GAP- \dots -GAP- u_k with $k - 1$ gaps. Grammars, in which every nonterminal symbol defines a set of such k -tuples, for some k , were independently defined by Seki et al. [77] (as “multiple context-free grammars”), and by Vijay-Shanker et al. [89] (as “linear context-free

rewriting systems”). In these grammars, which shall be called *multi-component grammars* in this paper, there is the following operation on constituents: given a k -tuple and an ℓ -tuple, the substrings therein may be concatenated with each other in any combinations, forming an m -tuple, with $1 \leq m \leq k + \ell$. The only logical operation is disjunction.

Definition 5 (Seki et al. [77]; Vijay-Shanker et al. [89]). A multi-component grammar is a quintuple $G = (\Sigma, N, \dim, R, S)$, where Σ and N are as in an ordinary grammar, the function $\dim: N \rightarrow \mathbb{N}$ defines the dimension of each nonterminal symbol, that is, the number of components its refers to, and each rule in R defines a possible structure of $(\dim A)$ -tuples with the property A as a composition of ℓ $(\dim B_i)$ -tuples with the property B_i , for some $B_1, \dots, B_\ell \in N$.

For each i , let $(x_{i,1}, \dots, x_{i,\dim B_i})$ be variables representing the components of a $(\dim B_i)$ -tuple with the property B_i . Let $\alpha_1, \dots, \alpha_{\dim A}$ be strings comprised of symbols from Σ and the variables $x_{i,j}$, with the condition that every variable $x_{i,j}$ occurs in $\alpha = \alpha_1 \dots \alpha_{\dim A}$ exactly once, and that, for every i , the variables $x_{i,1}, \dots, x_{i,\dim B_i}$ occur in α in their original order. These strings represent the form of the desired $(\dim A)$ -tuple, which is written down as the following rule in R .

$$A(\alpha_1, \dots, \alpha_{\dim A}) \rightarrow B_1(x_{1,1}, \dots, x_{1,\dim B_1}), \dots, B_\ell(x_{\ell,1}, \dots, x_{\ell,\dim B_\ell}),$$

The initial symbol has $\dim S = 1$. The dimension of the grammar is the maximum dimension of a nonterminal symbol: $\dim G = \max_{A \in N} \dim A$.

In this notation, a rule $A \rightarrow BC$ in an ordinary grammar is written down as $A(xy) \rightarrow B(x), C(y)$.

Example 3. The following multi-component grammar, with $\dim S = 1$ and $\dim A = 2$, defines the language $\{a^m b^n c^m d^n \mid m, n \geq 0\}$.

$$\begin{aligned} S(xy) &\rightarrow A(x,y) \\ A(axb,cyd) &\rightarrow A(x,y) \\ A(\varepsilon,\varepsilon) &\rightarrow \end{aligned}$$

An interesting special case of multi-component grammars are the *well-nested multi-component grammars*, in which the constituents are the same, and the operations on the constituents are restricted as follows. For each rule in R , for any variables $x_{i,j}$ and $x_{i,k}$ of some B_i , and for any variables $x_{i',m}$ and $x_{i',n}$ of some $B_{i'}$, their occurrences in $\alpha_1, \dots, \alpha_{\dim A}$ may not cross each other, as in $\dots x_{i,j} \dots x_{i',m} \dots x_{i,k} \dots x_{i',n} \dots$. The grammar in Example 3 is well-nested.

Well-nested 2-component grammars have received particular attention in the literature. In these grammars, constituents are substrings with at most one gap, and the operations are: *wrapping a pair* (u,v) around a pair (x,y) , producing a pair (ux,yv) ; *creating a pair* (w,ε) or (ε,w) out of a substring w ; *removing the gap* in a pair (u,v) , obtaining a string uv . This model was defined under a somewhat cryptic name of *head grammars* [71]; the definitions were later restated

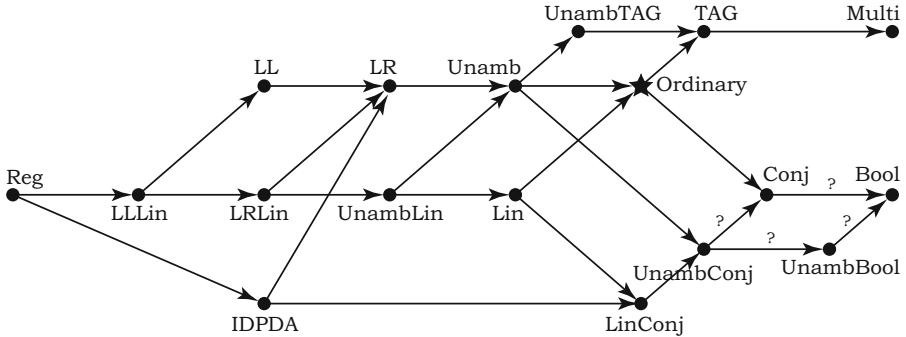


Fig. 1. Hierarchy of grammar families: inclusions.

in the modern form by Rounds [74]. These grammars were then found to be equivalent to the earlier studied *tree-adjoining grammars* [88].

The hierarchy of grammar families described in this paper is given in Fig. 1, where each arrow indicates containment; all inclusions without a question mark are known to be proper. The hierarchy is centered at the ordinary grammars (*Ordinary*), Chomsky’s “context-free”; other families are defined in reference to them. Their special cases are: the unambiguous grammars (*Unamb*); the LR and the LL grammars (*LR*, *LL*); and their subcases with linear concatenation (*Lin*, *UnambLin*, *LRLin*, *LLLIn*). A family of grammars dealing with well-nested strings is labelled with the name of the corresponding automata: the input-driven push-down automata (*IDPDA*). Then, there are the generalizations of ordinary grammars: well-nested 2-component grammars, labelled according to the name “tree-adjoining grammars” (*TAG*) conjunctive grammars (*Conj*), Boolean grammars (*Bool*), and their unambiguous subclasses (*UnambTAG*, *UnambConj*, *UnambBool*). Finally, there is the full family of multi-component grammars (*Multi*) and the linear conjunctive grammars (*LinConj*). The regular languages stand at the bottom of the hierarchy.

3 Inference Rules and Parse Trees

So far, grammars have been presented as an intuitively defined formalism for language specification, as in Example 1. Several mathematically precise definitions of grammars are known. Chomsky’s [17] string rewriting is one possible way of formalizing these syntactic descriptions. Although it is sufficient to reason about ordinary grammars, it does not explicitly follow the intuition behind Example 1, and is unsuitable for defining the extensions of ordinary grammars.

A much clearer definition, representing the right outlook on formal grammars, was presented, for instance, in a monograph by Kowalski [46, Chap. 3]. This definition regards a grammar as a logic, in which the properties of any string can be inferred from the properties of its substrings by the means of logical inference. For an ordinary grammar, the logic deals with *propositions* of the form

“a string $w \in \Sigma^*$ has the property $A \in N$ ”, denoted by $A(w)$. For instance, a rule $A \rightarrow BC$ allows the following deductions to be made, for all $u, v \in \Sigma^*$.

$$\frac{B(u) \quad C(v)}{A(uv)} \quad (A \rightarrow BC)$$

Example 4. For the ordinary grammar in Example 1, the well-nestedness of the string $abaabb$ is proved by the following logical derivation.

$$\frac{\begin{array}{c} S(\varepsilon) \\ \hline S(ab) \end{array}}{\begin{array}{c} S(\varepsilon) \quad \frac{S(\varepsilon)}{S(ab)} \\ \hline S(aabb) \end{array}} \quad \frac{S(aabb)}{S(abaabb)}$$

This object is essentially the parse tree.

In a conjunctive grammar, a proposition $A(w)$ can be deduced by a rule $A \rightarrow BC \& DE$ as follows, for any two partitions w into $w = uv = xy$, with $u, v, x, y \in \Sigma^*$.

$$\frac{B(u) \quad C(v) \quad D(x) \quad E(y)}{A(w)} \quad (A \rightarrow BC \& DE)$$

Example 5. For the conjunctive grammar in Example 2, the following derivation establishes that the string $w = abc$ is a well-formed sentence.

$$\frac{\begin{array}{cccc} A(\varepsilon) & B(\varepsilon) & D(\varepsilon) & C(\varepsilon) \\ \hline A(a) & B(bc) & D(ab) & C(c) \end{array}}{S(abc)}$$

This inference, as well as any such inference for a conjunctive grammar, represents a parse tree with shared leaves: indeed, there are only three symbols in the string, which are shared by two subtrees corresponding to the two conjuncts in the rule for S . This sharing represents multiple structures for the same substring.

Definition by logical inference perfectly works for multi-component grammars.

Example 6. The multi-component grammar in Example 3 defines the string $w = aabbccdd$ by the following logical derivation.

$$\frac{\begin{array}{c} A(\varepsilon, \varepsilon) \\ \hline A(ab, cd) \end{array}}{\begin{array}{c} A(ab, cd) \\ \hline A(aabb, ccdd) \end{array}} \quad \frac{A(aabb, ccdd)}{S(aabbccdd)}$$

The definitions by logical derivation cannot implement the negation in the rules, such as in Boolean grammars. Negation can be formalized within the more general approach to defining grammars explained in the next section.

Conclusions. Grammar families without negation can be defined by logical derivations. This is a formalization of a most intuitive object, a parse tree.

4 Language Equations

Another approach to defining the language described by a grammar is based upon a variant of informal definitions, such as the one in Example 1, this time written down as “if and only if” conditions.

Example 7. A string $w \in \{a, b\}^*$ is well-nested if and only if

- either $w = \varepsilon$,
- or $w = aub$, for some well-nested string u ,
- or $w = uv$, for some well-nested strings u and v .

This can be written down as the following *language equation*, with the set of well-nested strings X as the unknown.

$$X = \{\varepsilon\} \cup (\{a\} \cdot X \cdot \{b\}) \cup (X \cdot X)$$

The Dyck language is among the solutions of this equation, actually the least solution with respect to inclusion.

The representation of grammars by language equations was discovered by Ginsburg and Rice [31]. An ordinary grammar $G = (\Sigma, N, R, S)$, with $N = \{X_1, \dots, X_n\}$ is represented by a system of equations with the following form, where each nonterminal symbol X_i becomes a variable.

$$\left\{ \begin{array}{l} X_1 = \varphi_1(X_1, \dots, X_n) \\ \vdots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{array} \right. \quad (*)$$

For each X_i , the right-hand side of its equation is a union of concatenations, representing the rules for X_i , with each occurrence of a symbol $a \in \Sigma$ represented by a constant language $\{a\}$, as shown in the above Example 7.

Language equations corresponding to conjunctive grammars represent the conjunction operator by intersection of languages on the right-hand sides.

Language equations can be naturally extended to the cases of tree-adjoining grammars and multi-component grammars. These equations will use *sets of pairs* or *k-tuples of strings* as unknowns. The operations on the right-hand sides are: the set-theoretic union, and the operations on constituents extended to sets.

Language equations are particularly essential for defining Boolean grammars [58], but there are a few non-trivial details to take care of. First, the negation in the rules is implemented by a complementation operation on sets. However, in this case the equations might have no solutions, such as the equation $S = \overline{S}$ corresponding to the grammar $S \rightarrow \neg S$. One possibility of handling this problem is to impose a certain condition on grammars, under which the grammar $S \rightarrow \neg S$ is dismissed as ill-formed [58]. An improved definition of Boolean

grammars done in terms of three-valued logic was given by Kountouriotis et al. [45]: under their definition, every grammar defines a three-valued language, with each string having a “well-formed”, “ill-formed” or “undefined” status; in particular, the grammar $S \rightarrow \neg S$ defines a language with all strings undefined.

Language equations of the general form, with unrestricted left-hand sides, can define computationally universal sets by their solutions [41, 47, 60, 63], which makes them completely useless for the purpose of defining grammar models.

Conclusions. All grammar families have definitions by language equations, this is a common underlying principle.

5 Expressibility of Operations

Closure properties are among the main indicators of the expressive power of a grammar family: closure under some operation means that this operation can be expressed in those grammars. Some closure results are immediate, because the operation is a part of the formalism of rules: for instance, union and concatenation are expressible both in ordinary grammars and in conjunctive grammars, and in the latter, intersection is expressible as well. There are numerous closure results for various grammar families, and only a brief account of some recurring properties can be given in this paper.

Perhaps the most fundamental closure result for ordinary grammars is their closure under intersection with a regular language, proved by Bar-Hillel et al. [6]: in their construction, an ordinary grammar $G = (\Sigma, N, R, S)$ and a finite automaton $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ are combined into a new grammar G' with the nonterminal symbols of the form $A_{p,q}$, with $A \in N$ and $p, q \in Q$, which defines all strings with the property A , on which the finite automaton moves from state p to state q . Parse trees in G' have the same structure as parse trees in G , with the extra information on the computation of \mathcal{A} .

By a similar method, one can implement a nondeterministic finite transduction (NFT) on an ordinary grammar G , producing a grammar G' for the set of all strings that can be emitted by the transducer while processing a string defined by G [29]. This general closure result has important special cases, such as homomorphisms, inverse homomorphisms, inverse deterministic finite transductions (DFT), the set of prefixes, etc.

All the above results apply to linear grammars and to multi-component grammars. For unambiguous grammars, the construction for intersection with a regular language still applies, but the construction simulating an NFT is no longer applicable. In fact, this family is not closed already under homomorphisms. However, the special case of the latter construction involving an inverse DFT applies for unambiguous grammars as well.

For conjunctive and Boolean grammars, similarly, there is a non-closure under homomorphisms [57], but the construction for the closure under inverse DFT can still be generalized [51].

6 Normal Forms

Numerous normal form theorems exist for ordinary grammars, and some of them have been extended to other grammar families.

The most well-known normal form is the *Chomsky normal form* for ordinary grammars, which requires all rules to be of the form $A \rightarrow BC$, with $B, C \in N$, or $A \rightarrow a$, with $a \in \Sigma$. The known transformation to the normal form proceeds by first ensuring that the right-hand sides are of length at most one (this incurs a linear increase in the size of the grammar), then eliminating *null rules* of the form $A \rightarrow \varepsilon$ (linear increase), and finally eliminating *chain rules* of the form $A \rightarrow B$ (quadratic increase). A lower bound of the order $n^{\frac{3}{2}-o(1)}$ has been established by Blum [11].

The Chomsky normal form exists for several subclasses of ordinary grammars: namely, for unambiguous grammars, for LL grammars and for LR grammars.

For conjunctive grammars, there is a direct generalization, called the *binary normal form* [57]. A conjunctive grammar in the binary normal form has all rules of the form $A \rightarrow B_1C_1 \& \dots \& B_mC_m$, with $m \geq 1$ and $B_i, C_i \in N$, or of the form $A \rightarrow a$, with $a \in \Sigma$. The transformation follows the same plan as for ordinary grammar, consecutively eliminating *null conjuncts* in rules of the form $A \rightarrow \varepsilon \& \dots$, and *unit conjuncts* in rules of the form $A \rightarrow B \& \dots$. However, the elimination of unit conjuncts incurs an exponential blow-up. No lower bound on the complexity of this transformation is known.

In the *Greibach normal form* [32] for ordinary grammars, every rule is of the form $A \rightarrow a\alpha$, with $a \in \Sigma$ and $\alpha \in N^*$. The best known transformation to the Greibach normal form was developed by Rosenkrantz [73] and by Urbanek [85]: it transforms a grammar in the Chomsky normal form to a grammar in the Greibach normal form with a cubic blow-up. An $O(n^2)$ lower bound on the complexity of the latter transformation was proved by Kelemenová [44].

The definition of the Greibach normal form can be naturally extended to conjunctive grammars, which would have all rules of the form $A \rightarrow a\alpha_1 \& \dots \& a\alpha_m$, with $a \in \Sigma$, $m \geq 1$ and $\alpha_1, \dots, \alpha_m \in N^*$. However, it is not known whether every conjunctive grammar can be transformed to this form [65, Problem 5].

A stronger version of the Greibach normal form for ordinary grammars was defined by Rosenkrantz [73]. In the *Rosenkrantz normal form* (also known as double Greibach normal form), every rule is of the form $A \rightarrow a\alpha d$, with $a, d \in \Sigma$ and $\alpha \in (\Sigma \cup N)^*$, or $A \rightarrow a$. The transformation from the Chomsky normal form to the Rosenkrantz normal form, as stated by Engelfriet [24], produces a grammar of size $O(n^{10})$, no lower bounds are known.

In the *operator normal form* for ordinary grammars, defined by Floyd [27], each rule is of the form $A \rightarrow u_0B_1u_1B_2u_2 \dots u_{k-1}B_ku_k$, with $k \geq 0$, $u_0, u_k \in \Sigma^*$, $B_1, \dots, B_k \in N$ and $u_1, \dots, u_{k-1} \in \Sigma^+$. This normal form extends to conjunctive grammars [68]: every grammar can be transformed to one with all rules of the form $A \rightarrow B_1a_1C_1 \& \dots \& B_ma_mC_m$, with $m \geq 1$, $B_i, C_i \in N$, and $a_i \in \Sigma$, or $A \rightarrow a$, or $S \rightarrow aA$, as long as S never appears on the right-hand sides of any rules.

There is a generalized normal form theorem by Blattner and Ginsburg [10], in which all rules are either of the form $A \rightarrow w$, with $w \in \Sigma^*$, or of the form $A \rightarrow u_0B_1u_1 \dots B_\ell u_\ell$, where ℓ and the lengths u_i may be fixed almost arbitrarily.

Conclusions. Ordinary grammars can be normalized in many different ways. Some normal form theorems are extended to conjunctive grammars; other normal forms could be generalized as well, but it is open whether every grammar can be transformed to those forms. For multi-component grammars, such normal forms would be hard to formulate: apparently, one has to deal with grammars of more or less the general form.

7 Parsing

Most parsing algorithms applicable to grammars of the general form are based upon the dynamic programming method. The most well-known is the Cocke–Kasami–Younger algorithm, which, for an ordinary grammar $G = (\Sigma, N, R, S)$ in the Chomsky normal form, given a string $w = a_1 \dots a_n$, constructs, for each substring $a_{i+1} \dots a_j$, the sets $T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$. Its running time is $\Theta(n^3)$ and it uses $\Theta(n^2)$ space. The algorithm applies to conjunctive grammars [57] and to Boolean grammars [58] without any changes.

Valiant [86] proved that the same data structure can be constructed in time $O(n^\omega)$, where $O(n^\omega)$ is the number of operations needed to multiply two $n \times n$ matrices. Valiant’s algorithm was originally presented in a generalized algebraic form, which complicates the creation of any derivative algorithms. However, it can be reformulated easier and in elementary terms, and then it directly applies to conjunctive and Boolean grammars [66], with the same time complexity $O(n^\omega)$.

The ideas of the Cocke–Kasami–Younger algorithm are directly extended to multi-component grammars [77, Sect. 3.2], obtaining an algorithm working in time $O(n^k)$, where k depends on the number of components and on the complexity of rules. In particular, for tree-adjoining grammars, the basic algorithm works in time $O(n^6)$. Both algorithms can be accelerated using fast matrix multiplication [54, 72].

A variant of the Cocke–Kasami–Younger algorithm, the *Kasami–Torii algorithm* constructs a different data structure encoding the same sets $T_{i,j}$: for each position j and for each nonterminal symbol A , this is the sorted list of all positions i with $A \in T_{i,j}$. The resulting algorithm works in time $O(n^3)$ in the worst case, and in time $O(n^2)$ for unambiguous grammars, as well as for unambiguous conjunctive and unambiguous Boolean grammars [62]. Using the same idea, parsing for tree-adjoining grammars can be accelerated to $O(n^4)$.

There are well-known subclasses of ordinary grammars that have linear-time parsing algorithms: the LL grammars and the LR grammars. The LL grammars have a generalization for Boolean grammars [61], the LR grammars have an extension for conjunctive grammars [2].

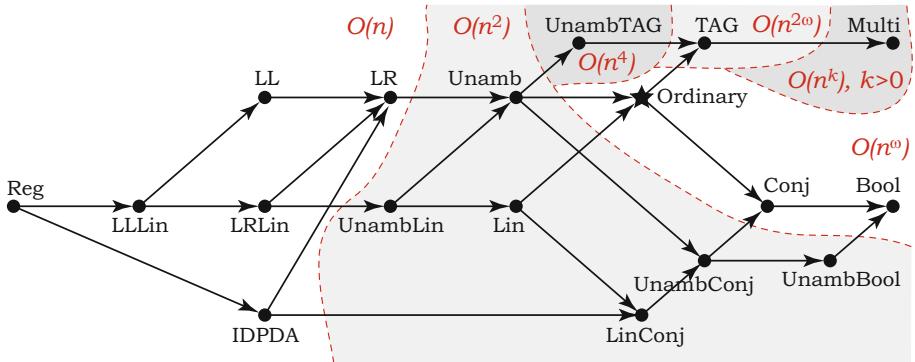


Fig. 2. Hierarchy of grammar families: parsing time.

Conclusions. Each grammar family has a simple dynamic programming parsing algorithm that works in polynomial time, with the degree of the polynomial depending on the grammar family. The presence of Boolean operations does not affect the complexity, whereas for multi-component grammars, the degree of the polynomial is determined by the structure of rules. For all known grammar families, the algorithm can be accelerated by using fast matrix multiplication, and with the unambiguity assumption, the algorithm can be accelerated even further. Linear-time algorithms for subclasses of ordinary grammars are well-developed, but their extensions to more powerful grammar families need further study. The running time for different families is compared in Fig. 2.

8 Representation in the FO(LFP) Logic

In 1988, Rounds [74] has identified the previously unknown foundation for formal grammars: the FO(LFP) logic. The fundamental theoretical property of this logic, discovered by Immerman [37] and by Vardi [87], is that it can describe exactly all problems decidable in polynomial time: the complexity class P. As it turned out, it is not only that all grammar families can be described within this logic—they can be described exactly according to their definition! Then, each grammar family becomes a clearly defined special case of the FO(LFP) logic.

Definition 6. Let Σ be an alphabet, let N be a finite set of predicate symbols, with each $A \in N$ having a finite number of arguments, denoted by $\dim A$.

The logic uses first-order variables referring to positions in the string. Positions are given by **terms**, defined as follows.

- The first position, the last position and any variables are terms.
- If t is a term, then so are $t + 1$ and $t - 1$.

Next, a formula is defined as follows.

- If $A \in N$ is a predicate symbol with $\dim A = k$ and t_1, \dots, t_k are terms, then $A(t_1, \dots, t_k)$ is a formula;
- If $a \in \Sigma$ is a symbol and t is a term, then $\varphi = a(t)$ is a formula;
- If t and t' are terms, then $t < t'$ and $t = t'$ are formulae;
- If φ and ψ are formulae, then so are $\varphi \vee \psi$ and $\varphi \wedge \psi$;
- If φ is a formula and x is a free variable in φ , then $(\exists x)\varphi$ and $(\forall x)\varphi$ are formulae as well.

A FO(LFP)-definition is a quintuple $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where each predicate $A \in N$ is defined by a formula φ_A with $\dim A$ free variables, and σ is a formula with no free variables that defines the condition of being a syntactically well-formed sentence.

Similarly to a grammar, an FO(LFP)-definition describes a language of well-formed strings. For each string $w \in \Sigma^*$, there is a least assignment of sets of $(\dim A)$ -tuples of positions in w to each predicate A , which satisfies the system of equations $A = \varphi_A$, for all A . This is essentially a generalization of language equations of Ginsburg and Rice [31]. Then, if σ is true under this assignment, the string w is considered well-formed.

Example 8. The grammar in Example 1 is transcribed as the FO(LFP)-definition $G = (\Sigma, \{S\}, \dim, \langle \varphi_S \rangle, \sigma)$, with $\dim S = 2$ and with S defined by the following formula.

$$S(x, y) = \underbrace{[(\exists z)(S(x, z) \wedge S(z, y))] \vee (a(x+1) \wedge S(x+1, y-1) \wedge b(y))}_{\varphi_S} \vee x = y$$

The condition of being a well-formed sentence is $\sigma = S(\text{first}, \text{last})$.

All other grammar families can be similarly expressed in the FO(LFP) logic: conjunctive grammars are represented using the conjunction, multi-component grammars require predicates with more than two arguments.

The following decision procedure for the FO(LFP) logic can be regarded as the mother of all parsing algorithms.

Theorem 1. Let $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$ be an FO(LFP)-definition, let k be the largest dimension of a predicate, let m be the largest number of nested quantifiers in a definition of a predicate. Then there exists an algorithm, which, given an input string $w \in \Sigma^*$ of length n , determines whether w is in $L(G)$, and does so in time $O(n^{2k+m})$, using space $O(n^k)$.

The algorithm calculates the least model by gradually proving all true elementary propositions. There are $O(n^k)$ propositions in total, and at each step, at least one new proposition is proved, which bounds the number of steps by $O(n^k)$. At each step, the algorithm cannot know, which propositions it is already able to prove, so it tries proving each of $O(n^k)$ propositions. Each nested quantifier requires considering n possibilities for the bounded variables, and thus an attempted proof of each proposition requires $O(n^m)$ steps.

Conclusions. All grammar families are representable in $\text{FO}(\text{LFP})$ and have a polynomial-time parsing algorithm provided by Theorem 1. The degree of the polynomial is usually not as good, as provided by the specialized algorithms given in Sect. 7.

9 Equivalent Models

The classical representation of ordinary grammars by nondeterministic pushdown automata [18] gives rise to several related results. First, the LR grammars are similarly characterized by deterministic pushdown automata [28], and linear grammars are characterized by one-turn pushdown automata,

A particularly important special case of pushdown automata are the *input-driven pushdown automata*, also known as *visibly pushdown automata*. This model was known already in 1980; von Braunmühl and Verbeek [14], proved that its deterministic and nondeterministic variants are equal in power. Later, Alur and Madhusudan [3] reintroduced the model under the names “visibly pushdown automata” and “nested word automata”, carried out a systematic study of its properties and inspired further work on the closure properties of input-driven automata and on their descriptive complexity [69].

A generalization of pushdown automata characterizing conjunctive grammars was defined by Aizikowitz and Kaminski [1]. Their model, the *synchronized alternating pushdown automata*, are pushdown automata with a tree-structured stack, with bottom of the stack as the root of the tree and with the top of the stack formed by all the leaves.

Linear conjunctive grammars have an automaton representation of an entirely different kind [59]. They are characterized by the simplest kind of one-dimensional cellular automata: the *one-way real-time cellular automata*, also known as *trellis automata*, studied, in particular, by Ibarra and Kim [36] and by Terrier [82–84]. These automata work in real time, making $n - 1$ parallel steps on an input of length n , and the next value of each cell is determined only by its own value and the value of its right neighbour.

An important alternative representation of ordinary grammars as *categorial grammars* was established by Bar-Hillel et al. [5]. An extension of categorial grammars, the *combinatory categorial grammars*, similarly characterizes tree-adjoining grammars [88]. A different extension augmented with conjunction was introduced by Kuznetsov [48], and its equivalence to conjunctive grammars was proved by Kuznetsov and Okhotin [49].

Conclusions. Representations by pushdown automata and by categorial grammars are among the recurring ideas of formal grammars. A representation by cellular automata has so far been found only for linear conjunctive grammars.

10 Homomorphic Characterizations

The Chomsky–Schützenberger theorem [18] was one of the first theoretical results on grammars. In its original form, it asserts that every language $L \subseteq \Sigma^*$

described by an ordinary grammar is a homomorphic image of an intersection of a Dyck language on k pairs of brackets D_k with a regular language M .

$$L = h(D_k \cap M)$$

The classical proofs of this result rely on using erasing homomorphisms.

There are several stronger forms of this theorem that use non-erasing homomorphisms. The simplest of them assumes that all strings in L are of even length; then it is sufficient to use only symbol-to-symbol homomorphisms.

Theorem 2 (Chomsky and Schützenberger [18]; Okhotin [64]; Crespi-Reghizzi and San Pietro [22]). *For each alphabet Σ , there exists such a number $k \geq 1$, that a language $L \subseteq (\Sigma^2)^*$ is described by an ordinary grammar if and only if there exist a regular language M over an alphabet of k pairs of brackets and a symbol-to-symbol homomorphism h mapping each bracket to Σ , such that $L = h(D_k \cap M)$.*

In plain words, the theorem asserts that if a language is defined by an ordinary grammar, then it is obtained from a nested bracketed structure checked by a finite automaton by *renaming the brackets* to symbols in Σ .

Yoshinaka et al. [91] extended the Chomsky–Schützenberger theorem (in its erasing form) to multi-component grammars, using a suitable generalization of the Dyck language. Salomaa and Soittola [76] and Droste and Vogler [23] established a variant for weighted grammars.

The theorem cannot be extended to conjunctive grammars, regardless of which language would be used instead of the Dyck language, for the reason that every recursively enumerable set is representable as a homomorphic image of a language described by a conjunctive grammar.

11 Hardest Languages

A famous theorem by Greibach [33] states that there exists a fixed language L_0 described by an ordinary grammar G_0 , with the property that every language L over any alphabet Σ that is described by an ordinary grammar G is reducible to L_0 by a homomorphic reduction. In other words, L is representable as an inverse homomorphic image $h^{-1}(L_0)$, for some homomorphism $h: \Sigma \rightarrow \Sigma_0^*$. In the proof, the image $h(a)$ of each symbol $a \in \Sigma$ encodes basically the entire grammar G , and for each string $w \in \Sigma^*$, its image $h(w)$ is defined by the “universal” grammar G_0 if and only if $w \in L$.

This theorem is similar in spirit to the results on the existence of *complete sets* in several complexity classes, such as NP-complete sets. In Greibach’s theorem, a homomorphism is a *reduction function*, cf. polynomial-time reductions in the definitions of NP-complete problems.

For conjunctive grammars and for Boolean grammars, there are hardest language theorems with exactly the same statement [67]. Likely, Greibach’s theorem could also hold for certain classes of multi-component grammars, such as for multi-component grammars of maximum dimension k , for each k .

Turning to special cases of ordinary grammars, Greibach [34] demonstrated that the LR grammars cannot have a hardest language under homomorphic reductions, and Boasson and Nivat [12] proved the same result for linear grammars. One could expect that there is no hardest language for the unambiguous grammars; however, as Boasson and Nivat [12] rightfully remarked, proving that “seems to be a hard problem”. Whether Greibach’s theorem holds for linear conjunctive grammars is another open problem.

Conclusions. Grammar families without any special restrictions, such as determinism, unambiguity or linearity of concatenation, tend to have hardest languages. For various combinations of special restrictions, the existence of hardest languages has either been disproved or remains open.

12 Limitations of Grammars

There are several known methods for proving that a language is not described by any grammar from a certain class.

For ordinary grammars, there is the classical *pumping lemma* of Bar-Hillel et al. [6] that exploits the possibility of inserting repetitive structure into any sufficiently large parse tree. The same idea yields stronger versions of the pumping lemma: *Ogden’s lemma* [55] featuring distinguished positions and the *Bader-Moura lemma* [4] that further allows some positions to be excluded from pumping. There are two special results based on exchanging subtrees between parse trees of different strings: *Sokółowski’s lemma* [79] and the *interchange lemma* by Ogden et al. [56].

In general, the idea behind the pumping lemma equally applies to multi-component grammars: one can also insert repetitive structure into their parse trees. However, the inserted fragments may scatter between the components, and, as demonstrated by Kanazawa et al. [43], a direct analogue of the ordinary pumping lemma does not hold for multi-component grammars of dimension 3 or more; it is only known that *some* string can be pumped, but not necessarily all of them. A standard pumping lemma exists for well-nested multi-component grammars [42].

Parikh’s theorem states that if a language is described by an ordinary grammar, then it can be transformed to a regular language by changing the order of symbols in the strings. Numerous proofs of this theorem are known, including the recent constructive proof by Esparza et al. [25]. Parikh’s theorem directly applies to multi-component grammars, and does not apply to conjunctive grammars, which can, for instance, describe some non-regular unary languages [39, 40].

Negative results for unambiguous grammars can be proved using the pumping lemma and its variants [55], or using the methods of analytic combinatorics [26]: if the generating function for a language is transcendental, it cannot be described by an unambiguous grammar. These methods extend to unambiguous multi-component grammars, but are not applicable to unambiguous conjunctive grammars.

For linear conjunctive grammars, several methods for proving non-representability of languages have been developed using their cellular automaton representation [16, 82, 84]. There are no known methods for proving that there is no conjunctive grammar for a certain language [65], and this remains a substantial gap in the knowledge on that family.

13 Complexity

In the late 1950s and early 1960s, when nothing was yet known about computational complexity, the Chomsky hierarchy by itself served as a pre-historic hierarchy of complexity classes, consisting of $\text{DSPACE}(\text{const})$, grammars, $\text{NSPACE}(n)$ and the recursively enumerable sets. It answered the natural question on the complexity of ordinary grammars by placing them between $\text{DSPACE}(\text{const})$ and $\text{NSPACE}(n)$, and put them in the context of the emerging theoretical computer science. The complexity theory has advanced since that time, and it is important to relate formal grammars to the modern complexity classes, as well as to compare different grammar families according to their complexity. The resulting picture is presented in Fig. 3.

First of all, different grammar families are special cases of the complexity class P due to their representation in the FO(LFP) logic; and linear conjunctive grammars can describe some P -complete languages [36, 65]. At the lower end of the hierarchy, there are families contained in the logarithmic space (L), the largest of them are the LR(1) linear grammars ($LRLin$); also, there exists an LR(1) linear grammar that describes an L -complete language [35]. The whole family of linear grammars (Lin) is similarly contained in NL and can describe an NL -complete language [80].

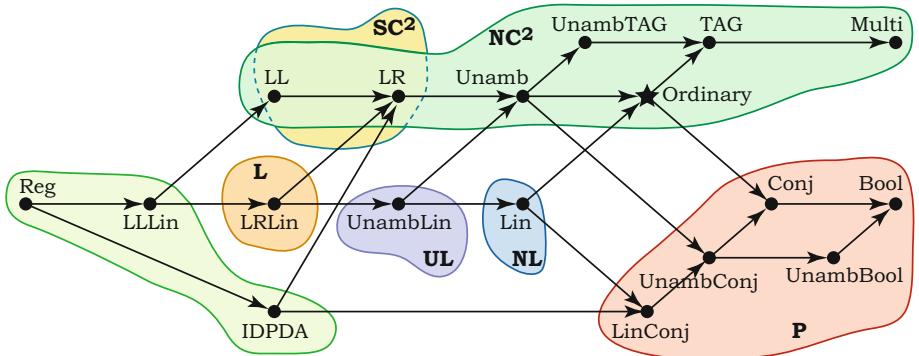


Fig. 3. Hierarchy of grammar families: complexity.

Turning to the complexity of ordinary grammars, in 1979, Cook [21] wrote: “I see no way of showing $\text{DCFL} \subseteq \text{NC}$ ”. Yet, a few years later, Brent and

Goldschlager [15] and Rytter [75], described a circuit of depth $(\log n)^2$ with $O(n^6)$ gates that recognizes the membership of a string in the language in time $O((\log n)^2)$. There was a much earlier algorithm based on the same idea: this was the recognition procedure for ordinary grammars by Lewis, Stearns and Hartmanis [52] that uses only $O((\log n)^2)$ bits of memory, at the expense of super-polynomial running time. The underlying idea of these algorithms is to use an augmented logical system, in which the height of proof trees is logarithmic in the length of a string. The small-space algorithm and the fast parallel algorithm both work by finding a shallow proof in the augmented system.

14 Towards Further Models

Some good grammar families may still remain undiscovered, and it would be interesting to identify such models.

In the early days of formal language theory, when, following Chomsky [17], grammars were regarded as rewriting systems, any new kinds of grammars were defined by modifying the rewriting rules. These attempts usually resulted in models that do not correspond to any intuitive syntactic descriptions. It is unlikely that any useful model can be defined in this way.

Judging by today's knowledge, since other grammar families are naturally expressed in $\text{FO}(\text{LFP})$, there are all reasons to expect some further well-chosen fragments of $\text{FO}(\text{LFP})$ to give rise to interesting models. Likely, an interesting model would express some condition that could be naturally used in informal definitions of syntax, and at the same time would maintain some of the recurring ideas of formal grammars. For instance, *grammars with context operators* [7, 8] were defined by taking the old idea of a rule applicable in a context and expressing it in $\text{FO}(\text{LFP})$.

Instead of using $\text{FO}(\text{LFP})$ as the base model, one could also consider the related logic theories studied in the field of descriptive complexity [38]; perhaps some of them could be useful as a source of inspiration in the search for new grammar families.

References

1. Aizikowitz, T., Kaminski, M.: Conjunctive grammars and alternating pushdown automata. *Acta Informatica* **50**(3), 175–197 (2013). https://doi.org/10.1007/978-3-540-69937-8_6
2. Aizikowitz, T., Kaminski, M.: LR(0) conjunctive grammars and deterministic synchronized alternating pushdown automata. *J. Comput. Syst. Sci.* **82**(8), 1329–1359 (2016). <https://doi.org/10.1016/j.jcss.2016.05.008>
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. *J. ACM* **56**, 3 (2009). <https://doi.org/10.1145/1516512.1516518>
4. Bader, C., Moura, A.: A generalization of Ogden's lemma. *J. ACM* **29**(2), 404–407 (1982). <https://doi.org/10.1145/322307.322315>
5. Bar-Hillel, Y., Gaifman, H., Shamir, E.: On categorial and phrase structure grammars. *Bull. Res. Counc. Israel* **9F**, 155–166 (1960)

6. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase-structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* **14**, 143–177 (1961)
7. Barash, M., Okhotin, A.: An extension of context-free grammars with one-sided context specifications. *Inf. Comput.* **237**, 268–293 (2014). <https://doi.org/10.1016/j.ic.2014.03.003>
8. Barash, M., Okhotin, A.: Two-sided context specifications in formal grammars. *Theoret. Comput. Sci.* **591**, 134–153 (2015). <https://doi.org/10.1016/j.tcs.2015.05.004>
9. Berstel, J., Boasson, L.: Balanced grammars and their languages. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) *Formal and Natural Computing*. LNCS, vol. 2300, pp. 3–25. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45711-9_1
10. Blattner, M., Ginsburg, S.: Position-restricted grammar forms and grammars. *Theoret. Comput. Sci.* **17**(1), 1–27 (1982). [https://doi.org/10.1016/0304-3975\(82\)90128-1](https://doi.org/10.1016/0304-3975(82)90128-1)
11. Blum, N.: More on the power of chain rules in context-free grammars. *Theoret. Comput. Sci.* **27**, 287–295 (1983). [https://doi.org/10.1016/0304-3975\(82\)90122-0](https://doi.org/10.1016/0304-3975(82)90122-0)
12. Boasson, L., Nivat, M.: Le cylindre des langages linéaires. *Math. Syst. Theory* **11**, 147–155 (1977). <https://doi.org/10.1007/BF01768473>
13. Boullier, P.: A cubic time extension of context-free grammars. *Grammars* **3**(2–3), 111–131 (2000). <https://doi.org/10.1023/A:1009907814595>
14. von Braunmühl, B., Verbeek, R.: Input driven languages are recognized in $\log n$ space. *North-Holland Math. Stud.* **102**, 1–19 (1985). [https://doi.org/10.1016/S0304-0208\(08\)73072-X](https://doi.org/10.1016/S0304-0208(08)73072-X)
15. Brent, R.P., Goldschlager, L.M.: A parallel algorithm for context-free parsing. *Aust. Comput. Sci. Commun.* **6**(7), 7.1–7.10 (1984)
16. Buchholz, T., Kutrib, M.: On time computability of functions in one-way cellular automata. *Acta Informatica* **35**(4), 329–352 (1998). <https://doi.org/10.1007/s002360050123>
17. Chomsky, N.: Three models for the description of language. *IRE Trans. Inf. Theory* **2**(3), 113–124 (1956). <https://doi.org/10.1109/TIT.1956.1056813>
18. Chomsky, N., Schützenberger, M.P.: The algebraic theory of context-free languages, In: Bräffort, H. (ed.) *Computer Programming and Formal Systems*, pp. 118–161. North-Holland, Amsterdam (1963). [https://doi.org/10.1016/S0049-237X\(08\)72023-8](https://doi.org/10.1016/S0049-237X(08)72023-8)
19. Chytil, M.P.: Kins of context-free languages. In: Gruska, J., Rovan, B., Wiedermann, J. (eds.) *MFCS 1986*. LNCS, vol. 233, pp. 44–58. Springer, Heidelberg (1986). <https://doi.org/10.1007/BFb0016233>
20. Clark, A., Eyraud, R., Habrard, A.: Using contextual representations to efficiently learn context-freelanguages. *J. Mach. Learn. Res.* **11**, 2707–2744 (2010). <http://www.jmlr.org/papers/v11/clark10a.html>
21. Cook, S.A.: Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In: 11th Annual ACM Symposium on Theory of Computing, (STOC 1979, 30 April–2 May 1979, Atlanta, Georgia, USA), pp. 338–345 (1979). <https://doi.org/10.1145/800135.804426>
22. Crespi Reghizzi, S., San Pietro, P.: The missing case in Chomsky-Schützenberger theorem. In: Didiu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) *LATA 2016*. LNCS, vol. 9618, pp. 345–358. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30000-9_27

23. Droste, M., Vogler, H.: The Chomsky-Schützenberger theorem for quantitative context-free languages. *Int. J. Found. Comput. Sci.* **25**(8), 955–970 (2014). <https://doi.org/10.1142/S0129054114400176>
24. Engelfriet, J.: An elementary proof of double Greibach normal form. *Inf. Process. Lett.* **44**(6), 291–293 (1992). [https://doi.org/10.1016/0020-0190\(92\)90101-Z](https://doi.org/10.1016/0020-0190(92)90101-Z)
25. 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>
26. Flajolet, P.: Analytic models and ambiguity of context-free languages. *Theoret. Comput. Sci.* **49**, 283–309 (1987). [https://doi.org/10.1016/0304-3975\(87\)90011-9](https://doi.org/10.1016/0304-3975(87)90011-9)
27. Floyd, R.W.: Syntactic analysis and operator precedence. *J. ACM* **10**(3), 316–333 (1963). <https://doi.org/10.1145/321172.321179>
28. Ginsburg, S., Greibach, S.A.: Deterministic context-free languages. *Inf. Control* **9**(6), 620–648 (1966). [https://doi.org/10.1016/S0019-9958\(66\)80019-0](https://doi.org/10.1016/S0019-9958(66)80019-0)
29. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. *J. ACM* **14**(2), 389–418 (1967). <https://doi.org/10.1145/321386.321403>
30. Ginsburg, S., Harrison, M.A.: Bracketed context-free languages. *J. Comput. Syst. Sci.* **1**(1), 1–23 (1967). [https://doi.org/10.1016/S0022-0000\(67\)80003-5](https://doi.org/10.1016/S0022-0000(67)80003-5)
31. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. *J. ACM* **9**, 350–371 (1962). <https://doi.org/10.1145/321127.321132>
32. Greibach, S.A.: A new normal-form theorem for context-free phrase structure grammars. *J. ACM* **12**, 42–52 (1965). <https://doi.org/10.1145/321250.321254>
33. Greibach, S.A.: The hardest context-free language. *SIAM J. Comput.* **2**(4), 304–310 (1973). <https://doi.org/10.1137/0202025>
34. Greibach, S.A.: Jump PDA's and hierarchies of deterministic context-free languages. *SIAM J. Comput.* **3**(2), 111–127 (1974). <https://doi.org/10.1137/0203009>
35. Holzer, M., Lange, K.-J.: On the complexities of linear LL(1) and LR(1) grammars. In: Ésik, Z. (ed.) FCT 1993. LNCS, vol. 710, pp. 299–308. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57163-9_25
36. Ibarra, O.H., Kim, S.M.: Characterizations and computational complexity of systolic trellis automata. *Theoret. Comput. Sci.* **29**, 123–153 (1984). [https://doi.org/10.1016/0304-3975\(84\)90015-X](https://doi.org/10.1016/0304-3975(84)90015-X)
37. Immerman, N.: Relational queries computable in polynomial time. *Inf. Control* **68**(1–3), 86–104 (1986). [https://doi.org/10.1016/S0019-9958\(86\)80029-8](https://doi.org/10.1016/S0019-9958(86)80029-8)
38. Immerman, N.: Descriptive Complexity. Springer, New York (1999). <https://doi.org/10.1007/978-1-4612-0539-5>
39. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. *Int. J. Found. Comput. Sci.* **19**(3), 597–615 (2008). <https://doi.org/10.1142/S012905410800584X>
40. Jeż, A., Okhotin, A.: Conjunctive grammars over a unary alphabet: undecidability and unbounded growth. *Theory Comput. Syst.* **46**(1), 27–58 (2010). <https://doi.org/10.1007/s00224-008-9139-5>
41. Jeż, A., Okhotin, A.: Computational completeness of equations over sets of natural numbers. *Inf. Comput.* **237**, 56–94 (2014). <https://doi.org/10.1016/j.ic.2014.05.001>
42. Kanazawa, M.: The pumping lemma for well-nested multiple context-free languages. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 312–325. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02737-6_25
43. Kanazawa, M., Kobele, G.M., Michaelis, J., Salvati, S., Yoshinaka, R.: The failure of the strong pumping lemma for multiple context-free languages. *Theory Comput. Syst.* **55**(1), 250–278 (2014). <https://doi.org/10.1007/s00224-014-9534-z>

44. Kelemenová, A.: Complexity of normal form grammars. *Theoret. Comput. Sci.* **28**(3), 299–314 (1983). [https://doi.org/10.1016/0304-3975\(83\)90026-9](https://doi.org/10.1016/0304-3975(83)90026-9)
45. Kountouriotis, V., Nomikos, C., Rondogiannis, P.: Well-founded semantics for Boolean grammars. *Inf. Comput.* **207**(9), 945–967 (2009). <https://doi.org/10.1016/j.ic.2009.05.002>
46. Kowalski, R.: Logic for Problem Solving. North-Holland, Amsterdam (1979)
47. Kunc, M.: The power of commuting with finite sets of words. *Theory Comput. Syst.* **40**(4), 521–551 (2007). <https://doi.org/10.1007/s00224-006-1321-z>
48. Kuznetsov, S.: Conjunctive grammars in Greibach normal form and the Lambek calculus with additive connectives. In: Morrill, G., Nederhof, M.-J. (eds.) FG 2012–2013. LNCS, vol. 8036, pp. 242–249. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39998-5_15
49. Kuznetsov, S., Okhotin, A.: Conjunctive categorial grammars. In: Proceedings of 15th Meeting on the Mathematics of Language (MOL 2017, London, UK, 13–14 July 2017), pp. 141–151. ACL (2017)
50. Lange, M.: Alternating context-free languages and linear time μ -calculus with sequential composition. *Electron. Notes Theoret. Comput. Sci.* **68**(2), 70–86 (2002). [https://doi.org/10.1016/S1571-0661\(05\)80365-2](https://doi.org/10.1016/S1571-0661(05)80365-2)
51. Lehtinen, T., Okhotin, A.: Boolean grammars and GSM mappings. *Int. J. Found. Comput. Sci.* **21**(5), 799–815 (2010). <https://doi.org/10.1142/S0129054110007568>
52. Lewis II, P.M., Stearns, R.E., Hartmanis, J.: Memory bounds for recognition of context-free and context-sensitive languages. In: IEEE Conference Record on Switching Circuit Theory and Logical Design, pp. 191–202 (1965). <https://doi.org/10.1109/FOCS.1965.14>
53. McNaughton, R.: Parenthesis grammars. *J. ACM* **14**(3), 490–500 (1967). <https://doi.org/10.1145/321406.321411>
54. Nakanishi, R., Takada, K., Nii, H., Seki, H.: Efficient recognition algorithms for parallel multiple context-free languages and for multiple context-free languages. *IEICE Trans. Inf. Syst.* **E81-D:11**, 1148–1161 (1998)
55. Ogden, W.F.: A helpful result for proving inherent ambiguity. *Math. Syst. Theory* **2**(3), 191–194 (1968). <https://doi.org/10.1007/BF01694004>
56. Ogden, W., Ross, R.J., Winklmann, K.: An ‘interchange lemma’ for context-free languages. *SIAM J. Comput.* **14**(2), 410–415 (1985). <https://doi.org/10.1137/0214031>
57. Okhotin, A.: Conjunctive grammars. *J. Automata Lang. Comb.* **6**(4), 519–535 (2001)
58. Okhotin, A.: Boolean grammars. *Inf. Comput.* **194**(1), 19–48 (2004). <https://doi.org/10.1016/j.ic.2004.03.006>
59. Okhotin, A.: On the equivalence of linear conjunctive grammars to trellis automata. *Informatique Théorique et Applications* **38**(1), 69–88 (2004). <https://doi.org/10.1051/ita:2004004>
60. Okhotin, A.: Unresolved systems of language equations: expressive power and decision problems. *Theoret. Comput. Sci.* **349**(3), 283–308 (2005). <https://doi.org/10.1016/j.tcs.2005.07.037>
61. Okhotin, A.: Recursive descent parsing for Boolean grammars. *Acta Informatica* **44**(3–4), 167–189 (2007). <https://doi.org/10.1007/s00236-007-0045-0>
62. Okhotin, A.: Unambiguous Boolean grammars. *Inf. Comput.* **206**, 1234–1247 (2008). <https://doi.org/10.1016/j.ic.2008.03.023>
63. Okhotin, A.: Decision problems for language equations. *J. Comput. Syst. Sci.* **76**(3–4), 251–266 (2010). <https://doi.org/10.1016/j.jcss.2009.08.002>

64. Okhotin, A.: Non-erasing variants of the Chomsky–Schützenberger theorem. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 121–129. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31653-1_12
65. Okhotin, A.: Conjunctive and Boolean grammars: the true general case of the context-free grammars. Comput. Sci. Rev. **9**, 27–59 (2013). <https://doi.org/10.1016/j.cosrev.2013.06.001>
66. Okhotin, A.: Parsing by matrix multiplication generalized to Boolean grammars. Theoret. Comput. Sci. **516**, 101–120 (2014). <https://doi.org/10.1016/j.tcs.2013.09.011>
67. Okhotin, A.: The hardest language for conjunctive grammars. In: Kulikov, A.S., Woeginger, G.J. (eds.) CSR 2016. LNCS, vol. 9691, pp. 340–351. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34171-2_24
68. Okhotin, A., Reitwießner, C.: Conjunctive grammars with restricted disjunction. Theoret. Comput. Sci. **411**(26–28), 2559–2571 (2010). <https://doi.org/10.1016/j.tcs.2010.03.015>
69. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News **45**(2), 47–67 (2014). <https://doi.org/10.1145/2636805.2636821>
70. Pereira, F.C.N., Warren, D.H.D.: Parsing as deduction. In: 21st Annual Meeting of the Association for Computational Linguistics (ACL 1983, Cambridge, Massachusetts, USA, 15–17 June 1983), pp. 137–144 (1983)
71. Pollard, C.J.: Generalized phrase structure grammars, head grammars, and natural language. Ph.D. thesis, Stanford University (1984)
72. Rajasekaran, S., Yooseph, S.: TAL recognition in $O(M(n^2))$ time. J. Comput. Syst. Sci. **56**(1), 83–89 (1998). <https://doi.org/10.1006/jcss.1997.1537>
73. Rosenkrantz, D.J.: Matrix equations and normal forms for context-free grammars. J. ACM **14**(3), 501–507 (1967). <https://doi.org/10.1145/321406.321412>
74. Rounds, W.C.: LFP: a logic for linguistic descriptions and an analysis of its complexity. Comput. Linguist. **14**(4), 1–9 (1988)
75. Rytter, W.: On the recognition of context-free languages. In: Skowron, A. (ed.) SCT 1984. LNCS, vol. 208, pp. 318–325. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-16066-3_26
76. 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>
77. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoret. Comput. Sci. **88**(2), 191–229 (1991). [https://doi.org/10.1016/0304-3975\(91\)90374-B](https://doi.org/10.1016/0304-3975(91)90374-B)
78. Sipser, M.: Introduction to the Theory of Computation, 3rd edn. Cengage Learning, Boston (2012)
79. Sokołowski, S.: A method for proving programming languages non context-free. Inf. Process. Lett. **7**(2), 151–153 (1978). [https://doi.org/10.1016/0020-0190\(78\)90080-7](https://doi.org/10.1016/0020-0190(78)90080-7)
80. Sudborough, I.H.: A note on tape-bounded complexity classes and linear context-free languages. J. ACM **22**(4), 499–500 (1975). <https://doi.org/10.1145/321906.321913>
81. Szabari, A.: Alternujúce Zásobníkové Automaty (Alternating Pushdown Automata). M.Sc. thesis, 45 p. Diploma work, University of Košice (Czechoslovakia) (1991). (in Slovak)
82. Terrier, V.: On real-time one-way cellular array. Theoret. Comput. Sci. **141**(1–2), 331–335 (1995). [https://doi.org/10.1016/0304-3975\(94\)00212-2](https://doi.org/10.1016/0304-3975(94)00212-2)
83. Terrier, V.: Recognition of poly-slender context-free languages by trellis automata. Theoret. Comput. Sci. **692**, 1–24 (2017). <https://doi.org/10.1016/j.tcs.2017.05.041>

84. Terrier, V.: Some computational limits of trellis automata. In: Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E. (eds.) AUTOMATA 2017. LNCS, vol. 10248, pp. 176–186. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58631-1_14
85. Urbanek, F.J.: On Greibach normal form construction. *Theoret. Comput. Sci.* **40**, 315–317 (1985). [https://doi.org/10.1016/0304-3975\(85\)90173-2](https://doi.org/10.1016/0304-3975(85)90173-2)
86. Valiant, L.G.: General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.* **10**(2), 308–314 (1975). [https://doi.org/10.1016/S0022-0000\(75\)80046-8](https://doi.org/10.1016/S0022-0000(75)80046-8)
87. Vardi, M.Y.: The complexity of relational query languages. In: STOC 1982, pp. 137–146 (1982). <https://doi.org/10.1109/SFCS.1981.18>
88. Vijay-Shanker, K., Weir, D.J.: The equivalence of four extensions of context-free grammars. *Math. Syst. Theory* **27**(6), 511–546 (1994). <https://doi.org/10.1007/BF01191624>
89. Vijay-Shanker, K., Weir, D.J., Joshi, A.K.: Characterizing structural descriptions produced by various grammatical formalisms. In: 25th Annual Meeting of Association for Computational Linguistics (ACL 1987), pp. 104–111 (1987)
90. Yoshinaka, R.: Distributional learning of conjunctive grammars and contextual binary feature grammars. *J. Comput. Syst. Sci.* (to appear). <https://doi.org/10.1016/j.jcss.2017.07.004>
91. Yoshinaka, R., Kaji, Y., Seki, H.: Chomsky-Schützenberger-type characterization of multiple context-free languages. In: Dediū, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 596–607. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13089-2_50



Reshaping the Context-Free Model: Linguistic and Algorithmic Aspects

Eli Shamir^(✉)

Hebrew University of Jerusalem, Jerusalem, Israel
shamir@cs.huji.ac.il

1 Introduction

The CF formal grammar model was twin-born and raised, 1956–1963 [[C](#), [CS](#), [BGS](#), [BPS](#)].

- (1) As the second in Chomsky hierarchy of models for syntactic phrase-structure in natural languages sentences.
- (2) As “Backus Normal Form” - to model syntax of programming languages like ALGOL 60.

The research of the twins soon diverged. Compilers built for translating programming languages to assembly languages - developed submodels, like $LR(k)$ which avoided ambiguity and were amenable to deterministic parsing.

The Linguist twin was worried by inadequacies, syntactic constructs which seem beyond the CF power. This led to extensions. “Mild context-sensitive” models and beyond. The tree-adjunction grammar [TAG] is the most popular. It received a fervent linguistic defence in Frank’s book [[F](#)].

However, the formal study of the extensions evolved toward “stringology”: Algebraic and algorithmic study of recursive-iterative mechanisms to generate infinite sets of strings, trees or other designs and how to parse the yields of those mechanisms. Kallmeyer’s book [[K](#)] gives an extensive account of this research.

But if the primary goal of the model is computerized parsing of natural languages texts, it should be comprehensive and efficient. Conforming with linguistic theories and over-generation - are secondary issues.

The reshape proposal goes in this direction. It stays away from extensions which increase the recursive power. Even within CFG, a “thinning diet” cuts away some iterative rules which create bad ambiguities, increase parsing complexity and seem syntactically unuseful.

The proposal is presented and reasoned in Sects. 4 and 5, which can be read right here. Its parsing is described in Sect. 6. Sections 2 and 3 are overviews of the CF model and its deficiencies: The universal and hardest CFG/CFL, its extreme non-determinism and inherent ambiguities, indicating how to ameliorate them toward a reshaped model.

2 Overview of CFG/L: Universal, Hardest, Extreme Ambiguity

Familiarity with basic CFG definitions and notations is assumed. The presentation follows the extensive survey [ABB].

Definition 2.1. The DYKE set of strings in $\{A_j, \bar{A}_j, j \in J\}^*$ is the subset of strings which cancel to e (the null string) under repeated cancellations $A_j \bar{A}_j = e$.

Remark 2.2. For arithmetic- and logic-expressions, with operators having two (independent) arguments, using infix notation, DYKE provides pairs of brackets. These are used to isolate substrings which serve as arguments, thus avoiding ambiguity in parsing. DYKE serves a similar role in CFG parsing (by PDA) to denote (start, end) of applying production $A \rightarrow \alpha$. But ambiguity creeps in the multiple choice: which production to use next. Thus, Shamir theorem [S1] characterizes each CFL by a homomorphism into **subsets** of DYKE. Its proof in [HK, ABB] is simplified via Greibach normal form for productions in G

$$A_j \rightarrow a\alpha_j, 1 \leq j \leq k, \text{ for } a \in T, A_j \in NT, \alpha_j \in (NT)^*$$

Now Φ on T^* is defined by the monomials

$$(2.3) \quad \Phi(a) = [\bar{A}, \alpha^R + \cdots + \bar{A}_k \alpha_k^R], \alpha^R = \text{reverse of } \alpha$$

$$(2.4) \quad \Phi(w = a_1 \dots a_r) = \Phi(a_1) \dots \Phi(a_r) ([, +,] \text{extra symbols})$$

Using the distributive law, this product in (2.4) opens to a sum of terms $\sum_\ell \beta_\ell. \beta_\ell \in \{A_j, \bar{A}_j\}^*$ and finally

$$(2.5) \quad w \in L_S(G) = \{w \in T^*, S \xrightarrow{*} w \in G\} \\ \text{iff } S \cdot \beta_\ell \in DYKE\{A_j, \bar{A}_j\} \text{ for some } \ell.$$

The proof in [ABB, HK] is by simple induction. Its idea lies in the pushdown [PDA] simulation of a derivation in G . Upon reading the terminal a , some \bar{A}_j, α_j^R is chosen to push into the stack, or if $A_j \rightarrow a$ is chosen, then \bar{A}_j is popped (corresponds to $A_j \bar{A}_j = e$). Acceptance is by empty PDA stack.

Clearly, a choice of the good productions leading to an empty stack is an extremely non-deterministic process, but also very handy in verifying a candidate language to be CFL.

A universal CFG/L is obtained by taking in 2.3, 2.4 all possible productions $A \rightarrow a\alpha$. It embeds all CFGs (up to size K), hence inherits all undesirable properties, unbounded inherent ambiguities, and more. Greibach [GR2] observed it is also the hardest CFG/L to parse.

Remark 2.6. The proof in [S1] relied on homomorphism [BGS] into subset of “category symbols” with different cancellation rules, and reduced it to DYKE-cancellations, which is simpler. \square

3 Inherent Ambiguities in CLF

Ginsburg's book [G] has a chapter on inherent ambiguities in CFLs contained in $a_1^* \dots a_n^*$. More realistic is the inherently unbounded ambiguity in L_1 (u^R asks: where is my partner u)

$$(3.1) \quad L_1 = HuH' \$ u^R, \quad u, H, H' = \{a_1 b\}^*$$

u^R is the reverse of u . L_1 has a linear CFG. First derive H' , then switch to derive the dependency $u \longrightarrow \$ u^R$, finally switch to derive H' . The ambiguity results from the non-deterministic choice when to switch. This is a **rhythrical** non-determinism/ambiguity, which appears also in NDFA. Intuitively it is unbounded and inherent - formally proved in [S2]. Consider now

$$(3.2) \quad L_2 = N \cdot M = u^R \$ Hu \cdot v H' \$ v^R$$

the factors N, M have unambiguous linear grammars. But an answer to the question: where is the dot between N and M ? creates unbounded (and inherent) **product ambiguity**:

$$(3.3) \quad X \rightarrow YZ, \quad Y \xrightarrow{*} u, \quad Y \xrightarrow{*} u', \quad Z \xrightarrow{*} v, \quad Z \xrightarrow{*} v', \quad uv = u'v'.$$

This type of ambiguity is more serious. In Earley parsing algorithm, unbounded product ambiguity in G raises the parsing time-complexity to cubic, $O(n^3)$.

Consider now the transformation $NM \diagdown N^\wedge M^\#$ called "top trunk rotation" - [TTR]. It takes the top trunk of N , creating the dependency $u^R - u$ rotates it by 180° and mounts it on M (formal definition in (3.5)). It is seen that the yield of NM is cyclic rotated; for example:

$$(3.4) \quad abb \$ Hbba dcd H' \$ ccd \diagdown \$ Hbbadcc H' \$ ccd abb$$

Now $N^\wedge M^\#$ is linear and essentially the same as L_1 above:

$$u \cdot v \text{ between the two \$ signs and } (u \cdot v)^R \text{ on the right.}$$

Moreover, the ambiguous D -trees are mapped, in 1–1 onto manner, from product type to rhythmical type.

Remark 3.5. Linear CFGs and their substitution closure class (= non expansive grammars class) [ABB, S4] are free of product ambiguity.

Definition 3.6. TTR transforms $N.M$ to $N^\wedge M^\#$ where N, M are grammars with disjoint non-terminals, root (N) = B is a pump symbol. Hence each D -tree of N starts with the $[B]$ trunk. It is rotated by 180° and mounted on M , by the corresponding rotation of the (binary) production of the trunk in N upon their transfer to $M^\#$:

in $N B' \rightarrow CB''$ rotates to $B'' \rightarrow CB'$ in $M^\#$
 in $N B' \rightarrow B''C$ rotates to $B'' \rightarrow B'C$ in $M^\#$

The cyclic rotation of the yield, as shown in (3.4) is readily verified.

4 The Reshaped Grammar - Formal Presentation

The reshaped grammar G consists of

- I. The kernel - a bounded bunch of linear grammars $G(i)$, sharing NT symbols and pre-terminal symbols

Definition 4.1. Symbol G if preterminal if it is a root of a CF grammar subs (C) which derives only D -trees without iterations along their branches - hence only a bounded number of bounded trees, whose yield are terminal strings.

- II. Compiled substitution banks, subs (C) for each preterminal C as described in (4.1)
- III. Compiled Adjunction Lists - list of substrings $u - u'$ of bounded size which are legally possible in terminal strings of G .

A typical example is cross-serial dependency $u - \varphi(u)$ (cf. 6.4). The concept comes from TAG, which enables insertion of subtrees around an inner node of an existing D -tree. It is beyond CF power. But if it occurs locally in a bounded section of a string, then a complicated (ugly) CFG could generate it. Integrating it and (subs (C) as well) into the CYK tabular parsing algorithm in Sect. 6, is simple to describe and much more efficient to parse.

Back to the kernel: In a linear $G(i)$ grammar (WLOG, its productions are binary or unary), each D -tree has a single main trunk, labelled by NT symbols and length 1 branches on both sides of the trunk, labeled by **preterminal symbols** (if all branches are on one side, it is a finite automaton).

The main trunks are like towers with several floors. Each floor hosts one pump equivalence class $[B]$.

$$(4.2) \quad B' \in \text{pump class } [B] \text{ if } B \xrightarrow{*} -B' - \text{ and } B' \xrightarrow{*} -B - .$$

In addition few symbols may appear as “stairs” connecting the floors.

The recursive iteration in the kernel can occur only inside the pump classes. As in example (3.1), the only non-determinism is rhythmic, when to exit a floor downward. The substring (of pre-terminals) derived by a pump class is embedded, like a subordinate clause in the substring generated above it. Embedding in the opposite direction is also valid by syntax rules. So several $G(i)$ in the kernels with various orders of the pump classes along the trunks are desired. Inversion of orders is obtained when TTR is applied to product $N \cdot M$ of linear grammars as in Sect. 3 (continued TTRs on N^\wedge until it is fully mounted on the other factor). Thus the class of CFG which admit decomposition to union of linear grammars (provided the yields are viewed as circular string) is closed under union and products.

5 The Reshaped Grammar-Linguistic Aspects

The proposed reshaped grammar is designed primarily for machines, for computerized parsing aided by completed repositories. It should not be evaluated as a formal generation model. E.g., some over-generation is acceptable.

The kernel is designed to contain all the recursive-iterative abilities of the model, and restrict them to be “vertical” inside the pump-classes of the linear D -trees. Derivation of (sideways) expansive grammars [AB_B], like $S \xrightarrow{*} -S-S-$ are out. Repeating them will crowd many unrelated S -categories (e.g., sentence category) onto one sentence, which creates bad ambiguities and complicated parsing, while the option of using a paragraph with several sentences is available and preferable. The linear trunks also provide embedding of one clause as subordinate to another (in a “higher floor”). The multi-linear character of the kernels enable the embeddings in various orders. Thus transformation grammar mechanism (like active to passive) which inverts the direction of embeddings are enabled.

The repositories of substitution and adjunction describe local sequences of fine-grained parts of speech in sentences (e.g. noun, verb, adjective, ...) which are descriptors of lexical items in dictionaries. These are small repositories compiled by linguists, unlike the huge corpora of parsed sentences (like Penn).

The kernel and substitution banks seem ubiquitous across most natural languages, which conforms to Chomsky’s belief in innate syntax faculty in humans. The adjunction lists are more custom-made, to the peculiarities of each language, designed to settle most of the inadequacies of the CF model, in bounded sentences, not in the asymptotic unbounded limits.

Remark 5.1. Viewing sentences and clauses as circular strings helps to resolve ambiguities. Consider the politically loaded sentence

The policeman shot \\$ the boy# with the gun

The ambiguity (who held the gun) is resolved both ways by the cyclic rotations starting at \$, #.

6 The Parsing Algorithm

The terminology follows Chap. 3 in [K]. A parsing scheme is presented as a collection of deduction rules which have antecedent items (“top”, “above the line”), and one consequent item (“bottom”, “below the line”), and side conditions which determine the validity of the deduction.

These rules support a bottom-up dynamic programming algorithm for deciding membership in $L_A(G)$ of a candidate circular string $*w$ of length n . Then, top-down, it finds a parse for the derivations $A \xrightarrow{*} *w$.

The algorithm here, for the reshaped grammar, is an adaptation of CYK for linear CFG. Items are of the form (A, i, j) signifying the derivation status of $w(i+1) \dots w(j)$, a cyclic substring of $*w$ of width k . Linearity implies that each

deduction has a single top item. Items (A, i, j) are placed in entry (i, j) of $n \times n$ table. Those of width k form the k -diagonal. The 1-diagonal contains the axioms (empty top)

$$(6.1) \quad \frac{\cdot}{(B, i, i+1)} \text{ if } B \rightarrow w(i+1) \text{ is a production in } G.$$

The other deductions come in two groups:

I **Substitutions** for the yields $Y(C)$ of preterminal C .

$$(6.2) \quad \frac{(B', i, \ell)}{(B, i, j)} \text{ if } B \rightarrow B'C \text{ in } G \text{ and } w(\ell+1) \dots, w(j) \in Y(C)$$

$$(6.3) \quad \frac{(B', \ell, j)}{(B, i, j)} \text{ if } B \rightarrow CB' \text{ in } G \text{ and } w(i+1) \dots, w(\ell) \in Y(C)$$

namely, a substring from $Y(C)$ found in w complements the ‘top’ substring from right or left, respectively.

II **Adjunction** permitted by a dependency LIST.

$$(6.4) \quad \frac{(B, i, i+r+s)}{(B, i, i+r+s+t)} \text{ if } (\alpha, \gamma) \in \text{LIST}$$

$$\alpha = w(i+1) \dots w(i+r)$$

$$\beta = w(i+r+1, \dots, w(i+r+s)),$$

$$\gamma = w(i+r+s+1, \dots, w(i+r+s+t))$$

namely $\alpha\beta\gamma$ is a substring of w with $\alpha\beta$ in the top item and the newly read γ complements it, since $(\alpha, \gamma) \in \text{LIST}$. This is a dependency across β . A typical one is a φ -copy $(\alpha, \varphi(\alpha))$. This is a bounded version of (linear) adjunction. In TAG, adjunction are more versatile and unbounded, e.g., the copy language $\alpha\alpha$ [K, p. 27].

Each validated deduction in the table is recorded by a backward arrow from the bottom to the top item. Successful termination happens when an item of width n $(i, i+n \bmod n)$ is reached. It signifies that some cyclic version of the circular $*w$ is derivable using production of G_A for substitution, and also adjunction steps (which do not change the category B). The proof is the same as for CYK.

Finally, tracing some path of backward arrows from a validated n -width item down to the axioms provides a parsing from $*w$, (there may be several in case of ambiguity).

Complexity: Each bottom item of width m has a bounded number of top items which may validate it, and they are in diagonals of width $\geq m-k$, so only those are kept in the work-memory which is $O(n)$, and the time for each item is $O(1)$, so total run-time $O(n^2)$. So, up to a constant it’s like parsing linear grammar by CYK!

Remark 6.5. The same table can be used for all the linear grammar $G(i)$ in the kernel (the skeleton) - simply color the backward arrows by distinct colors for the $G(i)$ s.

Remark 6.6. The multi-linear kernel is like making group of machines to work in parallel on the same data. This proved very useful in several complex computation problems. Also, the linear bound on work-space of the parsing algorithm may make it useful for algorithms designed to study DNA and RNA structures [JP].

Concluding remark: The proposed reshaped grammar was not implemented. It is a gedanken experiment. Hopefully it will have impact on implementations. Quoting the opening lines of Shakespeare's sonnet 116, with a small twist, Let me not to the marriage of minds and machine admit impediments.

Acronyms

- CFG - context-free grammar
- CFL - context-free language
- PDA - pushdown automaton
- NDFA - Non-deterministic finite automaton
- TAG - Tree adjoining grammar
- D-tree - derivation tree
- NT - Non terminal (symbols)
- TTR - Top trunk rotations
- $A \xrightarrow{*} -B-, \exists u, v, A \xrightarrow{*} uBv$
- $Y(c_1)$ - yield of a D -tree with root C
- F^* - $\bigcup_{j=0}^{\infty} F^j$
- w^R - the string w , reversed
- $*w$ - w considered as a circular string
- NLP - Natural language processing
- WLOG - without loss of generality

References

- [ABB] Autebert, J.M., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 111–174. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_3. Chap. 3
- [BGS] Bar-Hillel, Y., Gaifman, H., Shamir, E.: On categorical and phrase structure grammars. Bull. Res. Coun. Isr. **9F**, 1–16 (1960)
- [BPS] Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Zeitsch. Phonetick Spracgwiss. Kommunikationsforschung **14**, 143–172 (1961)
- [C] Chomsky, N.: Three models for the description of language. IRE Trans. Inf. Theory **IT-2**, 113–124 (1956)

- [CS] Chomsky, N., Schützenberger, M.P.: The algebraic theory of context-free languages. In: Brattart, P., Hirschberg, S. (eds.) Computer Programming and Formal Systems, pp. 116–121. North-Holland, Amsterdam (1970)
- [F] Frank, R.: Phrase Structure Composition and Syntactic Dependencies. MIT Press, Cambridge (2002)
- [G] Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw Hill, New York (1966)
- [GR1] Greibach, S.: A new normal for theorem for context-free phrase structure grammars. *J. Assoc. Comput. Math.* **12**, 42–52 (1965)
- [GR2] Greibach, S.: The hardest context-free language. *SIAM J. Comput.* **3**, 304–310 (1973)
- [GJ] Grune, D., Jacobs, C.J.H.: Parsing as intersection. In: Grune, D., Jacobs, C.J.H. (eds.) Parsing Techniques: A Practical Guide. MCS, pp. 425–442. Springer, New York (2008). https://doi.org/10.1007/978-0-387-68954-8_13
- [HU] Hopcroft, J., Ullman, J.: Formal Languages and Their Relation to Automata. Addison-Wesley, Boston (1969)
- [HK] Hotz, G., Kretschmer, T.: The power of Greibach norm form. *Electron. Infor. Kybernet.* **25**, 507–512 (1989)
- [JP] Jones, N.C., Pevzner, P.A.: Introduction to Bioinformatics Algorithms. MIT Press, Cambridge (2004)
- [K] Kallmeyer, L.: Parsing Beyond Context Free Grammars. Springer, Heidelberg (2010). <https://doi.org/10.1007/978-3-642-14846-0>
- [S1] Shamir, E.: A representation theorem for algebraic and context-free power series in noncommuting variables. *Inf. Comput.* **11**, 239–254 (1967)
- [S2] Shamir, E.: Some inherently ambiguous context-free languages. *Inf. Control* **18**, 355–363 (1971)
- [S3] Shamir, E.: Pumping, shrinking and pronouns: from context free to indexed grammars. In: Dediū, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 516–522. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37064-9_45
- [S4] Shamir, E.: Decomposition and parsing of non-expansive context-free grammars (2017, manuscript)



Formal Languages over GF(2)

Ekaterina Bakinova¹, Artem Basharin², Igor Batmanov³, Konstantin Lyubort⁴,
Alexander Okhotin⁵ and Elizaveta Sazhneva⁵

¹ Gymnasium №1, ul. Abelmana, 15, Kovrov 601900, Vladimir Region, Russia
katya-bakinova@mail.ru

² School №179, Bolshaya Dmitrovka, 5/6 building 7, Moscow 125009, Russia
artemvit@bk.ru

³ Moscow Institute of Physics and Technology, Dolgoprudny, Russia
igorbat99@gmail.com

⁴ St. Petersburg Academic University,
ul. Khlopina, 8, Saint Petersburg 194021, Russia
lyubortk@gmail.com

⁵ St. Petersburg State University, 7/9 Universitetskaya nab.,
Saint Petersburg 199034, Russia
alexander.okhotin@spbu.ru, sazhneva-eliza@ya.ru

Abstract. Variants of the union and concatenation operations on formal languages are investigated, in which Boolean logic in the definitions (that is, conjunction and disjunction) is replaced with the operations in the two-element field GF(2) (conjunction and exclusive OR). Union is thus replaced with symmetric difference, whereas concatenation gives rise to a new GF(2)-concatenation operation, which is notable for being invertible. All operations preserve regularity, and their state complexity is determined. Next, a new class of formal grammars based on GF(2)-operations is defined, and it is shown to have the same computational complexity as ordinary grammars with union and concatenation.

1 Introduction

The classical operations on formal languages are union and concatenation; both regular expressions and formal grammars are based on these operations. Union and concatenation are defined in terms of Boolean logic: the union $K \cup L$ is the set of all strings w with $w \in K$ or $w \in L$, which is a disjunction of two conditions; similarly, the membership of a string w in a concatenation $K \cdot L$ is a disjunction, over all partitions $w = uv$, of the conjunction of $u \in K$ and $v \in L$.

$$K \cdot L = \{ w \mid \# \text{ of partitions } w = uv, \text{ with } u \in K \text{ and } v \in L, \text{ is non-zero} \}$$

The purpose of this paper is to investigate a pair of related operations on languages defined using the *exclusive OR* instead of the disjunction, so that Boolean

This research was carried out as a summer project at the Sirius Education Centre, Sochi, Russia. Elizaveta Sazhneva was supported by “Native Towns”, a social investment programme of PJSC “Gazprom Neft”.

logic is effectively replaced with the two-element field GF(2). Thus, 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 interest in these operations is motivated by the fact that they generalize the two unambiguous operations on formal languages: the *disjoint union* and the *unambiguous concatenation*, that is, concatenation $K \cdot L$ with the restriction that each string $w \in K \cdot L$ has a unique factorization $w = uv$ with $u \in K$ and $v \in L$. Indeed, as long as two languages are disjoint, their symmetric difference equals their union, and if the concatenation of K and L is unambiguous, the GF(2)-concatenation $K \odot L$ coincides with the standard concatenation $K \cdot L$. The unambiguous operations are important, for instance, for being the only operations expressible in *unambiguous grammars* and their practically valuable subclasses with efficient parsing algorithms, such as the LL and the LR grammars. Accordingly, any model featuring the GF(2)-operations has the corresponding unambiguous model as a special case.

The two proposed operations can be equally obtained by regarding K and L as formal power series in non-commuting variables with coefficients in GF(2); then, their sum is $K \Delta L$ and their product is $K \odot L$. However, the authors' intention is to treat them as operations on standard formal languages, and to carry out a usual language-theoretic study of these operations.

The basic algebraic properties of GF(2)-operations, listed in Sect. 2, follow from the known facts on formal power series: formal languages form a ring with the symmetric difference as addition and with GF(2)-concatenation as multiplication. Furthermore, every language containing the empty string is an invertible element, which gives a somewhat unexpected property of *inverting the concatenation*, and presents another operation to study.

Closure properties of standard language families under GF(2)-concatenation and GF(2)-inverse are investigated in Sect. 3. It is shown that the family of regular languages is closed under both operations. This is proved by direct constructions on finite automata, which are shown to be optimal with respect to the number of states. The context-free languages and their basic subclasses are predictably not closed under any of the GF(2)-operations.

The next subject is a new family of formal grammars based on GF(2)-operations. The semantics of ordinary grammars with union and concatenation (Chomsky's “context-free”) are defined by least solutions of language equations [4], owing to the monotonicity of both union and concatenation. Since GF(2)-operations are not monotone, such a definition is not applicable to the desired new class of grammars. The definition given in Sect. 4 restricts a grammar, so that each string may have only finitely many parses, and under this restriction, language equations always have a solution.

In the last Sect. 5, the computational complexity of the new class of *GF(2)-grammars* is investigated. First, it is proved that every such grammar can be transformed to an analogue of the Chomsky normal form, leading to simple

variants of the Cocke–Kasami–Younger and Valiant’s parsing algorithms; the latter algorithm is particularly practical over GF(2). The NC² parallel parsing algorithm by Brent and Goldschlager [2] and by Rytter [12] is also adapted to handle GF(2)-grammars, thus establishing the same complexity upper bound for these grammars as for the ordinary grammars with union and concatenation. The uniform membership problem for GF(2)-grammars is shown to be P-complete, whereas for the subclass of *linear GF(2)-grammars*, this problem naturally turns out to be $\oplus L$ -complete.

2 GF(2)-Operations and Their Basic Properties

Two binary operations on formal languages are considered: the symmetric difference $K \Delta L$ and the GF(2)-concatenation $K \odot L$. The latter language consists of all strings w that have *an odd number of partitions* $w = uv$, with $u \in K$ and $v \in L$ (cf. *non-zero number of partitions* for the classical concatenation). By definition, $K \odot L \subseteq K \cdot L$, and if the concatenation is unambiguous, then these two languages coincide.

Example 1. $\{\varepsilon, a\} \odot \{\varepsilon, a\} = \{\varepsilon, aa\}$. The string a is missing from the GF(2)-concatenation, because its two factorizations cancel each other.

Elementary algebraic properties of GF(2)-concatenation and symmetric difference on formal languages follow from the general results on the representation of languages as formal power series with coefficients from an arbitrary semiring, with GF(2) used as the semiring of coefficients. In particular, both the symmetric difference and the GF(2)-concatenation are associative and have identities \emptyset and $\{\varepsilon\}$, respectively; furthermore, the symmetric difference is invertible by complementation, as $L \Delta \overline{L} = \emptyset$; the two operations are distributive.

Proposition 1. *For every alphabet Σ , the set of all languages 2^{Σ^*} forms a ring, with symmetric difference as sum and with GF(2)-concatenation as product.*

A further interesting property is that some languages have *inverses* with respect to GF(2)-concatenation.

Example 2. $\{\varepsilon, ab\} \odot (ab)^* = \{\varepsilon\}$.

No language L with $\varepsilon \notin L$ may have an inverse, because in this case $\varepsilon \notin K \odot L$ for any language K . On the other hand, every language containing the empty string has a GF(2)-inverse.

Theorem 1. *For every language $L \subseteq \Sigma^*$ with $\varepsilon \in L$, there exists a unique language $L^{-1} \subseteq \Sigma^*$ that satisfies $L \odot L^{-1} = L^{-1} \odot L = \{\varepsilon\}$.*

Although a direct proof can be given, this result is a direct adaptation of a known fact on formal power series—namely, that it is invertible if and only if its constant term is invertible in the semiring of coefficients.

Example 3. $\{\varepsilon, ab\}^{-1} = (ab)^*$ and, accordingly, $((ab)^*)^{-1} = \{\varepsilon, ab\}$.

Example 4. $(a^*b^*)^{-1} = \{\varepsilon, a, b, ba\}$.

The inverse can be equivalently represented as the following adaptation of the Kleene star.

Definition 1. For every language L , its GF(2)-star, denoted by L^\circledast , is the set of all strings w that have an odd number of representations of the form $w = w_1w_2 \dots w_k$, with $k \geq 0$ and $w_1, \dots, w_k \in L \setminus \{\varepsilon\}$.

Lemma 1. For every language L , a string w is in L^\circledast if and only if the number of representations $w = uv$, with $u \in L \setminus \{\varepsilon\}$ and $v \in L^\circledast$, is odd.

Theorem 2. For every language $L \subseteq \Sigma^*$ with $\varepsilon \in L$, the inverse L^{-1} equals the star L^\circledast .

3 Closure Properties and State Complexity

3.1 GF(2)-Concatenation on Regular Languages

For every operation on languages, the first basic question is whether it preserves the class of regular languages. If it does, the next question is its descriptive complexity, that is, how large an automaton is necessary to represent this operation on finite automata of a given size. The closure holds both for GF(2)-concatenation and for GF(2)-inverse, and deterministic finite automata (DFA) implementing these operations, which are optimal with respect to the number of states, are constructed below.

Theorem 3. Let $\mathcal{A} = (\Sigma, P, p_0, \eta, E)$ and $\mathcal{B} = (\Sigma, Q, q_0, \delta, F)$ be two DFA. Then the language $L(\mathcal{A}) \odot L(\mathcal{B})$ is recognized by a DFA \mathcal{C} with the set of states $P \times 2^Q$.

Proof (a sketch). The states of \mathcal{C} are of the form (p, S) , where $p \in P$ is the current state of \mathcal{A} reading the same input string, while S is the set of all states reached an odd number of times in the simulated computations of \mathcal{B} .

The initial state of \mathcal{C} is $(p_0, \{q_0\})$ if $\varepsilon \in \mathcal{A}$, and (p_0, \emptyset) otherwise.

The transition in a state (p, S) by a symbol a advances all currently simulated computations of \mathcal{B} cancelling out all states reached an even number of times.

$$S' = \{q' \mid \text{the number of states } q \in S, \text{ with } q' = \delta(q, a), \text{ is odd}\}$$

One step of \mathcal{A} is also simulated, and if it enters an accepting state, then another computation of \mathcal{B} is started.

$$\pi((p, S), a) = \begin{cases} (\eta(p, a), S'), & \text{if } \eta(p, a) \notin E \\ (\eta(p, a), S' \Delta \{q_0\}), & \text{if } \eta(p, a) \in E \end{cases}$$

A state (p, S) is accepting, if S has an odd number of accepting states of \mathcal{B} . \square

As shown in the next theorem, using all states in $P \times 2^Q$ is necessary in the worst case.

Theorem 4. *For every $m, n \geq 3$, there exist languages K and L over an alphabet $\{a, b\}$, recognized by an m -state DFA and by an n -state DFA, respectively, for which every DFA recognizing their GF(2)-concatenation $K \odot L$ must have at least $m \cdot 2^n$ states.*

Proof (a sketch). The language K is recognized by the DFA $\mathcal{A} = (\Sigma, \{0, \dots, m-1\}, 0, \eta, \{m-1\})$, with the transitions $\eta(i, a) = i$ and $\eta(i, b) = i + 1 \pmod{m}$, for all i . The automaton recognizing L is $\mathcal{B} = (\Sigma, \{0, \dots, n-1\}, 0, \delta, \{n-1\})$, with the following transitions: $\delta(i, a) = i + 1$, if $i \neq k-1$; $\delta(k-1, a) = k-1$; $\delta(k-2, b) = k-1$; $\delta(k-1, b) = k-2$; and $\delta(i, b) = i$ for the remaining states.

Let \mathcal{C} be the DFA with $m \cdot 2^n$ states constructed for \mathcal{A} and \mathcal{B} in Theorem 3. For every state (p, S) of \mathcal{C} , it is proved that it is reachable from the initial state q'_0 by some string, whereas for every pair of distinct states, some string is accepted from one of these states and not from the other. \square

This establishes the precise state complexity of GF(2)-concatenation for DFA as $m \cdot 2^n$. To compare, the state complexity of the classical concatenation is $m \cdot 2^n - 2^{n-1}$, as proved by Maslov [7], see also Yu et al. [15]. The number of states in an NFA representing GF(2)-concatenation of two NFA remains open: the only available construction is by determinizing the given automata and then applying Theorem 3, which yields an upper bound of $2^m \cdot 2^{2^n}$ states. On the other hand, for the class of *symmetric difference automata* (\oplus FA), studied by van Zijl [16], an automaton for GF(2)-concatenation can be naturally obtained by a series composition of two automata, using $m + n$ states.

3.2 GF(2)-Inverse on Regular Languages

Theorem 5. *For every n -state DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, the language $L(\mathcal{A})^\circledast$ is recognized by a DFA \mathcal{C} with the set of states $2^Q \cup \{q'_0\}$.*

Proof (a sketch). The construction relies on the representation of the GF(2)-star according to Lemma 1. All states of \mathcal{C} , except its initial state q'_0 , are subsets $S \subseteq Q$. The subset S consists of all states of \mathcal{A} reached an odd number of times in the simulated computations of \mathcal{A} .

Transitions in the initial state q'_0 start a single computation of \mathcal{A} as $\pi(q'_0, a) = \{\delta(q_0, a)\}$. The transition in a state S by $a \in \Sigma$ continues all simulations, leaving only the states reached an odd number of times. If $|S \cap F|$ is even, then

$$\pi(S, a) = \{q \mid \# \text{ of states } p \in S \text{ with } \delta(p, a) = q \text{ is odd}\},$$

and if $|S \cap F|$ is odd, then another computation is started.

$$\pi(S, a) = \{q \mid \# \text{ of states } p \in S \text{ with } \delta(p, a) = q \text{ is odd}\} \Delta \delta(q_0, a).$$

Subsets with an odd number of accepting states of \mathcal{A} are accepting in \mathcal{C} . \square

This construction is optimal as well, and it is optimal even for the special case of GF(2)-inverses (that is, for languages containing the empty string).

Theorem 6. *For every $n \geq 3$, there exist a language L over an alphabet $\Sigma = \{a, b, c\}$, with $\varepsilon \in L$, which is recognized by a DFA with n states, whereas every DFA recognizing its GF(2)-inverse L^{-1} must have at least $2^n + 1$ states.*

Proof. The desired n -state DFA has the set of states $Q = \{0, \dots, n-1\}$, where 0 is the initial state and the accepting states are 0 and $n-1$. The transitions in each state $i \in Q$ are defined as $\delta(a, i) = \min(i+1, n-1)$, $\delta(b, i) = i$ and $\delta(c, i) = \max(i-1, 0)$.

Then, for the DFA \mathcal{C} with the set of states $\{q'_0\} \cup 2^Q$, as constructed for \mathcal{A} by Theorem 5, it is proved that every subset $S \subseteq Q$ is reachable from the initial state q'_0 by some string, and that for every two states of \mathcal{C} , there exists a string that is accepted from one of them and not from the other. \square

For NFA, the GF(2)-star is representable using $2^{2^n} + 1$ states. For \oplus FA, the same can be naturally done using $n + 1$ states.

Computations indicate that the GF(2)-inverse over the unary alphabet apparently requires DFA with $2^{n-1} + 1$ states, with the worst-case examples producing DFA with period $2^{n-1} - 1$.

3.3 Non-closure Results

The non-closure of the context-free languages under the symmetric difference is well-known: indeed, there is no grammar for the language $\{a^\ell b^m c^n \mid \ell = m \text{ or } m = n, \text{ but not both}\}$. This non-closure extends to their subclasses: the unambiguous languages, the LR languages, etc. From this fact, the non-closure under GF(2)-concatenation and GF(2)-inverse is inferred as follows.

Lemma 2. *Let $K, L \subseteq \Sigma^+$ be any languages that do not contain the empty string, let $c \notin \Sigma$ be a new symbol, and let $M = \{c\} \cup cK$ and $N = \{c\} \cup Lc$. Then a string cwc , with $w \in \Sigma^+$, is in $M \odot N$ if and only if $w \in K \Delta L$.*

Then, if K and L are defined by grammars, then so are the languages M and N , and if their GF(2) concatenation $M \odot N$ were defined by a grammar as well, then there would be a grammar for $K \Delta L$. The non-closure under GF(2)-inverse is established similarly, using the following representation.

Lemma 3. *Let $K, L \subseteq \Sigma^+$ be any languages that do not contain the empty string, let $c \notin \Sigma$ be a new symbol, and let $M = \{\varepsilon, c\} \cup cK \cup Lc$. Then a string cwc , with $w \in \Sigma^*$, is in M^{-1} if and only if $w \in K \Delta L$.*

4 Formal Grammars with GF(2)-Operations

In the ordinary kind of formal grammars, called “context-free grammars” in Chomsky’s tradition, the available operations are union and concatenation. The

rest of grammar families, such as linear grammars or conjunctive grammars [8], differ from the ordinary grammars in the sets of available operations: in linear grammars, the operations include concatenation with single symbols and the union, whereas in conjunctive grammars, the operations are union, intersection and concatenation. This paper initiates the study of a new model, the *GF(2)-grammars*, with the operations of symmetric difference and GF(2)-concatenation.

One should note that none of the aforementioned grammar families involves any context dependencies, they differ only in the set of allowed operations. Hence, grammars with union and concatenation (Chomsky's "context-free") shall be referred to as *ordinary grammars*, to distinguish them from the GF(2)-grammars, which are defined in almost the same way.

Definition 2. A *GF(2)-grammar* is a quadruple $G = (\Sigma, N, R, S)$, in which Σ is the alphabet, N is the set of nonterminal symbols, with the initial symbol $S \in N$ and every rule in R is of the form $A \rightarrow X_1 \odot \dots \odot X_\ell$, with $\ell \geq 0$ and $X_1, \dots, X_\ell \in \Sigma \cup N$.

A grammar is linear GF(2), if, in each rule, at most one of X_1, \dots, X_ℓ is a nonterminal symbol.

Multiple rules for the same nonterminal can be written down using the sum modulo 2 operator (\oplus), so that two rules, $A \rightarrow B \odot C$ and $A \rightarrow D \odot E$, are written down as $A \rightarrow (B \odot C) \oplus (D \odot E)$. The general plan is to define the language described by a grammar, so that it is a solution of the corresponding system of language equations in variables N , where the sum modulo 2 operator is implemented as the symmetric difference of languages.

$$A = \Delta_{A \rightarrow X_1 \odot \dots \odot X_\ell \in R} X_1 \odot \dots \odot X_\ell \quad (A \in N) \quad (*)$$

However, there are certain complications, because in some cases this system has no solutions; for instance, such is the system corresponding to the grammar $S \rightarrow (S \odot S) \oplus \varepsilon$. The general theory of weighted grammars with weights from a semiring—see, for instance, a survey by Petre and Salomaa [10]—is not applicable here, because it requires certain monotonicity and continuity conditions, which do not hold for coefficients in GF(2). For that reason, the proposed definition requires the number of parse trees to be finite.

Definition 3. For each GF(2)-grammar $G = (\Sigma, N, R, S)$, let $\widehat{G} = (\Sigma, N, \widehat{R}, S)$ be the corresponding ordinary grammar with \widehat{R} containing a rule $A \rightarrow X_1 \dots X_\ell$ for each rule $A \rightarrow X_1 \odot \dots \odot X_\ell$ in R . Assuming that, for all $A \in N$ and $w \in \Sigma^*$, the number of parse trees of w as A in \widehat{G} is finite, the language $L_G(A)$ is defined as the set of all strings w with an odd number of parse trees as A in \widehat{G} . If the finiteness condition does not hold, then G is considered ill-formed.

The languages thus defined satisfy the system of language equations, and hence implement the desired definition.

Proposition 2. Let $G = (\Sigma, N, R, S)$ be a GF(2)-grammar that satisfies the condition in Definition 3. Then the substitution $A = L_G(A)$ for all $A \in N$ is a solution of the system (*).

What kind of languages can be described by GF(2)-grammars? First of all, if an unambiguous grammar is transcribed as a GF(2)-grammar, it still defines the same language. On the other hand, taking any ambiguous grammar and reinterpreting it as a GF(2)-grammar leads to simple non-trivial examples of these grammars.

Example 5. The linear GF(2)-grammar given below describes the language $L = \{a^\ell b^m c^n \mid \ell = m \text{ or } m = n, \text{ but not both}\}$.

$$\begin{array}{ll} S \rightarrow A \oplus C & \\ A \rightarrow aA \oplus B & C \rightarrow Cc \oplus D \\ B \rightarrow bBc \oplus \varepsilon & D \rightarrow aDb \oplus \varepsilon \end{array}$$

Here, as well as in all later examples, the GF(2)-concatenation with fixed strings u and v is denoted simply by uLv , owing to the fact that $\{u\} \cdot L \cdot \{v\} = \{u\} \odot L \odot \{v\}$ for every language L .

The same language is not representable by an ordinary grammar. At the same time, for the language $\{a^\ell b^m c^n \mid \ell = m \text{ or } m = n\}$ described by a similarly defined ordinary grammar, it is unclear whether a GF(2)-grammar exists.

The next example is of interest, in particular, because it is not known whether conjunctive grammars can describe languages of this kind [8, Problem 6].

Example 6. The following GF(2)-grammar describes the language $\{uv \mid u, v \in \{a, b\}^*, |u| = |v|, u \text{ and } v \text{ differ in an odd number of positions}\}$.

$$\begin{array}{l} S \rightarrow (A \odot B) \oplus (B \odot A) \\ A \rightarrow aAa \oplus aAb \oplus bAa \oplus bAb \oplus a \\ B \rightarrow aBa \oplus aBb \oplus bBa \oplus bBb \oplus b \end{array}$$

The next grammar representing a non-regular unary language achieves a result similar to a conjunctive grammar given by Jeż [5], and to a language equation with concatenation and complementation given by Leiss [6], even though there is apparently nothing common in the methods.

Example 7. The following grammar describes the language $\{a^{2^n} \mid n \geq 0\}$.

$$S \rightarrow (S \odot S) \oplus a$$

Indeed, the number of parse trees of a string a^ℓ in the grammar $S \rightarrow SS \mid a$ is the $(\ell - 1)$ -th Catalan number, which is odd if and only if ℓ is a power of 2.

The latter example indicates that GF(2)-grammars are not in all respects symmetric to ordinary grammars with concatenation and disjunction: whereas ordinary grammars over a unary alphabet can define only regular languages, GF(2)-grammars are strictly more powerful in this case.

5 Parsing for GF(2)-Grammars

In order to develop parsing algorithms for GF(2)-grammars and to assess their computational complexity, it is convenient to obtain a normal form for these grammars first. The following adaptation of the Chomsky normal form can be established.

Theorem 7. *Every GF(2)-grammar can be effectively transformed in polynomial time to a GF(2)-grammar that describes the same language, and has all rules of the form $A \rightarrow B \odot C$, with $B, C \in N$, or $A \rightarrow a$, with $a \in \Sigma$.*

The transformation follows the standard path of first cutting long rules into rules with at most two symbols on the right-hand side, then eliminating *null rules* of the form $A \rightarrow \epsilon$, next, eliminating *unit rules* of the form $A \rightarrow B$, with $B \in N$, and finally, moving all occurrences of terminal symbols on the right-hand sides to separate rules.

The elimination of null rules begins with determining all nonterminals that define the empty string.

Lemma 4. *For every ordinary grammar $G = (\Sigma, N, R, S)$ with finitely many parse trees of ϵ from each $A \in N$, the set $\text{ODD-NULLABLE} = \{A \mid \# \text{ of parses of } \epsilon \text{ from } A \text{ is odd}\}$ can be constructed in polynomial time.*

Since the number of parse trees is finite, the set ODD-NULLABLE is well-defined, and the only question is how to construct it efficiently. The algorithm uses the finiteness of the number of parse trees to determine the partial order, in which the nonterminals may occur in parse trees of ϵ . Then the algorithm calculates the number of parse trees in this order.

Using this set, the null rules are removed by the standard construction, modified to use the parity instead of existence.

Lemma 5. *For every GF(2) grammar $G = (\Sigma, N, R, S)$, let $G' = (\Sigma, N, R', S)$ be another GF(2) grammar that contains a rule $A \rightarrow X_1 \odot \dots \odot X_\ell$, if $\ell \geq 1$ and the number of rules $A \rightarrow \theta_0 \odot X_1 \odot \theta_1 \odot \dots \odot X_\ell \odot \theta_\ell$ in R , with each θ_i being a GF(2)-concatenation of zero or more nonterminals in ODD-NULLABLE, is odd. Then, for every $A \in N$, $L_{G'}(A) = L_G(A) \setminus \{\epsilon\}$.*

There is also a similar “parity” version of the classical unit rules elimination.

Lemma 6. *For every GF(2) grammar $G = (\Sigma, N, R, S)$ with no rules of the form $A \rightarrow \epsilon$, let $G' = (\Sigma, N, R', S)$ be another GF(2) grammar that contains a rule $A \rightarrow X_1 \odot \dots \odot X_\ell$, with $\ell \geq 2$ or $X_1 \in \Sigma$, if there is an odd number of chains of the form $A \rightarrow B_1, B_1 \rightarrow B_2, \dots, B_{n-1} \rightarrow B_n, B_n \rightarrow X_1 \odot \dots \odot X_\ell$. Then, for every $A \in N$, $L_{G'}(A) = L_G(A)$.*

With the normal form theorem established, an adaptation of the Cocke–Kasami–Younger algorithm follows immediately: for a GF(2)-grammar $G =$

(Σ, N, R, S) in the normal form, given an input string $w = a_1 \dots a_n$, the algorithm constructs the following sets inductively on the length of substring.

$$T_{i,j} = \{ A \in N \mid a_{i+1} \dots a_j \in L_G(A) \} \quad (\text{for } 0 \leq i < j \leq n)$$

For each one-symbol substring, the set is $T_{j-1,j} = \{ A \mid A \rightarrow a_j \in R \}$, and for every longer substring it is determined by the following expression.

$$T_{i,j} = \bigtriangleup_{k=i+1}^{j-1} \{ A \mid \# \text{ of rules } A \rightarrow BC, \text{ with } (B, C) \in T_{i,k} \times T_{k,j}, \text{ is odd} \}$$

Overall, there are $O(n^2)$ elements in the table, each of them is calculated in time $O(n)$, and thus the algorithm works in time $O(n^3)$.

Using Valiant's [14] algorithm, the same table $T_{i,j}$ can be constructed in time $O(n^\omega)$, where $O(n^\omega)$ is an upper bound on the number of operations needed to multiply a pair of $n \times n$ matrices in GF(2).

Theorem 8. *Every language described by a GF(2)-grammar is in DTIME(n^ω).*

In fact, practical algorithms for matrix multiplication over GF(2) are substantially faster than those for Boolean matrix multiplication: for smaller matrices there is a particularly efficient implementation of the Four Russians method [1], and for large matrices one can apply Strassen's algorithm directly. This makes the algorithm more efficient than the parsing algorithms for ordinary grammars.

The next question concerns the complexity of the uniform membership problem, where both the string and the grammar are given.

Theorem 9. *The uniform membership problem for GF(2)-grammars, stated as "Given a GF(2) grammar G and a string w , determine whether w is in $L(G)$ ", is P-complete.*

Proof (a sketch). The problem is decided in polynomial time by first transforming the given grammar to the normal form and then applying the cubic-time parsing algorithm. Its P-hardness follows from the P-hardness of the Circuit Value Problem, similarly to the classical result for ordinary grammars. \square

An extension of the Brent–Goldschläger–Rytter $(\log n)^2$ -time parallel parsing algorithm [2, 12] requires some work. The original algorithm determines the *existence* of parse trees, and while doing so, it may consider the same tree an unspecified number of times. In the algorithm for GF(2)-grammars, which has to test that the number of trees is *odd*, it is imperative that every subtree is considered exactly once. In a different context, Rossmanith and Rytter [11] ensured this property in a similar algorithm for unambiguous grammars. The following algorithm does this without the unambiguity assumption.

Theorem 10. *Let $G = (\Sigma, N, R, S)$ be a GF(2)-grammar. Then there is a uniform family of circuits for testing strings of length n for being in $L(G)$, which are of depth $O((\log n)^2)$ and contain $O(n^7)$ nodes.*

Proof (a sketch). The circuit has the following main gates: for each $A \in N$, a gate (A, i, j) determines whether the number of parse trees of $a_{i+1} \dots a_j$ from A is odd; for $A, D \in N$, a gate (A, D, i, k, ℓ, j) similarly computes the parity of the number of parse trees with root A and with a missing subtree with root D , where the leaves left of the path from A to D are $a_{i+1} \dots a_k$, and the leaves right of that path are $a_{\ell+1} \dots a_j$. Each gate is expressed through the gates referring to substrings of length smaller by a constant factor, using $O(n^3)$ intermediate gates, and thus ensuring logsquare circuit depth. The structure of the computation elaborates the structure of the Brent–Goldschlager–Rytter algorithm. \square

The last complexity question concerns *linear GF(2) grammars*, which have all rules of the form $A \rightarrow uBv$ or $A \rightarrow w$, with $B \in N$ and $u, v, w \in \Sigma^*$. Whereas ordinary linear grammars with the union operation are known to have an NL-complete uniform membership problem [13], the same problem for linear GF(2)-grammars is—by all means, predictably—complete for the complexity class $\oplus L$ of all problems decided by a nondeterministic logarithmic-space Turing machine that accepts by having an odd number of accepting paths.

Theorem 11. *The uniform membership problem for linear GF(2)-grammars is $\oplus L$ -complete. Furthermore, there exists a linear GF(2)-grammar that describes an $\oplus L$ -complete language.*

Proof (a sketch). The uniform membership problem for linear GF(2)-grammars is solved in $\oplus L$ by attempting to parse a given string using two pointers, choosing each rule nondeterministically.

The “hardest” linear GF(2)-grammar is based on the problem of testing whether the number of s - t -paths in a given directed graph is odd; this problem is $\oplus L$ -complete, see Damm [3]. A grammar for the yes-instances of this problem can reuse the classical construction by Sudborough [13]: the encoding of graphs remains the same, and the union operation in the grammar is replaced with the symmetric difference. \square

The complexity of grammar families is compared in Table 1. It would be important to know whether the emptiness problem for GF(2)-grammars is

Table 1. Complexity of grammars with different operations.

	Fixed membership			Uniform
	Time	Space	Class	
Unamb. linear (\sqcup , LIN \cdot)	$O(n^2)$	$O((\log n)^2)$	in UL	in UL
Linear (\cup , LIN \cdot)	$O(n^2)$	$O((\log n)^2)$	NL-complete [13]	NL-complete
Linear GF(2) (\triangle , LIN \cdot)	$O(n^2)$	$O((\log n)^2)$	$\oplus L$-complete	$\oplus L$-complete
Unambiguous (\sqcup , UNAMB \cdot)	$O(n^2)$	$O((\log n)^2)$	in NC 2 [11]	P-complete
Ordinary (\cup , \cdot)	$O(n^\omega)$ [14]	$O((\log n)^2)$	in NC 2 [2, 12]	P-complete
GF(2) (\triangle , \odot)	$O(n^\omega)$	$O((\log n)^2)$	in NC2	P-complete
Conjunctive (\cup, \cap, \cdot)	$O(n^\omega)$ [9]	$O(n)$ [8]	P-complete	P-complete

decidable. However, the equivalence problem for the unambiguous grammars reduces to this problem, and whether it is decidable is a major and long-standing open problem in formal language theory. On the other hand, if the emptiness of GF(2)-grammars is undecidable, proving that would require new methods.

Another important question is developing a method for proving that some languages are not described by any GF(2)-grammar.

References

1. Albrecht, M.R., Bard, G.V., Hart, W.: Algorithm 898: efficient multiplication of dense matrices over GF(2). *ACM Trans. Math. Softw.* **37**(1), 9:1–9:14 (2010)
2. Brent, R.P., Goldschlager, L.M.: A parallel algorithm for context-free parsing. *Aust. Comp. Sci. Comm.* **6**(7), 1–10 (1984)
3. Damm, C.: Problems complete for $\oplus L$. *Inf. Process. Lett.* **36**(5), 247–250 (1990)
4. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. *J. ACM* **9**(3), 350–371 (1962)
5. Jeż, A.: Conjunctive grammars generate non-regular unary languages. *Int. J. Found. Comput. Sci.* **19**(3), 597–615 (2008)
6. Leiss, E.L.: Unrestricted complementation in language equations over a one-letter alphabet. *Theor. Comput. Sci.* **132**(2), 71–84 (1994)
7. Maslov, A.N.: Estimates of the number of states of finite automata. *Sov. Math. Dokl.* **11**, 1373–1375 (1970)
8. Okhotin, A.: Conjunctive and Boolean grammars: the true general case of the context-free grammars. *Comput. Sci. Rev.* **9**, 27–59 (2013)
9. Okhotin, A.: Parsing by matrix multiplication generalized to Boolean grammars. *Theor. Comput. Sci.* **516**, 101–120 (2014)
10. Petre, I., Salomaa, A.: Algebraic systems and pushdown automata. In: Droste, M., Kuich, W., Vogler, H. (eds.) *Handbook of Weighted Automata*, pp. 257–289. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-01492-5_7. Chap. 7
11. Rossmanith, P., Rytter, W.: Observations on $\log(n)$ time parallel recognition of unambiguous cfl's. *Inf. Process. Lett.* **44**(5), 267–272 (1992)
12. Rytter, W.: On the recognition of context-free languages. In: *Computation Theory - Proceedings of 5th Symposium*, Zaborów, Poland, 3–8 December 1984, pp. 318–325 (1984)
13. Sudborough, I.H.: A note on tape-bounded complexity classes and linear context-free languages. *J. ACM* **22**(4), 499–500 (1975)
14. Valiant, L.G.: General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.* **10**(2), 308–315 (1975)
15. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.* **125**(2), 315–328 (1994)
16. van Zijl, L.: Magic numbers for symmetric difference NFAs. *Int. J. Found. Comput. Sci.* **16**(5), 1027–1038 (2005)



Event-Clock Nested Automata

Laura Bozzelli, Aniello Murano^(✉), and Adriano Peron

Università degli Studi di Napoli Federico II, Naples, Italy
murano@na.infn.it

Abstract. In this paper we introduce and study Event-Clock Nested Automata (ECNA), a formalism that combines Event Clock Automata (ECA) and Visibly Pushdown Automata (VPA). ECNA allow to express real-time properties over non-regular patterns of recursive programs. We prove that ECNA retain the closure and decidability properties of ECA and VPA being closed under Boolean operations and having a decidable language-inclusion problem. In particular, we prove that emptiness, universality, and language-inclusion for ECNA are EXPTIME-complete problems. As for the expressiveness, we have that ECNA properly extend any previous attempt in the literature of combining ECA and VPA.

1 Introduction

Model checking is a well-established formal-method technique to automatically check for global correctness of reactive systems [7]. In this setting, automata theory over infinite words plays a crucial role: the set of possible (potentially infinite) behaviors of the system and the set of admissible behaviors of the correctness specification can be modeled as languages accepted by automata. The verification problem of checking that a system meets its specification then reduces to testing language inclusion between two automata over infinite words.

In the last two decades, model checking of pushdown automata (PDA) has received a lot of attention [11, 17, 20]. PDA represent an infinite-state formalism suitable to model the control flow of typical sequential programs with nested and recursive procedure calls. Although the general problem of checking context-free properties of PDA is undecidable [16], algorithmic solutions have been proposed for interesting subclasses of context-free requirements [3, 5, 6, 13]. A well-known approach is that of *Visibly Pushdown Automata* (VPA) [5, 6], a subclass of PDA where the input symbols over a *pushdown alphabet* control the admissible operations on the stack. Precisely, the alphabet is partitioned into a set of *calls*, representing a procedure call and forcing a push stack-operation, a set of *returns*, representing a procedure return and forcing a pop stack-operation, and a set of *internal* actions that cannot access or modify the content of the stack. This restriction makes the class of resulting languages (*visibly pushdown languages* or VPL) very similar in tractability and robustness to that of regular languages [5, 6]. VPL are closed under Boolean operations, and language inclusion is EXPTIME-complete. VPA capture all regular properties, and, additionally, allow to specify regular requirements over two kinds of *non-regular* patterns on input words:

abstract paths and *caller paths*. An abstract path captures the local computation within a procedure with the removal of subcomputations corresponding to nested procedure calls, while a caller path represents the call-stack content at a given position of the input.

Recently, many works [1, 8, 10, 14, 15, 19] have investigated real-time extensions of PDA by combining PDA with *Timed Automata* (TA) [2], a model widely used to represent real-time systems. TA are finite automata augmented with a finite set of real-valued clocks, which operate over words where each symbol is paired with a real-valued timestamp (*timed words*). All clocks progress at same speed and can be reset by transitions (thus, each clock keeps track of the elapsed time since the last reset). Constraints on clocks are associated with transitions to restrict the behavior of the automaton. The emptiness problem for TA is decidable and PSPACE complete [2]. However, since in TA, clocks can be reset nondeterministically and independently of each other, the resulting class of timed languages is not closed under complement and, in particular, language inclusion is undecidable [2]. As a consequence, the general verification problem (i.e., language inclusion) of formalisms combining unrestricted TA with robust subclasses of PDA such as VPA, i.e. *Visibly Pushdown Timed Automata* (VPTA), is undecidable as well. In fact, checking language inclusion for VPTA is undecidable even in the restricted case of specifications using at most one clock [15].

Event-clock automata (ECA) [4] are an interesting subclass of TA where the explicit reset of clocks is disallowed. In ECA, clocks have a predefined association with the input alphabet symbols. Precisely, for each symbol a , there are two clocks: the *global recorder clock*, recording the time elapsed since the last occurrence of a , and the *global predictor clock*, measuring the time required for the next occurrence of a . Hence, the clock valuations are determined only by the input timed word being independent of the automaton behavior. Such a restriction makes the resulting class of timed languages closed under Boolean operations, and in particular, language inclusion is PSPACE-complete [4].

Recently, a robust subclass of VPTA, called *Event-Clock Visibly Pushdown Automata* (ECVPA), has been proposed in [18], combining ECA with VPA. ECVPA are closed under Boolean operations, and language inclusion is EXPTIME-complete. However, ECVPA do not take into account the nested hierarchical structure induced by a timed word over a pushdown alphabet, namely, they do not provide any explicit mechanism to relate the use of a stack with that of event clocks.

Our contribution. In this paper, we introduce an extension of ECVPA, called *Event-Clock Nested Automata* (ECNA) that, differently from ECVPA, allows to relate the use of event clocks and the use of the stack. To this end, we add for each input symbol a , three additional event clocks: the *abstract recorder clock* (resp., *abstract predictor clock*), measuring the time elapsed since the last occurrence (resp., the time for the next occurrence) of a along the maximal abstract path visiting the current position; the *caller clock*, measuring the time elapsed since the last occurrence of a along the caller path from the current position. In this way, ECNA allow to specify relevant real-time non-regular properties including:

- Local bounded-time responses such as “in the local computation of a procedure A , every request p is followed by a response q within k time units”.
- Bounded-time total correctness requirements such as “if the pre-condition p holds when the procedure A is invoked, then the procedure must return within k time units and q must hold upon return”.
- Real-time security properties which require the inspection of the call-stack such as “a module A should be invoked only if module B belongs to the call stack and within k time units since the activation of module B ”.

We show that ECNA are strictly more expressive than ECVPA and, as for ECVPA, the resulting class of languages is closed under all Boolean operations. Moreover, language inclusion and visibly model-checking of VPTA against ECNA specifications are decidable and EXPTIME-complete. The key step in the proposed decision procedures is a translation of ECNA into equivalent VPTA.

Related work. Pushdown Timed Automata (PTA) have been introduced in [10], and their emptiness problem is EXPTIME-complete. An extension of PTA, namely Dense-Timed Pushdown Automata (DTPA), has been studied in [1], where each symbol in the stack is equipped with a real-valued clock representing its ‘age’ (the time elapsed since the symbol has been pushed onto the stack). It has been shown in [14] that DTPA do not add expressive power and can be translated into equivalent PTA. Our proposed translation of ECNA into VPTA is inspired from the construction in [14]. In [9], an equally-expressive extension of ECVPA [18] over finite timed words, by means of a *timed* stack (like in DTPA), is investigated.

2 Preliminaries

In the following, \mathbb{N} denotes the set of natural numbers and \mathbb{R}_+ the set of non-negative real numbers. Let w be a finite or infinite word over some alphabet. By $|w|$ we denote the length of w (we set $|w| = \infty$ if w is infinite). For all $i, j \in \mathbb{N}$, with $i \leq j$, w_i is i -th letter of w , while $w[i, j]$ is the finite subword $w_i \cdots w_j$.

An *infinite timed word* w over a finite alphabet Σ is an infinite word $w = (a_0, \tau_0)(a_1, \tau_1), \dots$ over $\Sigma \times \mathbb{R}_+$ (intuitively, τ_i is the time at which a_i occurs) such that the sequence $\tau = \tau_0, \tau_1, \dots$ of timestamps satisfies: (1) $\tau_i \leq \tau_{i+1}$ for all $i \geq 0$ (monotonicity), and (2) for all $t \in \mathbb{R}_+$, $\tau_i \geq t$ for some $i \geq 0$ (divergence). The timed word w is also denoted by the pair (σ, τ) , where σ is the untimed word $a_0 a_1 \dots$ and τ is the sequence of timestamps. An ω -*timed language* over Σ is a set of infinite timed words over Σ .

Pushdown alphabets, abstract paths, and caller paths. A *pushdown alphabet* is a finite alphabet $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$ which is partitioned into a set Σ_{call} of *calls*, a set Σ_{ret} of *returns*, and a set Σ_{int} of *internal actions*. The pushdown alphabet Σ induces a nested hierarchical structure in a given word over Σ obtained by associating to each call the corresponding matching return (if any) in a well-nested manner. Formally, the set of *well-matched words* is the set of finite words σ_w over Σ inductively defined by the following grammar:

$$\sigma_w := \varepsilon \mid a \cdot \sigma_w \mid c \cdot \sigma_w \cdot r \cdot \sigma_w$$

where ε is the empty word, $a \in \Sigma_{int}$, $c \in \Sigma_{call}$, and $r \in \Sigma_{ret}$.

Fix an infinite word σ over Σ . For a call position $i \geq 0$, if there is $j > i$ such that j is a return position of σ and $\sigma[i+1, j-1]$ is a well-matched word (note that j is uniquely determined if it exists), we say that j is the *matching return* of i along σ . For a position $i \geq 0$, the *abstract successor* of i along σ , denoted $\text{succ}(a, \sigma, i)$, is defined as follows:

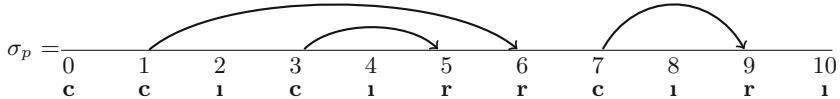
- If i is a call, then $\text{succ}(a, \sigma, i)$ is the matching return of i , if such a matching return exists; otherwise, $\text{succ}(a, \sigma, i) = \perp$ (\perp denotes the *undefined* value).
- If i is not a call, then $\text{succ}(a, \sigma, i) = i+1$ if $i+1$ is not a return position, and $\text{succ}(a, \sigma, i) = \perp$, otherwise.

The *caller* of i along σ , denoted $\text{succ}(c, \sigma, i)$, is instead defined as follows:

- if there exists the greatest call position $j_c < i$ such that either $\text{succ}(a, \sigma, j_c) = \perp$ or $\text{succ}(a, \sigma, j_c) > i$, then $\text{succ}(c, \sigma, i) = j_c$; otherwise, $\text{succ}(c, \sigma, i) = \perp$.

In the analysis of recursive programs, a *maximal abstract path* captures the local computation within a procedure removing computation fragments corresponding to nested calls, while the *caller path* represents the call-stack content at a given position of the input. Formally, a *maximal abstract path* (*MAP*) of σ is a *maximal* (finite or infinite) increasing sequence of natural numbers $\nu = i_0 < i_1 < \dots$ such that $i_j = \text{succ}(a, \sigma, i_{j-1})$ for all $1 \leq j < |\nu|$. Note that for every position i of σ , there is exactly one *MAP* of σ visiting position i . For each $i \geq 0$, the *caller path* of σ from position i is the maximal (finite) decreasing sequence of natural numbers $j_0 > j_1 \dots > j_n$ such that $j_0 = i$ and $j_{h+1} = \text{succ}(c, \sigma, j_h)$ for all $0 \leq h < n$. Note that the positions of a *MAP* have the same caller (if any).

For instance, consider the finite untimed word σ_p of length 10 depicted below where $\Sigma_{call} = \{c\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{1\}$.



Let σ be $\sigma_p \cdot 1^\omega$. Note that 0 is the unique unmatched call position of σ : hence, the *MAP* visiting 0 consists of just position 0 and has no caller. The *MAP* visiting position 1 is the infinite sequence $1, 6, 7, 9, 10, 11, 12, 13, \dots$ and the associated caller is position 0; the *MAP* visiting position 2 is the sequence $2, 3, 5$ and the associated caller is position 1, and the *MAP* visiting position 4 consists of just position 4 whose caller path is $4, 3, 1, 0$.

3 Event-Clock Nested Automata

In this section, we define the formalism of *Event-Clock Nested Automata* (ECNA), which allow a combined used of event clocks and visible operations on the stack.

To this end, we augment the standard set of event clocks [4] with a set of *abstract event clocks* and a set of *caller event clocks* whose values are determined by considering maximal abstract paths and caller paths of the given word, respectively.

In the following, we fix a pushdown alphabet $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$. The set C_Σ of event clocks associated with Σ is given by $C_\Sigma := \bigcup_{b \in \Sigma} \{x_b^g, y_b^g, x_b^a, y_b^a, x_b^c\}$. Thus, we associate with each symbol $b \in \Sigma$, five event clocks: the *global recorder clock* x_b^g (resp., the *global predictor clock* y_b^g) recording the time elapsed since the last occurrence of b , if any, (resp., the time required to the next occurrence of b if any); the *abstract recorder clock* x_b^a (resp., the *abstract predictor clock* y_b^a) recording the time elapsed since the last occurrence of b , if any, (resp. the time required to the next occurrence of b) along the *MAP* visiting the current position; and the *caller (recorder) clock* x_b^c recording the time elapsed since the last occurrence of b if any along the caller path from the current position. Let $w = (\sigma, \tau)$ be an infinite timed word over Σ and $i \geq 0$. We denote by $Pos(a, w, i)$ the set of positions visited by the *MAP* of σ associated with position i , and by $Pos(c, w, i)$ the set of positions visited by the caller path of σ from position i . In order to allow a uniform notation, we write $Pos(g, w, i)$ to mean the full set \mathbb{N} of positions. Fixed the input word w , when the automaton reads the i -th position σ_i at time τ_i , the values of the clocks are uniquely determined as follows.

Definition 1 (Input determinisitic clock valuations). A *clock valuation* over C_Σ is a mapping $val : C_\Sigma \mapsto \mathbb{R}_+ \cup \{\perp\}$, assigning to each event clock a value in $\mathbb{R}_+ \cup \{\perp\}$ (\perp denotes the *undefined* value). Given an infinite timed word $w = (\sigma, \tau)$ over Σ and a position i , the *clock valuation* val_i^w over C_Σ , specifying the values of the event clocks at position i along w , is defined as follows for each $b \in \Sigma$, where $dir \in \{g, a, c\}$ and $dir' \in \{g, a\}$:

$$val_i^w(x_b^{dir}) = \begin{cases} \tau_i - \tau_j & \text{if } \exists j < i : b = \sigma_j, j \in Pos(dir, w, i), \text{ and} \\ & \forall k : (j < k < i \text{ and } k \in Pos(dir, w, i)) \Rightarrow b \neq \sigma_k \\ \perp & \text{otherwise} \end{cases}$$

$$val_i^w(y_b^{dir'}) = \begin{cases} \tau_j - \tau_i & \text{if } \exists j > i : b = \sigma_j, j \in Pos(dir', w, i), \text{ and} \\ & \forall k : (i < k < j \text{ and } k \in Pos(dir', w, i)) \Rightarrow b \neq \sigma_k \\ \perp & \text{otherwise} \end{cases}$$

It is worth noting that while the values of the global clocks are obtained by considering the full set of positions in w , the values of the abstract clocks (resp., caller clocks) are defined with respect to the *MAP* visiting the current position (resp., with respect to the caller path from the current position).

For $C \subseteq C_\Sigma$ and a clock valuation val over C_Σ , $val|_C$ denotes the restriction of val to the set C . A *clock constraint* over C is a conjunction of atomic formulas of the form $z \in I$, where $z \in C$, and I is either an interval in \mathbb{R}_+ with bounds in $\mathbb{N} \cup \{\infty\}$, or the singleton $\{\perp\}$ (also denoted by $[\perp, \perp]$). For a clock valuation val and a clock constraint θ , val satisfies θ , written $val \models \theta$, if for each conjunct $z \in I$ of θ , $val(z) \in I$. We denote by $\Phi(C)$ the set of clock constraints over C .

For technical convenience, we first introduce an extension of the known class of *Visibly Pushdown Timed Automata* (VPTA) [10, 15], called *nested* VPTA.

Nested VPTA are simply VPTA augmented with event clocks. Therefore, transitions of *nested* VPTA are constrained by a pair of disjoint finite sets of clocks: a finite set C_{st} of standard clocks and a disjoint set $C \subseteq C_\Sigma$ of event clocks. As usual, a standard clock can be reset when a transition is taken; hence, its value at a position of an input word depends in general on the behaviour of the automaton and not only, as for event clocks, on the word.

The class of *Event-Clock Nested Automata* (ECNA) corresponds to the subclass of nested VPTA where the set of standard clocks C_{st} is empty.

A (standard) clock valuation over C_{st} is a mapping $sval : C_{st} \mapsto \mathbb{R}_+$ (note that the undefined value \perp is not admitted). For $t \in \mathbb{R}_+$ and a reset set $Res \subseteq C_{st}$, $sval + t$ and $sval[Res]$ denote the valuations over C_{st} defined as follows for all $z \in C_{st}$: $(sval + t)(z) = sval(z) + t$, and $sval[Res](z) = 0$ if $z \in Res$ and $sval[Res](z) = sval(z)$ otherwise. For $C \subseteq C_\Sigma$ and a valuation val over C , $val \cup sval$ denotes the valuation over $C_{st} \cup C$ defined in the obvious way.

Definition 2 (Nested VPTA). A Büchi *nested* VPTA over $\Sigma = \Sigma_{call} \cup \Sigma_{int} \cup \Sigma_{ret}$ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, D = C \cup C_{st}, \Gamma \cup \{\top\}, \Delta, F)$, where Q is a finite set of (control) states, $Q_0 \subseteq Q$ is a set of initial states, $C \subseteq C_\Sigma$ is a set of event clocks, C_{st} is a set of standard clocks disjoint with C_Σ , $\Gamma \cup \{\top\}$ is a finite stack alphabet, $\top \notin \Gamma$ is the special *stack bottom symbol*, $F \subseteq Q$ is a set of accepting states, and $\Delta_c \cup \Delta_r \cup \Delta_i$ is a transition relation, where ($D = C \cup C_{st}$):

- $\Delta_c \subseteq Q \times \Sigma_{call} \times \Phi(D) \times 2^{C_{st}} \times Q \times \Gamma$ is the set of *push transitions*,
- $\Delta_r \subseteq Q \times \Sigma_{ret} \times \Phi(D) \times 2^{C_{st}} \times (\Gamma \cup \{\top\}) \times Q$ is the set of *pop transitions*,
- $\Delta_i \subseteq Q \times \Sigma_{int} \times \Phi(D) \times 2^{C_{st}} \times Q$ is the set of *internal transitions*.

We now describe how a nested VPTA \mathcal{A} behaves over an infinite timed word w . Assume that on reading the i -th position of w , the current state of \mathcal{A} is q , val_i^w is the event-clock valuation associated with w and i , $sval$ is the current valuation of the standard clocks in C_{st} , and $t = \tau_i - \tau_{i-1}$ is the time elapsed from the last transition (where $\tau_{-1} = 0$). If \mathcal{A} reads a call $c \in \Sigma_{call}$, it chooses a push transition of the form $(q, c, \theta, Res, q', \gamma) \in \Delta_c$ and pushes the symbol $\gamma \neq \top$ onto the stack. If \mathcal{A} reads a return $r \in \Sigma_{ret}$, it chooses a pop transition of the form $(q, r, \theta, Res, \gamma, q') \in \Delta_r$ such that γ is the symbol on top of the stack, and pops γ from the stack (if $\gamma = \top$, then γ is read but not removed). Finally, on reading an internal action $a \in \Sigma_{int}$, \mathcal{A} chooses an internal transition of the form $(q, a, \theta, Res, q') \in \Delta_i$, and, in this case, there is no operation on the stack. Moreover, in all the cases, the constraint θ of the chosen transition must be fulfilled by the valuation $(sval + t) \cup (val_i^w)|_C$, the control changes from q to q' , and all the standard clocks in Res are reset (i.e., the valuation of the standard clocks is updated to $(sval + t)[Res]$).

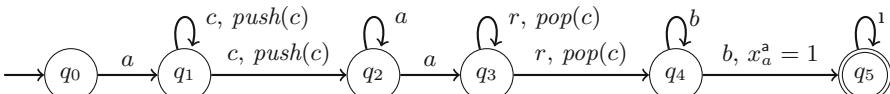
Formally, a configuration of \mathcal{A} is a triple $(q, \beta, sval)$, where $q \in Q$, $\beta \in \Gamma^* \cdot \{\top\}$ is a stack content, and $sval$ is a valuation over C_{st} . A run π of \mathcal{A} over $w = (\sigma, \tau)$ is an infinite sequence of configurations $\pi = (q_0, \beta_0, sval_0), (q_1, \beta_1, sval_1), \dots$ such that $q_0 \in Q_0$, $\beta_0 = \top$, $sval_0(z) = 0$ for all $z \in C_{st}$ (initialization requirement), and the following holds for all $i \geq 0$, where $t_i = \tau_i - \tau_{i-1}$ ($\tau_{-1} = 0$):

- **Push:** If $\sigma_i \in \Sigma_{call}$, then for some $(q_i, \sigma_i, \theta, Res, q_{i+1}, \gamma) \in \Delta_c$, $\beta_{i+1} = \gamma \cdot \beta_i$, $sval_{i+1} = (sval_i + t_i)[Res]$, and $(sval_i + t_i) \cup (val_i^w)|_C \models \theta$.
- **Pop:** If $\sigma_i \in \Sigma_{ret}$, then for some $(q_i, \sigma_i, \theta, Res, \gamma, q_{i+1}) \in \Delta_r$, $sval_{i+1} = (sval_i + t_i)[Res]$, $(sval_i + t_i) \cup (val_i^w)|_C \models \theta$, and either $\gamma \neq \top$ and $\beta_i = \gamma \cdot \beta_{i+1}$, or $\gamma = \beta_i = \beta_{i+1} = \top$.
- **Internal:** If $\sigma_i \in \Sigma_{int}$, then for some $(q_i, \sigma_i, \theta, Res, q_{i+1}) \in \Delta_i$, $\beta_{i+1} = \beta_i$, $sval_{i+1} = (sval_i + t_i)[Res]$, and $(sval_i + t_i) \cup (val_i^w)|_C \models \theta$.

The run π is *accepting* if there are infinitely many positions $i \geq 0$ such that $q_i \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of \mathcal{A} is the set of infinite timed words w over Σ such that there is an accepting run of \mathcal{A} on w . The *greatest constant* of \mathcal{A} , denoted $K_{\mathcal{A}}$, is the greatest natural number used as bound in some clock constraint of \mathcal{A} . For technical convenience, we also consider nested VPTA equipped with a *generalized Büchi acceptance condition* \mathcal{F} consisting of a family of sets of accepting states. In such a setting, a run π is accepting if for each Büchi component $F \in \mathcal{F}$, the run π visits infinitely often states in F .

A VPTA [15] corresponds to a nested VPTA whose set C of event clocks is empty. An ECNA is a nested VPTA whose set C_{st} of standard clocks is empty. For ECNA, we can omit the reset component Res from the transition function and the valuation component $sval$ from each configuration $(q, \beta, sval)$. Note the class of Event-Clock Visibly Pushdown Automata (ECVPA) [18] corresponds to the subclass of ECNA where abstract and caller event-clocks are disallowed. We also consider three additional subclasses of ECNA: *abstract predicting* ECNA (AP_ECNNA, for short) which do not use abstract recorder clocks and caller clocks, *abstract recording* ECNA (AR_ECNNA, for short) which do not use abstract predictor clocks and caller clocks, and *caller* ECNA (C_ECNNA, for short) which do not use abstract clocks. Note that these three subclasses of ECNA subsume ECVPA.

Example 1. Let us consider the AR_ECNNA depicted below, where $\Sigma_{call} = \{c\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{a, b, \text{i}\}$. The control part of the transition relation ensures that for each accepted word, the MAP visiting the b -position associated with the transition tr from q_4 to q_5 cannot visit the a -positions following the call positions. This implies that the abstract recorder constraint $x_a^a = 1$ associated with tr is fulfilled *only if* all the occurrences of calls c and returns r are matched. Hence, constraint $x_a^a = 1$ ensures that the accepted language, denoted by \mathcal{L}_T^{rec} ,



consists of all the timed words of the form $(\sigma, \tau) \cdot (\text{i}^\omega, \tau')$ such that σ is a well-matched word of the form $a \cdot c^n \cdot a^m \cdot r^n \cdot b^k$ with $n, m, k > 0$, and the time difference in (σ, τ) between the first and last symbols is 1, i.e. $\tau|_{\sigma|-1} - \tau_0 = 1$. The example shows that ECNA allow to express a meaningful real-time property of recursive systems, namely the ability of bounding the time required to perform an internal activity consisting of an unbounded number of returning recursive procedure calls. Similarly, it is easy to define an AP_ECNNA accepting the timed language, denoted by \mathcal{L}_T^{pred} , consisting of all the timed words of the

form $(\sigma, \tau) \cdot (1^\omega, \tau')$ such that σ is a well-matched word of the form $a^k \cdot c^n \cdot b^m \cdot r^n \cdot b$, with $n, m, k > 0$, and the time difference in (σ, τ) between the first and last symbol is 1. Finally, we consider the timed language $\mathcal{L}_T^{\text{caller}}$, which can be defined by a C_ECNA, consisting of the timed words of the form $(c, t_0) \cdot (\sigma, \tau) \cdot (1^\omega, \tau')$ such that σ is a well-matched word of the form $a \cdot c^n \cdot a^m \cdot r^n \cdot b^k$, with $n, m, k > 0$, and the time difference in $(c, t_0) \cdot (\sigma, \tau)$ between the first and last symbols is 1.

Closure properties of Büchi ECNA. As stated in the following theorem, the class of languages accepted by Büchi ECNA is closed under Boolean operations. The proof exploits a technique similar to that used in [18] to prove the analogous closure properties for ECVPA (for details, see Appendix A of [12]).

Theorem 1. *The class of ω -timed languages accepted by Büchi ECNA is closed under union, intersection, and complementation. In particular, given two Büchi ECNA $\mathcal{A} = (\Sigma, Q, Q_0, C, \Gamma \cup \{\top\}, \Delta, F)$ and $\mathcal{A}' = (\Sigma, Q', Q'_0, C', \Gamma' \cup \{\top\}, \Delta', F')$ over Σ , one can construct*

- a Büchi ECNA accepting $\mathcal{L}_T(\mathcal{A}) \cup \mathcal{L}_T(\mathcal{A}')$ with $|Q| + |Q'|$ states, $|\Gamma| + |\Gamma'| + 1$ stacks symbols, and greatest constant $\max(K_{\mathcal{A}}, K_{\mathcal{A}'})$;
- a Büchi ECNA accepting $\mathcal{L}_T(\mathcal{A}) \cap \mathcal{L}_T(\mathcal{A}')$ with $2|Q||Q'|$ states, $|\Gamma||\Gamma'|$ stacks symbols, and greatest constant $\max(K_{\mathcal{A}}, K_{\mathcal{A}'})$;
- a Büchi ECNA accepting the complement of $\mathcal{L}_T(\mathcal{A})$ with $2^{O(n^2)}$ states, $O(2^{O(n^2)} \cdot |\Sigma_{\text{call}}| \cdot |\text{Const}|^{O(|\Sigma|)})$ stack symbols, and greatest constant $K_{\mathcal{A}}$, where $n = |Q|$ and Const is the set of constants used in the clock constraints of \mathcal{A} .

Expressiveness results. We now summarize the expressiveness results for ECNA. First of all, the timed languages $\mathcal{L}_T^{\text{rec}}$, $\mathcal{L}_T^{\text{pred}}$, and $\mathcal{L}_T^{\text{caller}}$ considered in Example 1 and definable by AR_ECNA, AP_ECNA, and C_ECNA, respectively, can be used to prove that the three subclasses AR_ECNA, AP_ECNA, and C_ECNA of ECNA are mutually incomparable. Hence, these subclasses strictly include the class of ECVPA and are strictly included in ECNA. The incomparability result directly follows from Proposition 1 below, whose proof is in Appendix B of [12].

As for ECNA, we have that they are less expressive than Büchi VPTA. In fact, by Theorem 3 in Sect. 4, Büchi ECNA can be converted into equivalent Büchi VPTA. The inclusion is strict since, while Büchi ECNA are closed under complementation (Theorem 1), Büchi VPTA are not [15].

In [9], an equally-expressive extension of ECVPA over finite timed words, by means of a *timed* stack, is investigated. The Büchi version of such an extension can be trivially encoded in Büchi AR_ECNA. Moreover, the proof of Proposition 1 can also be used for showing that Büchi ECVPA with timed stack are less expressive than Büchi AR_ECNA, Büchi AP_ECNA, and Büchi C_ECNA.

The general picture of the expressiveness results is summarized by Theorem 2.

Proposition 1. *The language $\mathcal{L}_T^{\text{rec}}$ is not definable by Büchi ECNA which do not use abstract recorder clocks, $\mathcal{L}_T^{\text{pred}}$ is not definable by Büchi ECNA which*

do not use abstract predictor clocks, and $\mathcal{L}_T^{\text{caller}}$ is not definable by Büchi ECNA which do not use caller clocks. Moreover, the language $\mathcal{L}_T^{\text{rec}} \cup \mathcal{L}_T^{\text{pred}} \cup \mathcal{L}_T^{\text{caller}}$ is not definable by Büchi AR_ECNA, Büchi AP_ECNA and Büchi C_ECNA.

Theorem 2. *The classes AR_ECNA, AP_ECNA, and C_ECNA are mutually incomparable, and $\text{AP_ECNA} \cup \text{AR_ECNA} \cup \text{C_ECNA} \subset \text{ECNA}$. Moreover,*

- (1) $\text{ECVPA} \subset \text{AR_ECNA}$
- (2) $\text{ECVPA} \subset \text{AP_ECNA}$
- (3) $\text{ECVPA} \subset \text{C_ECNA}$
- (4) $\text{ECNA} \subset \text{VPTA}$

Note that the expressiveness results above also hold for finite timed words.

4 Decision Procedures for Büchi ECNA

In this section, we first investigate emptiness, universality, and language inclusion problems for Büchi ECNA. Then, we consider the *Visibly model-checking problem against Büchi ECNA*, i.e., given a *visibly pushdown timed system* \mathcal{S} over Σ (that is a Büchi VPTA where all the states are accepting) and a Büchi ECNA \mathcal{A} over Σ , the problem whether $\mathcal{L}_T(\mathcal{S}) \subseteq \mathcal{L}_T(\mathcal{A})$ hold. We establish that the above mentioned problems are decidable and EXPTIME-complete. The key intermediate result is an exponential-time translation of Büchi ECNA into language-equivalent generalized Büchi VPTA. More precisely, we show that event clocks in *nested VPTA* can be removed with a single exponential blow-up.

Theorem 3 (Removal of event clocks from nested VPTA). *Given a generalized Büchi nested VPTA \mathcal{A} , one can construct in singly exponential time a generalized Büchi VPTA \mathcal{A}' (which do not use event clocks) such that $\mathcal{L}_T(\mathcal{A}') = \mathcal{L}_T(\mathcal{A})$ and $K_{\mathcal{A}'} = K_{\mathcal{A}}$. Moreover, \mathcal{A}' has $n \cdot 2^{O(p \cdot |\Sigma|)}$ states and $m + O(p)$ clocks, where n is the number of \mathcal{A} -states, m is the number of standard \mathcal{A} -clocks, and p is the number of event-clock atomic constraints used by \mathcal{A} .*

In the following we sketch a proof of Theorem 3. Basically, the result follows from a sequence of transformation steps all preserving language equivalence. At each step, an event clock is replaced by a set of fresh standard clocks. To remove global event clocks we use the technique from [4]. Here, we focus on the removal of an *abstract predictor* clock y_b^a with $b \in \Sigma$, referring to Appendix D and E of [12] for the treatment of abstract recorder clocks and caller clocks, respectively.

Fix a generalized Büchi nested VPTA $\mathcal{A} = (\Sigma, Q, Q_0, C \cup C_{st}, \Gamma \cup \{\top\}, \Delta, \mathcal{F})$ such that $y_b^a \in C$. By exploiting nondeterminism, we can assume that for each transition tr of \mathcal{A} , there is exactly one atomic constraint $y_b^a \in I$ involving y_b^a used as conjunct in the clock constraint of tr . If $I \neq \{\perp\}$, then $y_b^a \in I$ is equivalent to a constraint of the form $y_b^a \succ \ell \wedge y_b^a \prec u$, where $\succ \in \{>, \geq\}$, $\prec \in \{<, \leq\}$, $\ell \in \mathbb{N}$, and $u \in \mathbb{N} \cup \{\infty\}$. We call $y_b^a \succ \ell$ (resp., $y_b^a \prec u$) a lower-bound (resp., upper-bound) constraint. Note that if $u = \infty$, the constraint $y_b^a \prec u$ is always fulfilled, but we include it to have a uniform notation. We construct a generalized Büchi nested VPTA $\mathcal{A}_{y_b^a}$ equivalent to \mathcal{A} whose set of event clocks is $C \setminus \{y_b^a\}$,

and whose set of standard clocks is $C_{st} \cup C_{new}$, where C_{new} consists of the fresh standard clocks $z_{\succ\ell}$ (resp., $z_{\prec u}$), for each lower-bound constraint $y_b^a \succ \ell$ (resp., upper-bound constraint $y_b^a \prec u$) of \mathcal{A} involving y_b^a .

We now report the basic ideas of the translation. Consider a lower-bound constraint $y_b^a \succ \ell$. Assume that a prediction $y_b^a \succ \ell$ is done by \mathcal{A} at position i of the input word for the first time. Then, the simulating automaton $\mathcal{A}_{y_b^a}$ exploits the standard clock $z_{\succ\ell}$ to check that the prediction holds by resetting it at position i . Moreover, if i is not a call (resp., i is a call), $\mathcal{A}_{y_b^a}$ carries the obligation $\succ\ell$ in its control state (resp., pushes the obligation $\succ\ell$ onto the stack) in order to check that the constraint $z_{\succ\ell} \succ \ell$ holds when the next b occurs at a position j_{check} along the MAP ν visiting position i . We observe that:

- if a new prediction $y_b^a \succ \ell$ is done by \mathcal{A} at a position $j > i$ of ν strictly preceding j_{check} , $\mathcal{A}_{y_b^a}$ resets the clock $z_{\succ\ell}$ at position j rewriting the old obligation. This is safe since the fulfillment of the lower-bound prediction $y_b^a \succ \ell$ at j guarantees that prediction $y_b^a \succ \ell$ is fulfilled at i along ν .
- If a call position $i_c \geq i$ occurs in ν before j_{check} , the next position of i_c in ν is the matching return i_r of i_c , and any MAP visiting a position $h \in [i_c + 1, i_r - 1]$ is finite and ends at a position $k < i_r$. Thus, the clock $z_{\succ\ell}$ can be safely reset to check the prediction $y_b^a \succ \ell$ raised in positions in $[i_c + 1, i_r - 1]$ since this check ensures that $z_{\succ\ell} \succ \ell$ holds at position j_{check} .

Thus, previous obligations on a constraint $y_b^a \succ \ell$ are always rewritten by more recent ones. At each position i , $\mathcal{A}_{y_b^a}$ records in its control state the lower-bound obligations for the current MAP ν (i.e., the MAP visiting the current position i). Whenever a call i_c occurs, the lower-bound obligations are pushed on the stack in order to be recovered at the matching return i_r . If $i_c + 1$ is not a return (i.e., $i_r \neq i_c + 1$), then $\mathcal{A}_{y_b^a}$ moves to a control state having an empty set of lower-bound obligations (position $i_c + 1$ starts the MAP visiting $i_c + 1$).

The treatment of an upper-bound constraint $y_b^a \prec u$ is symmetric. Whenever a prediction $y_b^a \prec u$ is done by \mathcal{A} at a position i , and the simulating automaton $\mathcal{A}_{y_b^a}$ has no obligation on the constraint $y_b^a \prec u$, $\mathcal{A}_{y_b^a}$ resets the standard clock $z_{\prec u}$. If i is not a call (resp., i is a call) the fresh obligation $(first, \prec u)$ is recorded in the control state (resp., $(first, \prec u)$ is pushed onto the stack). When, along the MAP ν visiting position i , the next b occurs at a position j_{check} , the constraint $z_{\prec u} \prec u$ is checked, and the obligation $(first, \prec u)$ is removed or confirmed (in the latter case, resetting the clock $z_{\prec u}$), depending on whether the prediction $y_b^a \prec u$ is asserted at position j_{check} or not. We observe that:

- if a new prediction $y_b^a \prec u$ occurs in a position $j > i$ of ν strictly preceding j_{check} , $\mathcal{A}_{y_b^a}$ simply ignores it (the clock $z_{\prec u}$ is not reset at position j) since checking the prediction $y_b^a \prec u$ at the previous position i guarantees the fulfillment of the prediction $y_b^a \prec u$ at the position $j > i$ along ν .
- If a call position $i_c \geq i$ occurs in ν before j_{check} , then all the predictions $y_b^a \prec u$ occurring in a MAP visiting a position $h \in [i_c + 1, i_r - 1]$, with $i_r \leq j_{check}$ being the matching-return of i_c , can be safely ignored (i.e., $z_{\prec u}$ is not reset there) since they are subsumed by the prediction at position i .

Thus, for new obligations on an upper-bound constraint $y_b^a \prec u$, the clock $z_{\prec u}$ is not reset. Whenever a call i_c occurs, the updated set O of upper-bound and lower-bound obligations is pushed onto the stack to be recovered at the matching return i_r of i_c . Moreover, if $i_c + 1$ is not a return (i.e., $i_r \neq i_c + 1$), then $\mathcal{A}_{y_b^a}$ moves to a control state where the set of lower-bound obligations is empty and the set of upper-bound obligations is obtained from O by replacing each upper-bound obligation $(f, \prec u)$, for $f \in \{\text{live}, \text{first}\}$, with the live obligation $(\text{live}, \prec u)$. The latter asserted at the initial position $i_c + 1$ of the MAP ν visiting $i_c + 1$ (note that ν ends at $i_r - 1$) is used by $\mathcal{A}_{y_b^a}$ to remember that the clock $z_{\prec u}$ cannot be reset along ν . Intuitively, live upper-bound obligations are propagated from the caller MAP to the called MAP . Note that fresh upper-bound obligations ($\text{first}, \prec u$) always refer to predictions done along the current MAP and, differently from the live upper-bound obligations, they can be removed when the next b occurs along the current MAP .

Extra technicalities are needed. At each position i , $\mathcal{A}_{y_b^a}$ guesses whether i is the last position of the current MAP (i.e., the MAP visiting i). For this, it keeps track in its control state of the guessed type (call, return, or internal symbol) of the next input symbol. In particular, when i is a call, $\mathcal{A}_{y_b^a}$ guesses whether it has a matching return. If not, $\mathcal{A}_{y_b^a}$ pushes onto the stack a special symbol, say *bad*, and the guess is correct iff the symbol is never popped from the stack. Conversely, $\mathcal{A}_{y_b^a}$ exploits a special proposition p_∞ whose Boolean value is carried in the control state: p_∞ does not hold at a position j of the input iff the MAP visiting j has a caller whose matching return exists. Note that p_∞ holds at infinitely many positions. The transition function of $\mathcal{A}_{y_b^a}$ ensures that the Boolean value of p_∞ is propagated consistently with the guesses. Doing so, the guesses about the matched calls are correct iff p_∞ is asserted infinitely often along a run. A Büchi component of $\mathcal{A}_{y_b^a}$ ensures this last requirement. Finally, we have to ensure that the lower-bound obligations and fresh upper-bound obligations at the current position are eventually checked, i.e., the current MAP eventually visits a b -position. For finite MAP , this can be ensured by the transition function of $\mathcal{A}_{y_b^a}$. For infinite MAP , we note that at most one infinite MAP ν exists along a word, and ν visits only positions where p_∞ holds. Moreover, each position i greater than the initial position i_0 of ν is either a ν -position, or a position where p_∞ does not hold. Thus, a Büchi component of $\mathcal{A}_{y_b^a}$ using proposition p_∞ ensures the b -liveness requirements along the unique infinite MAP (if any). Full details of the construction of $\mathcal{A}_{y_b^a}$ are in Appendix C of [12].

By exploiting Theorems 1 and 3, we establish the main result of the paper.

Theorem 4. *Emptiness, universality, and language inclusion for Büchi ECNA, and visibly model-checking against Büchi ECNA are EXPTIME-complete.*

Proof. For the upper bounds, first observe that by [1, 10] the emptiness problem of generalized Büchi VPTA is EXPTIME-complete and solvable in time $O(n^4 \cdot 2^{O(m \cdot \log Km)})$, where n is the number of states, m is the number of clocks, and K is the largest constant used in the clock constraints of the automaton (hence, the time complexity is polynomial in the number of states). Now, given two

Büchi ECNA \mathcal{A}_1 and \mathcal{A}_2 over Σ , checking whether $\mathcal{L}_T(\mathcal{A}_1) \subseteq \mathcal{L}_T(\mathcal{A}_2)$ reduces to check emptiness of the language $\mathcal{L}_T(\mathcal{A}_1) \cap \overline{\mathcal{L}_T(\mathcal{A}_2)}$. Similarly, given a Büchi VPTA \mathcal{S} where all the states are accepting and a Büchi ECNA \mathcal{A} over the same pushdown alphabet Σ , model-checking \mathcal{S} against \mathcal{A} reduces to check emptiness of the language $\mathcal{L}_T(\mathcal{S}) \cap \overline{\mathcal{L}_T(\mathcal{A})}$. Since Büchi VPTA are polynomial-time closed under intersection and universality can be reduced in linear-time to language inclusion, by the closure properties of Büchi ECNA (Theorem 1) and Theorem 3, membership in EXPTIME for the considered problems directly follow.

For the matching lower-bounds, the proof of EXPTIME-hardness for emptiness of Büchi VPTA can be easily adapted to the class of Büchi ECNA. For the other problems, the result directly follows from EXPTIME-hardness of the corresponding problems for Büchi VPA [5,6] which are subsumed by Büchi ECNA. \square

Conclusions. In this paper we have introduced and studied ECNA, a robust subclass of VPTA allowing to express meaningful non-regular timed properties of recursive systems. The closure under Boolean operations, and the decidability of languages inclusion and visibly model-checking makes ECNA amenable to specification and verification purposes. As future work, we plan to investigate suitable extensions of the Event Clock Temporal Logic introduced for ECA so that a logical counterpart for ECNA can be similarly recovered.

References

1. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: Proceedings of 27th LICS, pp. 35–44. IEEE Computer Society (2012)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoret. Comput. Sci. **126**(2), 183–235 (1994)
3. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_35
4. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: a determinizable class of timed automata. Theoret. Comput. Sci. **211**(1–2), 253–273 (1999)
5. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proceedings of 36th STOC, pp. 202–211. ACM (2004)
6. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM **56**(3), 16:1–16:43 (2009)
7. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
8. Benerecetti, M., Peron, A.: Timed recursive state machines: expressiveness and complexity. Theoret. Comput. Sci. **625**, 85–124 (2016)
9. Bhave, D., Dave, V., Krishna, S.N., Phawade, R., Trivedi, A.: A logical characterization for dense-time visibly pushdown automata. In: Dediu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) LATA 2016. LNCS, vol. 9618, pp. 89–101. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30000-9_7

10. Bouajjani, A., Echahed, R., Robbana, R.: On the automatic verification of systems with continuous variables and unbounded discrete data structures. In: Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1994. LNCS, vol. 999, pp. 64–85. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60472-3_4
11. Bozzelli, L., Murano, A., Peron, A.: Pushdown module checking. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 504–518. Springer, Heidelberg (2005). https://doi.org/10.1007/11591191_35
12. Bozzelli, L., Murano, A., Peron, A.: Event-clock nested automata (2017). <http://arxiv.org/abs/1711.08314>
13. Chatterjee, K., Ma, D., Majumdar, R., Zhao, T., Henzinger, T.A., Palsberg, J.: Stack size analysis for interrupt-driven programs. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694, pp. 109–126. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44898-5_7
14. Clemente, L., Lasota, S.: Timed pushdown automata revisited. In: Proceedings of 30th LICS, pp. 738–749. IEEE Computer Society (2015)
15. Emmi, M., Majumdar, R.: Decision problems for the verification of real-time software. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 200–211. Springer, Heidelberg (2006). https://doi.org/10.1007/11730637_17
16. Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown specifications. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS (LNAI), vol. 2514, pp. 262–277. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36078-6_18
17. Murano, A., Perelli, G.: Pushdown multi-agent system verification. In: Proceedings of IJCAI, pp. 1090–1097 (2015)
18. Van Tang, N., Ogawa, M.: Event-clock visibly pushdown automata. In: Nielsen, M., Kučera, A., Miltersen, P.B., Palamidessi, C., Tůma, P., Valencia, F. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 558–569. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-95891-8_50
19. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 306–324. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15643-4_23
20. Walukiewicz, I.: Pushdown processes: games and model checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 62–74. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61474-5_58



On the Synchronization of Planar Automata

J. Andres Montoya^(✉) and Christian Nolasco

Departamento de Matemáticas, Universidad Nacional de Colombia, Bogotá, Colombia
{jamontoyaa,cnolasco}@unal.edu.co
<http://www.unal.edu.co>

Abstract. Planar automata seems to be representative of the synchronizing behavior of deterministic finite state automata. We conjecture that Černy's conjecture holds true, if and only if, it holds true for planar automata. We provide new (and old) evidence concerning the conjectured Černy-universality of planar automata.

This work is related to the synchronization of deterministic finite state automata (DFAs, for short), and wherever we say automaton we are referring a DFA.

Let \mathcal{M} be a DFA, and let $\Sigma_{\mathcal{M}}$ be its input alphabet, we use the symbol $\Sigma_{\mathcal{M}}^*$ to denote the set of finite strings over the alphabet $\Sigma_{\mathcal{M}}$. The function $\widehat{\delta}_{\mathcal{M}} : \Sigma_{\mathcal{M}}^* \times Q_{\mathcal{M}} \rightarrow Q_{\mathcal{M}}$ is defined by the recursion:

$$\begin{aligned}\widehat{\delta}_{\mathcal{M}}(w_1, q) &= \delta_{\mathcal{M}}(w_1, q); \\ \widehat{\delta}_{\mathcal{M}}(w_1 \dots w_n, q) &= \delta_{\mathcal{M}}\left(w_n, \widehat{\delta}_{\mathcal{M}}(w_1 \dots w_{n-1}, q)\right),\end{aligned}$$

where $\delta_{\mathcal{M}}$ is the transition function of \mathcal{M} .

A *synchronizing string (reset word)* for \mathcal{M} is a string $w \in \Sigma_{\mathcal{M}}^*$ such that the equality $\widehat{\delta}_{\mathcal{M}}(w, p) = \widehat{\delta}_{\mathcal{M}}(w, q)$ holds for all $p, q \in Q_{\mathcal{M}}$. We say that the deterministic automaton \mathcal{M} is *synchronizing*, if and only if, there exists a synchronizing string for \mathcal{M} . Let \mathcal{M} be a synchronizing automaton, the *shortest reset length* of \mathcal{M} , denoted by $rl_{\mathcal{M}}$, is equal to the length of the shortest synchronizing strings for \mathcal{M} . It is easy to prove that $rl_{\mathcal{M}} \in O(|Q_{\mathcal{M}}|^3)$. Černy conjectured that $rl_{\mathcal{M}} \leq (|Q_{\mathcal{M}}| - 1)^2$ (see [5]). This conjecture is called *Černy's Conjecture* and it is considered the most important open problem in the combinatorial theory of finite state automata.

The universality conjecture for planar automata. It is well known that Černy's conjecture holds true for strongly connected synchronizing automata, if and only if, it holds true for synchronizing automata. Therefore, we say that the class of strongly connected automata is *Černy-universal*. We conjecture that the same is true for the class of planar automata. Let us discuss some facts that led us to formulate the *Černy-universality conjecture* for planar automata.

We are interested in some algorithmic problems related to the synchronization of DFA's. The algorithmic complexity of most of those problems is well

understood, and there are many deep results characterizing their intrinsic hardness (see for example [6, 8, 11] and the references therein). It happens that the same hardness results can be obtained for planar automata by noticing that their proofs for general automata work verbatim or with minor adjustments in the planar framework. Then, we have that the planar restrictions of the aforementioned algorithmic problems are as hard as the unrestricted versions. It suggests that the class of planar automata is an *universal class* with respect to the algorithmic hardness of synchronization.

It is also interesting to observe that all the sequences of slowly synchronizing automata registered in the literature are sequences of planar automata (see [1]). It is important to remark, at this point, that it is fairly easy to transform a sequence of slowly synchronizing planar automata into a sequence of slowly synchronizing non-planar automata. Suppose we have a sequence of slowly synchronizing planar automata, and suppose that all those automata are *quaternary* (the sizes of their input alphabets are all equal to 4), then it suffices to insert a copy of $\mathcal{K}_{3,3}$ into each one of the automata in the sequence while taking care of preserving the synchronizability. However, it seems that all the sequences of slowly synchronizing planar automata have a *planar core*.

The above two observations are the origin of our conjecture. In this work we present some new results that can be considered as additional evidence supporting the conjectured Černy-universality of planar automata. We think that the aforementioned results are interesting in their own right.

Organization of the work and contributions. This work contains a single section besides the introduction. We characterize the algorithmic hardness of some synchronization problems related to planar automata and we show that all those problems are as hard as the unrestricted versions. We get most of those results by simply noticing that their proofs for general automata work verbatim in the planar framework. We get a new hardness result for planar and unrestricted automata. The proof of this result for planar automata is not trivial. The aforementioned result is related to the parameterized complexity of synchronizing small sets of states, we prove that the corresponding algorithmic problem, as well as its planar restriction, are *WNL* complete (see [9]).

1 On the Algorithmic Hardness of Synchronizing Planar Automata

We investigate the synchronization of finite state automata focussing on the class of planar DFA's. A deterministic finite state automaton is planar, if and only if, its transition digraph is planar. Planar automata have been previously studied and it is known that there are regular languages which cannot be recognized by deterministic planar automata [4]. This last fact indicates that the class of planar automata is not universal with respect to recognition power. However, we conjecture that this restricted class is universal with respect to the hardness of synchronization. This conjecture motivates us to study the synchronization of planar automata. Consider the following two algorithmic problems:

Problem 1. (**Synch** [P]: Optimal synchronization of planar automata)

- Input: (\mathcal{M}, k) , where \mathcal{M} is a synchronizing planar automaton and k is a positive integer.
- Problem: Decide if there exists a synchronizing string for \mathcal{M} whose length is upperbounded by k .

Problem 2. (**ESynch** [P]: Deciding shortest reset length)

- Input: (\mathcal{M}, k) , where \mathcal{M} is a synchronizing planar automaton and k is a positive integer.
- Problem: Decide if the shortest reset length of \mathcal{M} is equal to k .

It is easy to prove the following theorem.

Theorem 1. *We have that*

1. *The problem **Synch** [P] is NP complete.*
2. *Given $\varepsilon > 0$, it is NP hard to approximate the shortest reset length of planar synchronizing automata within the ratio $O(n^{1-\varepsilon})$, while the ratio $O(n)$ can be achieved in polynomial time.*
3. **ESynch** [P] is complete for the class DP.

Proof. Proofs of the same results but for unrestricted automata are presented in the Refs. [6, 8, 11]. It is enough to observe that the aforementioned proofs work verbatim or with minor adjustments for planar automata.

In next section we add a further hardness result.

1.1 The Parameterized Hardness of Synchronizing Small Sets of States

In this section we characterize the parameterized complexity of *subset synchronization* [10]. We refer the reader to [7] for a pedagogical introduction to the basics of parameterized complexity.

Let \mathcal{M} be a DFA, and let $q_1, \dots, q_k \in Q_{\mathcal{M}}$, a *synchronizing string for these k states* is a string w such that the equality $\widehat{\delta}_{\mathcal{M}}(w, q_i) = \widehat{\delta}_{\mathcal{M}}(w, q_j)$ holds for all $i, j \leq k$. In the later case we say that w synchronizes the subset $\{q_1, \dots, q_k\}$.

We think that subset synchronization is a powerful concept that allows one to model some different types of discrete dynamics. Suppose, for instance, that we have a troop of agents scattered over a territory and we want to broadcast an instruction, the same one for all the agents, which must lead the agents to a common site on the territory. If the territory is the transition digraph of a synchronizing automaton and we do not know the initial locations of the agents, then we must broadcast a reset word for the underlying automaton. On the other hand, if we know the initial locations q_1, \dots, q_k , then we must broadcast a synchronizing string for these k states. Notice that a shortest synchronizing string for the states q_1, \dots, q_k could be very much shorter than any shortest

synchronizing string for the whole automaton. It could happen if the set of states to be synchronized is very much smaller than $Q_{\mathcal{M}}$. We consider that planar automata constitutes a set of natural scenarios for some of the most representative instances of subset synchronization. Notice that planar digraphs are the discrete versions of two-dimensional territories.

Let $p\text{-Synch}[P]$ be the parameterized problem defined by:

Problem 3. ($p\text{-synch}[P]$: Parameterized synchronization of planar automata)

- *Input:* $((\mathcal{M}, \{q_1, \dots, q_k\}, l), k)$, where \mathcal{M} is a synchronizing planar automaton and $q_1, \dots, q_k \in Q_{\mathcal{M}}$.
- *Parameter:* k .
- *Problem:* Decide if there exists a synchronizing string of length l for the states q_1, \dots, q_k .

Recall that a parameterized problem is *fixed-parameter tractable*, if and only if, it can be solved in time $O(f(k) \cdot n^c)$, for some function f and some constant c (see [7]). Is $p\text{-Synch}[P]$ fixed-parameter tractable? We prove that the problem $p\text{-Synch}[P]$ is WNL complete. The class WNL is supposed to be the parameterized analogue of PSPACE [9]. This class is located above of the W-hierarchy, and hence we have that $p\text{-Synch}[P]$ is $W[t]$ hard for all $t \geq 1$. The class WNL is defined as the closure under *fpt-reductions* (see [7]) of the following problem.

Problem 4. ($p\text{-}WNL$: Deciding acceptance of parameterized space bounded computations)

- *Input:* $((\mathcal{M}, t), k)$, where \mathcal{M} is a nondeterministic Turing machine, t is a positive integer given in unary, and $k \geq 1$.
- *Parameter:* k .
- *Problem:* Decides if \mathcal{M} accepts the empty input in at most t steps and checking at most k cells.

Let L be a parameterized problem. Suppose we want to check that L belongs to WNL . It is enough to exhibit a nondeterministic RAM \mathcal{N} accepting L and satisfying the following constraint: Let $R(X, k)$ be the maximum number of registers used by \mathcal{N} , along its computations on input (X, k) , the quantity $R(X, k)$ is bounded above by a function $r(k)$ that only depends on the parameter k (see [9]).

It is easy to prove that $p\text{-synch}[P]$ belongs to WNL . It is harder to prove that $p\text{-synch}[P]$ is WNL hard. We prove the later by exhibiting a fpt Turing reduction of *The parameterized longest common subsequence problem* in $p\text{-Synch}[P]$. The parameterized longest common subsequence problem, denoted by $p\text{-LCS}$, is the parameterized problem defined by:

Problem 5. (*p-LCS*: Parameterized longest common subsequence)

- Input: $((\{w_1, \dots, w_k\}, \Sigma, m), k)$, where Σ is a finite alphabet, $w_1, \dots, w_k \in \Sigma^*$ and m is a positive integer.
- Parameter: k .
- Problem: Decide if there exists a string $w \in \Sigma^*$ such that for all $i \leq k$ the string w is a substring of w_i , and such that $|w| = m$.

Guillemot [9] proved that *p-LCS* is hard for *WNL*, and it means that our reduction suffices.

Theorem 2. *The problems p-Synch [P] and p-Synch are WNL complete.*

Proof. First we check that *p-Synch* belongs to *WNL*. To this end we construct a suitable nondeterministic RAM accepting the problem *p-Synch*. The machine works, on input $((\mathcal{M}, \{q_1, \dots, q_k\}, l), k)$, as follows:

The machine stores in the first k registers a tuple of positive integers (s_1, \dots, s_k) such that for all $i \leq k$ the inequality $s_i \leq |Q_{\mathcal{M}}|$ holds. It begins with $(0, \dots, 0)$, and then it overwrites (q_1, \dots, q_k) . Set $(s_1^1, \dots, s_k^1) = (q_1, \dots, q_k)$, for all $i \leq l$ the machine nondeterministically chooses a tuple $(s_1^{i+1}, \dots, s_k^{i+1})$ which can (over)write on the first k registers, if and only if, there exists $a \in \Sigma_{\mathcal{M}}$ such that the equality $\delta_{\mathcal{M}}(a, s_j^i) = s_j^{i+1}$ holds for all $j \leq k$. The machine accepts, if and only if, the entries of the last tuple are all equal.

We get that *p-Synch* [P] and *p-Synch* belongs to *WNL*. It remains to be proved that *p-Synch* [P] is *WNL* hard.

First we prove that *p-LCS* is *fpt* many-one reducible to *p-Synch*, and then we prove that *p-Synch* is *fpt* Turing reducible to *p-Synch* [P].

First stage (Reducing *p-LCS* to *p-Synch*).

Let $X = ((\{w_1, \dots, w_k\}, \Sigma, m), k)$ be an instance of *p-LCS*. Let $i \leq k$, we use Baeza-Yates construction (see [2]) to compute in polynomial time a DFA, say \mathcal{M}_i , that accepts the language constituted by all the subsequences of w_i .

Notice that for all $i \leq k$ we are using the automaton \mathcal{M}_i as a language acceptor. The later fact implies that for all $i \leq k$ there exists a *marked state* (the initial state of \mathcal{M}_i) which we denote with the symbol q_0^i . Moreover, we have that for all $i \leq k$ there exists a nonempty subset of Q_i , denoted with the symbol A_i , and which is equal to the set of accepting states of automaton \mathcal{M}_i .

We use the set $\{\mathcal{M}_i : i \leq k\}$ to define an automaton $\mathcal{M} = (\Omega, Q, \delta)$ in the following way:

1. $\Omega = \Sigma \cup \{d\}$, where $d \notin \Sigma$.
2. $Q = \left(\bigsqcup_{i \leq k} Q_i \right) \sqcup \{q, p_1, \dots, p_{m+1}\}$, where \sqcup denotes disjoint union, and given $i \leq k$, the symbol Q_i denotes the set of states of the automaton \mathcal{M}_i . Moreover, we have that $q, p_1, \dots, p_{m+1} \notin \bigsqcup_{i \leq k} Q_i$.

3. The transition function of \mathcal{M} , which we denote with the symbol δ , is defined as follows

$$\delta(a, p) = \begin{cases} \delta_i(a, p), & \text{if } p \in Q_i \text{ and } a \neq d \\ q, & \text{if } p \in \bigsqcup_{i \leq k} A_i \text{ and } a = d \\ p_1, & \text{if } p \in (Q_i \setminus A_i) \text{ and } a = d \\ q, & \text{if } p = q \\ p_{j+1}, & \text{if } p = p_j, j < m+1 \text{ and } a \in \Sigma \\ p_1, & \text{if } p = p_j, j < m+1, \text{ and } a = d \\ q, & \text{if } p = p_{m+1} \text{ and } a = d \\ p_1, & \text{if } p = p_{m+1} \text{ and } a \neq d \end{cases}$$

Let $Y(X)$ be equal to $((\mathcal{M}, \{q_0^1, \dots, q_0^k, p_1\}, m+1), k+1)$, it is the output of our reduction. We check that $X \in p\text{-LCS}$, if and only if, the states q_0^1, \dots, q_0^k, p_1 can be synchronized in time $m+1$.

Suppose that $((\{w_1, \dots, w_k\}, \Sigma, m), k)$ is a positive instance of $p\text{-LCS}$. Let $w \in \Sigma^m$ such that w is a subsequence of each one of the strings w_1, \dots, w_k . Let $u = wd \in (\Sigma \cup \{d\})^{m+1}$, we claim that u synchronizes the states q_0^1, \dots, q_0^k, p_1 . We have that for all $i \leq k$ the state $\widehat{\delta(w, q_0^i)}$ belongs to A_i . Moreover, we have that $\widehat{\delta(w, p_1)} = p_{m+1}$. Then, given $r \in \{\widehat{\delta(w, x)} : x \in \{q_0^1, \dots, q_0^k, p_1\}\}$ the equality $\delta(d, x) = q$ holds. Thus, we get that u is a string that sends the states q_0^1, \dots, q_0^k, p_1 to the sink q . We conclude that the states q_0^1, \dots, q_0^k, p_1 can be synchronized in time $m+1$.

Now suppose that the states q_0^1, \dots, q_0^k, p_1 can be synchronized in time $m+1$ by a string $u \in (\Sigma \cup \{d\})^{m+1}$. We observe that the synchronizing state must be equal to the sink q . We also observe that the distance between p_1 and q is equal to $m+1$, and it implies that the synchronization of the states q_0^1, \dots, q_0^k, p_1 cannot be achieved with strings of length smaller than $m+1$. Given $i \leq m$, we have that at least one of the tokens is not placed on q at time i . The latter implies that the last character of u must be equal to d . Let $u = wd$. If $w \notin \Sigma^m$, then $\widehat{\delta(w, p_1)} \neq p_{m+1}$ and u does not synchronize the set $\{q_0^1, \dots, q_0^k, p_1\}$. We can conclude that $w \in \Sigma^m$. Suppose that there exists $i \leq k$ such that $\widehat{\delta(w, q_0^i)} \notin A_i$, we get that $\widehat{\delta(wd, q_0^i)} \neq q$ and the string $u = wd$ does not synchronize. Thus, we have that for all $i \leq k$ the Baeza-Yates automaton \mathcal{M}_i accepts the string w . The later fact implies that w is a subsequence of each one of the strings in the set $\{w_1, \dots, w_k\}$, and we can conclude that X is a positive instance of $p\text{-LCS}$.

It is interesting to observe that we used the states p_1, \dots, p_{m+1} to built a *clock*, a gadget that was used to prevent that synchronization occurs earlier than time $m+1$. It is not a surprise if we have to use clocks when we want to synchronize.

It happens that Baeza-Yates construction is non-planar, and hence if $Y(X)$ is equal to $((\mathcal{M}, \{q_0^1, \dots, q_0^k, p_1\}, m+1), k+1)$, it could occur that the automaton \mathcal{M} is a non-planar one. Therefore, we have to proceed with the second reduction

Second stage (*Reducing p-Synch to p-Synch [P]*).

Let $p\text{-Synch}$ [2] be the restriction of $p\text{-Synch}$ to the set of instances

$$\{((\mathcal{M}, \{q_1, \dots, q_k\}, l), k) : \mathcal{M} \text{ is a binary synchronizing automaton}\}.$$

The construction used in [3] yields a *fpt* many-one reduction of the problem $p\text{-Synch}$ in its restriction $p\text{-Synch}$ [2]. We exhibit a *fpt* Turing reduction of the problem $p\text{-Synch}$ [2] in the problem $p\text{-Synch}$ [P].

Let $((\mathcal{M}, \{q_1, \dots, q_k\}, m), k)$ be an instance of $p\text{-Synch}$ [2]. A planar drawing of the automaton \mathcal{M} is an embedding of its transition digraph in \mathbb{R}^2 , and which satisfies the following three constraints:

- Edges are mapped on simple curves.
- No three edges meet at a common crossing.
- Two edges meet at most once.

Let $\mathcal{M} = (\{a, b\}, Q, \delta)$. A Planar drawing of \mathcal{M} can be computed in polynomial time in $|Q|$, and if the automaton \mathcal{M} is a planar one, then the computed drawing can be chosen to be a planar embedding (a drawing without crossings).

Suppose that \mathcal{M} is non-planar, and let ρ be a planar drawing of \mathcal{M} . Let e be an edge (transition) of \mathcal{M} , we use the symbol $cr_\rho(e)$ to denote the number of crossings involving edge e . We have that for all ρ and for all e the inequality $cr_\rho(e) \leq 2|Q|$ holds. The later inequality follows from the following fact: Automaton \mathcal{M} has exactly $2|Q|$ transitions (edges), and given that any two edges meet at most once, each one of the edges can get involved in at most $2|Q| - 1$ crossings.

To begin with the reduction we compute a planar drawing of \mathcal{M} , say ρ , and we use ρ to compute a planar automaton \mathcal{N}_0 . The computation of \mathcal{N}_0 goes as follows:

1. The input alphabet of \mathcal{N}_0 is equal to $\{a, b\} \times \{0, 1\}$.
2. The set of states of automaton \mathcal{N}_0 contains the set $\{\rho(p) : p \in Q\}$. It is important to remark that we will have to use $16|Q|^2 - |Q|$ additional states.
3. Given e , a transition of \mathcal{M} , edge e is represented in \mathcal{N} by a path of length $8|Q|$. To construct the later path we subdivide $\rho(e)$ into $2|Q|$ disjoint segments. The segments can be chosen to be connected, with a nonempty interior, and such that each one of the $cr_\rho(e)$ crossings involving e is an inner point of one of those intervals. Moreover, we can choose the $2|Q|$ segments in such a way that each one of them contains at most one crossing. Notice that we need no more than $2|Q|$ segments because we have no more than $2|Q|$ crossings. Each one of those $2|Q|$ segments is in turn subdivided into four subsegments. Those four segments are used to built the gadgets that will allow us to eliminate the crossings involving the edge e . Thus, suppose that e is directed from p to q and let $1 \leq i \leq 2|Q|$. We choose four points in the i -th segment of $\rho(e)$, let $v_1^{e,i}, v_2^{e,i}, v_3^{e,i}$ and $v_4^{e,i}$ be those four points, and suppose that $v_1^{e,i}$ is the *start-point* of the segment (the point that is closest to $\rho(p)$). We also suppose that $v_2^{e,i}$ lies between $v_1^{e,i}$ and $v_3^{e,i}$, while $v_3^{e,i}$ lies between $v_2^{e,i}$ and $v_4^{e,i}$. Moreover, the point $v_4^{e,i}$ belongs to the interior of the i -th segment. If $i = 1$, we have

that $v_1^{e,1} = \rho(p)$. If $i = 2|Q|$, we have that $v_1^{e,2|Q|+1} = \rho(q)$. All the points in the set

$$\left\{ v_k^{e,i} : k \leq 4, i \leq 2|Q| \text{ and } e \text{ is an edge of } \mathcal{M} \right\}$$

are states of the automaton \mathcal{N}_0 that we want to construct. So far we have only computed a subdivision of \mathcal{M} .

4. Let $i \leq 2|Q|$, let I be the i -th segment and let $r(e)_1^i, r(e)_2^i, r(e)_3^i, r(e)_4^i$ be the four subsegments of I . Each one of those four segments become edges of \mathcal{N} . Given $j \leq 3$, the edge $r(e)_j^i$ is directed from $v_j^{e,i}$ to $v_{j+1}^{e,i}$, while the edge $r(e)_4^i$ is directed from $v_4^{e,i}$ to $v_1^{e,i+1}$. Moreover, we assign to those four edges the labels $(c, 0), (c, 1), (c, 1)$ and $(c, 0)$, where c is the label assigned to edge e in \mathcal{M} .
5. Now suppose that edges e and f meet at some point x . There exists $i, j \leq 2|Q|$ such that x lies on the i -th segment of e , and x lies on the j -th segment of f . We can choose the points $v_1^{e,i}, v_2^{e,i}, v_3^{e,i}$ and $v_4^{e,i}$, and the points $v_1^{f,j}, v_2^{f,j}, v_3^{f,j}$ and $v_4^{f,j}$ in such a way that:
 - The equalities $v_3^{e,i} = v_2^{f,j}$ and $v_4^{e,i} = v_3^{f,j}$ hold.
 - If the labels of e and f are equal, the segments $r(e)_3^i$ and $r(f)_2^j$ are equal, otherwise they meet each other only at the points $v_3^{e,i}$ and $v_4^{e,i}$.
 - If the labels of e and f are equal, the point x lies in the interior the segment $r(e)_3^i$, otherwise it lies in the interior of the simple closed curve formed by $r(e)_3^i$ and $r(f)_2^j$.

Notice that the above construction allows us to eliminate the crossing x .

It is interesting to observe that the later construction is somewhat asymmetrical. The asymmetric nature of the construction is not a problem: Given two edges that meet each other at some point x , it makes not difference which edge plays the role of e and which one plays the role of f . Suppose we are constructing the automaton \mathcal{N}_0 and we are given a pair of edges of \mathcal{M} that meet each other, we can choose at random the role played by each one of the two edges. Moreover, the asymmetrical nature of this construction is used to prevent that tokens being synchronized on \mathcal{N}_0 use the crossings of \mathcal{M} to find shortcuts (see Fig. 1 below).

6. Recall that we are trying to draw a planar automaton \mathcal{N}_0 . To this end, we use the set of points

$$P = \left\{ v_j^{e,i} : i \leq 2|Q|, j = 1, 2, 3, 4 \text{ and } e \text{ is an edge of } \mathcal{M} \right\},$$

and the set of edges

$$E = \left\{ r(e)_j^i : i \leq 2|Q|, j = 1, 2, 3, 4 \text{ and } e \text{ is an edge of } \mathcal{M} \right\}.$$

Notice that $|P| = 16|Q|^2$, and $|E| = 64|Q|^2$. If we add some loops we get a planar synchronizing automaton denoted by \mathcal{N}_0 . Let us check that \mathcal{N}_0 is synchronizing. Let $f : \{a, b\}^* \rightarrow (\{a, b\} \times \{0, 1\})^*$ be the homomorphism defined

by: Given $k \geq 1$ and given $w_1 \cdots w_k \in \{a, b\}^*$ we have that $f(w_1 \cdots w_k)$ is equal to

$$((w_1, 0) (w_1, 1) (w_1, 1) (w_1, 0))^{|Q|} \cdots ((w_k, 0) (w_k, 1) (w_k, 1) (w_k, 0))^{|Q|}.$$

If the string w synchronizes the automaton \mathcal{M} , then $f(w)$ is a reset word for \mathcal{N}_0 . Moreover, if the string w synchronizes the states q_1, \dots, q_k , we have that $f(w)$ synchronizes the states $\rho(q_1), \dots, \rho(q_k)$. Notice that the later property of \mathcal{N}_0 is almost all that we need. However, the later property is not really enough, notice that we also have to prevent the existence of short synchronizing strings that do not belong to the image of f . The later can happen because of the following: Edges of \mathcal{M} are represented in \mathcal{N}_0 by paths of length $8|Q|$, and the tokens moving on \mathcal{N}_0 can prematurely leave those paths and find shortcuts thanks to the crossings in \mathcal{M} . We use clocks to avoid the later possibility.

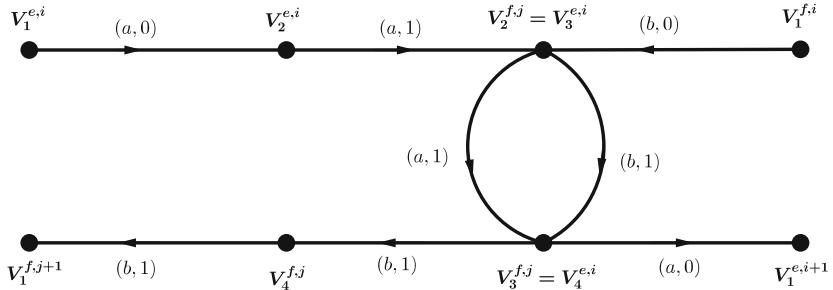


Fig. 1. Elimination of the crossing.

Let $m \geq 1$ and let p be an state of \mathcal{M} , we use the symbol $\mathcal{N}_{m,p}$ to denote the planar automaton that is obtained from \mathcal{N}_0 by attaching to node $\rho(p)$ a planar digraph that we call $\mathcal{C}_{m,p}$. The gadget $\mathcal{C}_{m,p}$ is computed from m and $|Q|$ (it does not depend on p), and it is used as a clock for the synchronization process. The clock $\mathcal{C}_{m,p}$ is used to force two things:

- Optimal synchronization occurs at node $\rho(p)$ (any other node is excluded).
- Synchronization cannot occur earlier than time $8m|Q|$.

The construction of $\mathcal{C}_{m,p}$ must fulfill the following additional condition:

The states $\{q_1, \dots, q_k\}$ can be send to state p using m characters, if and only if, the states $\rho(q_1), \dots, \rho(q_k)$ and the *clock-state* $w(p)$ can be synchronized in time $8m|Q|$.

If we succeed with the construction of the automata $\{\mathcal{N}_{m,p} : p \in Q\}$, then we can use this small set of automata to define our Turing reduction. If we want to know whether the states $\{q_1, \dots, q_k\}$ can be synchronized in time m , it suffices if we make exactly $|Q|$ queries: Given $p \in Q$ we ask whether the states $\rho(q_1), \dots, \rho(q_k), w(p) \in Q_{\mathcal{N}_{m,p}}$ can be synchronized at state $\rho(p)$ in time $8m|Q|$.

The clock $\mathcal{C}_{m,p}$ is constituted by m segments. Each one of those segments is constituted by two directed paths of length $8|Q|$, which meet only at the start-node (the start-node of both paths is the same, and it is the single common node). Let $\lambda \leq m$, the λ -th segment is constituted by the states

$$\left\{ p_{j,x}^{i,\lambda} : i \leq 2|Q|; j = 1, 2, 3, 4; x = a, b \right\},$$

where $p_{1,a}^{1,\lambda} = p_{1,b}^{1,\lambda} = p_1^{1,\lambda}$ (the start nodes are equal). If $\lambda = 1$, we set

$$p_{1,a}^{1,1} = p_{1,b}^{1,1} = w(p),$$

and we call this later state the *clock-state* of $\mathcal{N}_{m,p}$.

Given $i \leq 2|Q| - 1$ and given $x = a, b$, we add the edge $(p_{1,x}^{i,\lambda}, p_{2,x}^{i,\lambda})$ and we label it with the letter $(x, 0)$. We also add the edges $(p_{2,x}^{i,\lambda}, p_{3,x}^{i,\lambda})$ and $(p_{3,x}^{i,\lambda}, p_{4,x}^{i,\lambda})$ and we label them with the letter $(x, 1)$. Moreover we add the edge $(p_{4,x}^{i,\lambda}, p_{1,x}^{i+1,\lambda})$ and we label it with the letter $(x, 0)$. We glue together the λ -th segment and the $\lambda + 1$ -th by adding the edges $(p_{4,a}^{2|Q|,\lambda}, p_1^{1,\lambda+1})$ and $(p_{4,b}^{2|Q|,\lambda}, p_1^{1,\lambda+1})$, we label them with the letters $(a, 0)$ and $(b, 0)$ (respectively).

We embed $\mathcal{C}_{m,p}$ in the plane in such a way that, after adding the edges $(p_{4,a}^{2|Q|,m}, \rho(p))$ and $(p_{4,b}^{2|Q|,m}, \rho(p))$, the whole construction becomes planar. To achieve the later, it suffices to use a miniaturized copy of $\mathcal{C}_{m,p}$ that can be inserted without crossings into a small neighborhood of $\rho(p)$. To finish with the construction we add the necessary loops to obtain a full transition function.

We observe that the planar automaton $\mathcal{N}_{m,p}$ can be constructed in fpt time. We also have that $\mathcal{N}_{m,p}$ is synchronizing. Let us check the later. Automaton \mathcal{N}_0 could be synchronized before the insertion of $\mathcal{C}_{m,p}$. Then, it suffices to send all the states in $\mathcal{C}_{m,p}$ to the state $\rho(p)$, and then synchronize the states that are out of $\mathcal{C}_{m,p}$. The former is quite easy to achieve, while the later corresponds to synchronize the automaton \mathcal{N}_0 .

It only remains to be proved that the states q_1, \dots, q_k can be simultaneously sent to the state p using a string of length m , if and only if, the states $\rho(q_1), \dots, \rho(q_k), w(p) \in Q_{\mathcal{N}_{m,p}}$ can be synchronized in time $8m|Q|$.

Suppose that $w \in \{a, b\}^m$ sends the states q_1, \dots, q_k to the state p . Then, the string $f(w)$ synchronizes the states $\rho(q_1), \dots, \rho(q_k), w(p)$. Now suppose that $\rho(q_1), \dots, \rho(q_k), w(p)$ can be synchronized in time $8m|Q|$, and let $W \in (\{a, b\} \times \{0, 1\})^{8m|Q|}$ be a synchronizing string for those states. String W sends the state $w(p)$ to $\rho(p)$. Notice that the minimal paths connecting those two states are labeled by strings that belong to the image of f . Thus, we have that there exists $w \in \{a, b\}^m$ such that the equality $W = f(w)$ holds. It remains to be proved that w synchronizes the states q_1, \dots, q_k . To prove the later it is enough to show the following:

Claim. *Let $e = (p, q)$ be an edge of \mathcal{M} that is labeled with the letter $x \in \{a, b\}$, and let $i \leq m$. Suppose that we are moving the tokens that were placed at states*

$\rho(q_1), \dots, \rho(q_k), w(p)$ according to the synchronizing string $f(w)$, and suppose that at time $8i|Q| + 1$ a token is moved from $\rho(p) = v_1^{e,1}$ to $v_2^{e,1}$. Then, at time $8(i+1)|Q|$ the same token is reaching the state $\rho(q)$.

Proof of the claim. Suppose that the claim is false. Then, we have that while reading the substring $W[8i|Q| + 1, \dots, 8(i+1)|Q|]$ the token prematurely left the path that represents the edge e . It could only happen at one of the crossings involving e . Then, there exists f , an edge of \mathcal{M} , and there exist $k, l \leq 2|Q|$ such that the k -th segment of e and the l -th segment of f meet each other. We suppose that our *lost* token left the path representing e at this crossing. Let us also suppose that the equalities

$$v_3^{e,k} = v_2^{f,l} \text{ and } v_4^{e,k} = v_3^{f,l}$$

hold. Notice that we are using, for the first time, the asymmetric architecture of the crossing-gadgets. The lost token had to use the character

$$W[8i|Q| + 4(k-1) + 4]$$

to leave the e -path, choosing edge $(v_3^{f,l}, v_4^{f,l})$ instead of edge $(v_4^{e,k}, v_1^{e,k+1})$. We observe that the later is not possible: There exist $x, y \in \{a, b\}$ such that $W[8i|Q| + 4(k-1) + 3] = (x, 0)$, the label of $(v_4^{e,k}, v_1^{e,k+1})$ is equal to $(x, 0)$, and the label of $(v_3^{f,l}, v_4^{f,l})$ is equal to $(y, 1)$. We can conclude that tokens cannot prematurely leave the edges of \mathcal{M} and the claim is proved.

Given the automaton \mathcal{M} and given $S = \{q_1, \dots, q_k\}$, we use the symbol $\mathcal{I}_{p,m,S}$ to denote the tuple

$$(\mathcal{N}_{m,p}, \{\rho(q_1), \dots, \rho(q_k), w(p)\}, 8m|Q|).$$

We have that S can be synchronized in time m , if and only if, there exists $p \in Q$ such that the states $\rho(q_1), \dots, \rho(q_k), w(p)$ of the automaton $\mathcal{N}_{m,p}$ can be synchronized in time $8m|Q|$. Altogether we have a *true table* Turing reduction of p -*Synch* [2] in p -*Synch* [P] which can be computed in *fpt* time. The theorem is proved.

Remark 1. Let us use the symbol p -*Synch* [$P, 4$] to denote the restriction of p -*Synch* [P] to the class of planar automata defined over a four letter alphabet. The above reduction shows that p -*Synch* [$P, 4$] is *WNL* complete.

Acknowledgement. The second author would like to thank the support provided by Universidad Nacional de Colombia through the project Hermes 8943 (32083).

References

1. Ananichev, D., Gusev, V., Volkov, M.: Slowly synchronizing automata and digraphs. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 55–65. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_7
2. Baeza-Yates, R.: Searching subsequences. *Theor. Comput. Sci.* **78**(2), 363–376 (1991)
3. Berlinkov, M.V.: On two algorithmic problems about synchronizing automata. In: Shur, A.M., Volkov, M.V. (eds.) DLT 2014. LNCS, vol. 8633, pp. 61–67. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09698-8_6
4. Book, R., Chandra, A.: Inherently nonplanar automata. *Acta Inf.* **6**, 89–94 (1976)
5. Černý, J.: Poznámka k homogénnym experimentom s konečnymi automatmi. *Mat. fyz. cas SAV* **14**, 208–215 (1964)
6. Eppstein, D.: Reset sequences for monotonic automata. *SIAM J. Comput.* **19**, 500–510 (1990)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006). <https://doi.org/10.1007/3-540-29953-X>
8. Gawrychowski, P., Straszak, D.: Strong inapproximability of the shortest reset word. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9234, pp. 243–255. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48057-1_19
9. Guillemot, S.: Parameterized complexity and approximability of the longest compatible sequence problem. *Discret. Optim.* **8**(1), 50–60 (2011)
10. Montoya, J., Nolasco, C.: On the synchronization of small sets of states. *Appl. Math. Sci.* **11**(44), 2151–2173 (2017)
11. Olschewski, J., Ummels, M.: The complexity of finding reset words in finite automata. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 568–579. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_50



Descriptional and Computational Complexity of the Circuit Representation of Finite Automata

Māris Valdats^(✉)

Faculty of Computing, University of Latvia, Raiņa Bulv. 19, Riga, Latvia
maris@kautkur.lv

Abstract. In this paper we continue to investigate the complexity of the circuit representation of DFA—BC-complexity. We compare it with nondeterministic state complexity, obtain upper and lower bounds which differ only by a factor of 4 for a Binary input alphabet. Also we prove that many simple operations (determining if a state is reachable or if an automaton is minimal) are PSPACE-complete for DFA given in circuit representation.

1 Introduction

As finite automata is one of the most popular models of computation, state complexity is by far the most popular complexity measure of finite automata. It serves well in the structural language theory, however it does not always reflect the complexity (or simplicity) of automata implementation. In this paper we consider another complexity measure of automata: BC-complexity, that tries to capture the details of automata implementation.

As state complexity is related to the state minimization problem, BC-complexity is related to the *state assignment* problem [1]. State assignment is a classical problem in the synthesis of finite state machines—how to encode states of an automaton into a vector of binary memory elements to have the “simplest” switching function as a transition function for this automaton.

It is a well-studied problem but only from the functional optimization point of view. A lot of papers and books have been devoted to methods of optimal state assignment with a respect to different notions of “simplest” switching function. Standard optimization methods try to minimize the dependencies among state variables that leads to effective implementation of the transition function as a Boolean circuit [1,2], but there are other interpretations of simplicity, for example, the minimization of the average switching of memory elements [3,4] that corresponds to the minimal power dissipation of the circuit.

But despite the large amount of research in the area during the last 40 years, until recently the state assignment problem has not been studied from the complexity point of view although it is a natural complexity question.

The notion of BC-complexity corresponds to the standard interpretation of a “simplicity” of a switching function—the number of gates in the circuit. Roughly

the model is as following: encode inputs and states of a finite automaton as bit vectors and its transition function as a Boolean circuit. The complexity of this circuit we take as the complexity (BC-complexity) of (this circuit representation of) the automaton.

This paper is a logical continuation to the work that has been started in [5, 6]. BC-complexity was formally defined in [5], upper and lower bounds for the BC-complexity were obtained in [6] and also there it was shown that the Shannon effect takes place: for almost all languages with a given state complexity their BC-complexity is close to its maximum value. Also in [5] it was shown that the minimization of the number of states can lead to a dramatic increase in the BC-complexity—a proof to a well-known statement that state minimization should be considered together with state assignment.

The contribution of this paper is three-fold. At first, we prove a useful lemma (Lemma 3), that allows us to prove lower bounds of BC-complexity in a uniform manner for different language classes.

Then we use this lemma to prove the lower bound for BC-complexity of languages for a given non-deterministic state complexity and also improve the upper bound from [6] so, that these bounds now differ only by a constant multiplicative factor (at most 4, for a Binary input alphabet).

In the final part of this paper we consider the algorithmic complexity of various algorithms on DFAs in circuit representation. It is proved that many simple problems (e.g. determining if a state is reachable) under such representation become PSPACE-complete. In particular, we prove that the minimization of BC-complexity for DFAs given in their circuit representation and for NFAs in their standard state table representation are PSPACE-complete.

2 Finite Automata and Boolean Circuits

In this paper we use the standard models of deterministic and nondeterministic finite automata (DFA and NFA), we assume that the reader is familiar with notions of their equivalence, state reachability and minimization. By $sc(L)$ ($nsc(L)$) we denote the state (nondeterministic state) complexity of a regular language L , the minimal number of states of a DFA (NFA) that recognizes L .

For the sake of simplicity we will use estimates of the number of distinct languages with a bounded state or nondeterministic state complexity from [7] in a slightly relaxed form. Let \mathfrak{L}_s^k (\mathfrak{N}_s^k) be the set of languages over k -letter alphabet whose state (nondeterministic state) complexity is not larger than s .

Theorem 1 [7].

$$\begin{aligned} |\mathfrak{L}_s^k| &\geq 2^s s^{(k-1)s} \quad \text{for } s \geq 3. \\ 2^{(k-1)s^2} &\leq |\mathfrak{N}_s^k| \leq 2s2^{ks^2} \quad \text{for } k \geq 2 \quad \text{and} \\ 2^s &\leq |\mathfrak{N}_s^1| \leq 2^{s \log s} \quad \text{if } k = 1. \end{aligned}$$

We will use the standard notion of a Boolean circuit [8, 9] and restrict our attention to circuits in the standard base ($\&$, \vee , \neg). The size of the circuit $C(F)$ is the number of gates in it, the complexity of a Boolean function $C(f)$ is the size of the smallest circuit that represents this function.

We will use the following notation from [9] for the asymptotic comparison of two functions f and g :

$$f(n) \lesssim g(n) \iff \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq 1 \quad \left(\iff f(n) < g(n)(1 + o(1)) \right).$$

This is more precise than the standard big-Oh notation, which ignores constant factors.

Given a family of sets S_n we will say that property $P(x)$ is true for almost all $x \in S_n$ if the fraction of x in S_n for which $P(x)$ is not true tends to zero when n goes to infinity. We will say that for almost all $x \in S_n$ $f(x) \lesssim h(n)$ if there is a function $g(n)$ such that for almost all $x \in S_n$ $f(x) \leq g(n)$ and $g(n) \lesssim h(n)$.

The next theorem of this chapter is the crucial part that we will use in the estimation of lower bounds for BC-complexity. Although this result has not appeared in the literature before, it is very similar to Theorem 16 in [9] and the technique we use to prove it (counting argument) is quite standard, for this reason its proof is just sketched.

For each circuit x with n input bits and C gates let $mC(x) = n + C$ be its modified complexity—we need it to avoid the fact that the number of distinct circuits even with zero (standard) circuit complexity is infinite.

Theorem 2. *Let \mathfrak{A}_n be an arbitrary sequence of finite sets of increasing size, whose elements are non-isomorphic Boolean circuits with the number of output variables not exceeding the number of input variables plus one. Then for arbitrary constants a_1 and a_2 for almost all $x \in \mathfrak{A}_n$*

$$mC(x) > \frac{\log |\mathfrak{A}_n|}{\log \log |\mathfrak{A}_n| - a_1} + a_2$$

Proof. The number of Boolean circuits with n input variables, m output variables and no more than C gates, that correspond to distinct Boolean functions does not exceed $9^{C+n}(C+n)^{C+m}$ [6].

Let k be a given natural number and let $N(k)$ denote the number of distinct Boolean circuits with n inputs, no more than $n+1$ outputs whose modified complexity does not exceed k . Then $N(k) < (16k)^k$ for sufficiently large k , this can be obtained by summing the previous expression for all $n, m \leq k$.

Now let $r = \frac{\log |\mathfrak{A}_n|}{\log \log |\mathfrak{A}_n| - a_1} + a_2$ and let ε be the fraction of the circuits from \mathfrak{A}_n whose modified complexity is less or equal to r . It is enough to show that $\varepsilon \rightarrow 0$ when $|\mathfrak{A}_n| \rightarrow \infty$, and that follows from the fact that $\varepsilon \leq \frac{N(r)}{|\mathfrak{A}_n|} < \frac{(16r)^r}{|\mathfrak{A}_n|}$ with the aid of some algebra. \square

3 Circuit Representation of DFA and BC-Complexity

In this section we shortly repeat the main definitions from [6] regarding circuit representation of DFA and BC-complexity.

An encoding $E(X)$ of a set X onto a binary string is an injective mapping $f_X : X \rightarrow \{0, 1\}^{b_X}$ where b_X is the length of the encoding. An encoding of a DFA consists of an encoding of its input alphabet f_Σ and an encoding of the state space f_Q which we call input encoding and state encoding, respectively. Additionally, for the state encoding we ask that the start state is encoded as a string of all zeros $f_Q(q_0) = 0^{b_Q}$.

A circuit representation of a DFA consists of two circuits that compute its transition function and acceptance function (characteristic function of the subset of accepting states). The input of the transition function is encoded input and encoded state and its output is encoded (next) state. The input of the acceptance circuit is encoded state and it has one bit output whether this state is accepting state or not.

The BC-complexity of a circuit representation of a DFA (F, G) is the sum of complexities of its transition circuit and acceptance circuit and the number of state bits:

$$C_{BC}((F, G)) = C(F) + C(G) + b_Q.$$

The BC-complexity of a DFA A , $C_{BC}(A)$, is the minimal BC-complexity of its circuit representations. BC-complexity of a regular language L is the minimal BC-complexity of a DFA that recognizes L .

The number of state bits b_Q is included in the definition to avoid a situation that an automaton has a large number of states but zero BC-complexity. It is natural to assume that it costs something to create a circuit even if it has no gates and this is one of the possible ways how to reflect that in the definition.

4 Lower Bounds on BC-Complexity

In the beginning we will prove a lemma that will help us to estimate lower bounds of BC-complexity in the future.

Lemma 3. *Let a be an arbitrary constant and let \mathfrak{L}_n be a sequence of finite sets of increasing size whose elements are distinct regular languages over a fixed alphabet. Then for almost all $L \in \mathfrak{L}_n$*

$$C_{BC}(L) \geq \frac{\log |\mathfrak{L}_n|}{\log \log |\mathfrak{L}_n| - a}.$$

Proof. For each $L \in \mathfrak{L}_n$ we find the DFA that has a representation (F, G) with minimal BC-complexity. We can combine its transition and acceptance circuits into one circuit H (see Fig. 1) that has $b_Q + b_\Sigma$ input variables and $b_Q + 1$ output variables. Notice that $mC(H) = C(F) + C(G) + b_Q + b_\Sigma = C_{BC}(F, G) + b_\Sigma$. It is easy to see that as languages are distinct then for each two of them these

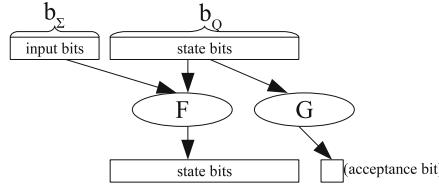


Fig. 1. Transition (F) and acceptance (G) circuits combined together

circuits H are non-isomorphic. Also note that the number of output variables can be by at most one larger than the number of input variables.

If we apply Theorem 2 for this sequence of sets of circuits H with $a_1 = a$ and $a_2 = b_{\Sigma}$, then we get that for almost all $L \in \mathcal{L}_n$:

$$C_{\text{BC}}(L) = mC(H) - b_{\Sigma} > \frac{\log |\mathcal{L}_n|}{\log \log |\mathcal{L}_n| - a} + b_{\Sigma} - b_{\Sigma} = \frac{\log |\mathcal{L}_n|}{\log \log |\mathcal{L}_n| - a}$$

□

We illustrate the usage of this lemma in the next theorem. The Shannon effect for BC-complexity here means that the BC-complexity of almost all languages in \mathcal{L}_s^k is close to its maximum value.

Theorem 4. [6] *For almost all $x \in \mathcal{L}_s^k$ ($k \geq 2$)*

$$(k-1)s \lesssim C_{\text{BC}}(L) \lesssim (k-1)s$$

For almost all $x \in \mathcal{L}_s^1$

$$\frac{s}{\log s} \lesssim C_{\text{BC}}(L) \lesssim \frac{s}{\log s}.$$

Proof. We will prove just the second statement (lower bound) for $k \geq 2$. From Theorem 1 we know that $|\mathcal{L}_s^k| \geq 2^s s^{(k-1)s} \geq s^{(k-1)s}$ and from Lemma 3 we see that for arbitrary a for almost all $L \in \mathcal{L}_s^k$

$$C_{\text{BC}}(L) \geq \frac{\log s^{(k-1)s}}{\log \log s^{(k-1)s} - a} = \frac{(k-1)s \log s}{\log s + \log \log s + \log(k-1) - a}$$

If we choose $a = \log(k-1)$ and use $\frac{\log s}{\log s + \log \log s} \gtrsim 1$, then we get the result. For $k=1$ the lower bound follows in the same manner. □

5 BC-Complexity and Nondeterministic State Complexity

In this chapter we compare the nondeterministic state complexity with the BC-complexity of a language. It is well known that if $nsc(L) = s$ then $\log sc(L) \leq s \leq$

$sc(L)$ and putting that together with Theorem 4 we obtain the first estimation of BC-complexity for $L \in \mathfrak{N}_s^k$:

$$\log s \leq C_{\text{BC}}(L) \lesssim (k - 1)2^s. \quad (1)$$

At first, let's look at the lower bound. It is well known that there are regular languages L for which $nsc(L) = sc(L)$ e.g. the language L_s that consists of all words whose length is divisible by s . For this language $nsc(L_s) = sc(L_s) = n$, but its BC-complexity is not larger than $c[\log n]$: its transition circuit does an addition modulo s . Hence BC-complexity of L_s is just a constant times larger than the lower bound in our rough estimation (1).

But the upper bound is far too high. Even when we know that for almost all languages in \mathfrak{L}_s^k the BC-complexity is around its maximal value $(k - 1)s$ (Theorem 4, Shannon effect), it turns out that languages L with $nsc(L) \ll sc(L)$ have their BC-complexity much lower than this maximum. The next theorem improves the naive upper bound (1) exponentially (and the bound obtained in [6] by a $\log s$ factor).

Theorem 5. *For all $L \in \mathfrak{N}_s^k$*

$$C_{\text{BC}}(L) \lesssim \frac{ks^2}{\log s}.$$

Proof. Let N be an NFA that recognizes L with s states $Q = \{q_1, q_2, \dots, q_s\}$. Let A be DFA that is obtained from N with a powerset construction without any further minimization. Its state space is 2^Q (set of all subsets of Q), and it can be naturally encoded into s state bits: subset $S \subseteq Q$ we can encode with a bit vector $f_Q(S) = z_1, \dots, z_s$ such that $z_i = 1 \iff q_i \in S$. Input alphabet $\Sigma = \{a_1, \dots, a_k\}$ we will encode with k bits $f_\Sigma(a_m) = x_1, \dots, x_k$, from which all are zeros except the m -th: $x_i = 1 \iff i = m$.

Denote $Q_m^i \subseteq Q$ to be the subset of the states of NFA N from which by reading letter $a_m \in \Sigma$ the automaton leads to state q_i . The property that by reading $a_m \in \Sigma$ NFA N moves from a subset of its states $S \subseteq Q$ to $S' \subseteq Q$ can be expressed with formula $q_i \in S' \iff (S \cap Q_m^i) \neq \emptyset$. It means that in a transition circuit for each output (state) bit z'_i we have to compute the following Boolean formula:

$$z'_i = \bigvee_{m=1}^k \left(x_m \& \bigvee_{q_j \in Q_m^i} z_j \right).$$

If we would create a circuit for each of these s formulas separately, then after careful counting the complexity of the transition circuit $C(F)$ would be $t + (k - 1)s$ where t is the number of transitions of N . This is not bad if t is small, but it can be as high as $t = ks^2$.

However we can improve the complexity it by a logarithmic factor. The main part of our computation is ks disjunctions $\bigvee_{q_j \in Q_m^i} z_j$: one for each output state bit and each input letter, and the main idea of the optimization is to reuse some intermediate results among them.

At first we split all our state bits into groups of c bits (the constant c will be chosen later), there are $\lceil \frac{s}{c} \rceil$ such groups. For each group we precompute all $2^c - 1$ disjunctions among its inputs for which we need $2^c - c - 1$ OR gates (one for each disjunction except the given inputs and the empty disjunction) and for all groups this gives us in total $\lceil \frac{s}{c} \rceil (2^c - c - 1)$ gates.

Now each of our ks disjunctions $\bigvee_{q_j \in Q_m^i} z_j$ can be computed as a disjunction of $\lceil \frac{s}{c} \rceil$ precomputed elements using $ks(\lceil \frac{s}{c} \rceil - 1)$ OR gates in total. For each output bit z'_j we also need k AND gates and we need $(k - 1)$ OR gates to compute the outer disjunction $\bigvee_{m=1}^k$. This would add additional $(2k - 1)s$ gates for the transition circuit F .

The acceptance circuit G is simply a disjunction of all input bits z_i that correspond to an accepting state of N . Its size does not exceed s . The number of state bits $b_Q = s$ also is counted into BC-complexity and now we have all the parts for it:

$$C_{\text{BC}}(A) \leq C(F) + C(G) + b_Q \leq ks\left(\lceil \frac{s}{c} \rceil - 1\right) + \lceil \frac{s}{c} \rceil (2^c - c - 1) + (2k - 1)s + s + s$$

and by grouping similar terms and estimating $\lceil \frac{s}{c} \rceil (-c - 1) + s \leq 0$ we finally get $C_{\text{BC}}(A) \leq \lceil \frac{s}{c} \rceil (ks + 2^c) + ks$.

It remains to find the best value for the constant c and we choose $c = \lceil \log s - \log \log s \rceil$. Then $\lceil \frac{s}{c} \rceil \leq \frac{s}{\log s - \log \log s - 1}$, $2^c \leq 2^{\log s - \log \log s + 1} = \frac{2s}{\log s}$ and

$$C_{\text{BC}}(A) \leq \left(\frac{s}{\log s - \log \log s - 1} \right) \left(ks + \frac{2s}{\log s} \right) + ks \lesssim \frac{ks^2}{\log s}.$$

□

At the end of this section we will show that this upper bound is almost (within a constant factor) optimal.

Theorem 6. *If $k \geq 2$, then for almost all $L \in \mathfrak{N}_s^k$*

$$C_{\text{BC}}(L) > \frac{(k - 1)s^2}{2 \log s}.$$

Proof. From Theorem 1 we know that $|\mathfrak{N}_s| \geq 2^{(k-1)s^2}$ if $k \geq 2$. Then Lemma 3 with $a = \log(k - 1)$ tells us that for almost all $L \in \mathfrak{N}_s^k$

$$C_{\text{BC}}(L) > \frac{\log |\mathfrak{N}_s|}{\log \log |\mathfrak{N}_s| - a} > \frac{(k - 1)s^2}{\log(k - 1)s^2 - \log(k - 1)} = \frac{(k - 1)s^2}{2 \log s}.$$

□

For deterministic state complexity the upper and lower bounds on BC-complexity coincide (Theorem 4). But if we look at Theorems 5 and 6 we see that $\frac{(k-1)s^2}{2 \log s} < C_{\text{BC}}(L) \lesssim \frac{ks^2}{\log s}$, upper and lower bounds differ by a factor $2k/(k - 1) < 4$. Why is this so and how can we improve these bounds?

The difference in coefficients $k - 1$ and k comes from the estimation of $|\mathfrak{N}_s^k|$ in Theorem 1 where one can see the same factors k and $k - 1$ in the upper and lower bounds. If one could improve the lower bound of Theorem 1, then that would automatically improve the lower bound of Theorem 6.

But the factor 2 in the denominator comes from the transition circuit where ks disjunctions $\bigvee q_j$ are constructed. We could consider it as a separate problem and for simplicity lets assume that $k = 1$. Let $f : \{0, 1\}^s \rightarrow \{0, 1\}^s$ be a function, where if x_i and y_i are input and output Boolean variables and each output variable is a disjunction of some input variables: $y_i = x_{i_1} \vee x_{i_2} \vee \dots \vee x_{i_k}$.

For this class of functions the best known upper and lower bounds on circuit complexity differ twice. The upper bound of $\frac{s^2}{\log s}$ gates can be constructed as in Theorem 5, the lower bound of $\frac{s^2}{2\log s}$ can be obtained by a counting argument. The problem looks simple but turns out to be a hard one that can additionally be illustrated by a fact that despite intuition there are such functions f , for which the smallest circuit besides OR gates contain some AND gates as well [10].

In this chapter we have said very little about unary languages. For them the best lower bound that we can get is the same as for the deterministic case (Theorem 4), and the upper bound from the Theorem 5 gives us that $\frac{s}{\log s} < C_{BC}(L) \lesssim \frac{s^2}{\log s}$ for almost all $L \in \mathfrak{N}_s^1$.

The difference in these two formulas is by a factor of s and it cannot be explained just by a difference in the upper and lower bound of Theorem 1. Possibly one can create a circuit representation of unary NFAs more effectively than in Theorem 5, but this question requires further research.

6 Computational Complexity of Automata Problems in Circuit Representation

For the standard automata representation (state table) problems like state reachability, state equivalence or automata minimization are “easy” ones, they can easily be computed in polynomial time and some of them even in logspace. But the situation becomes completely different if a DFA is given in its circuit representation, in such a setting all these problems are PSPACE-complete.

To show this we will use the theory of succinct versions of algorithmic instances developed in 80s and 90s in a series of papers [11–13]. We will adopt the definitions and results from [13].

Let $c(x_1, \dots, x_n)$ be a circuit with n input variables and one output variable. It describes a word $w(c)$ of length 2^n over a binary alphabet in a natural way: its i -th letter is the value of its i -th assignment, where assignments are ordered lexicographically. In other words, $w(c)$ is the result column of the truth table representation of c .

Let the word described by a circuit c and an integer m , formally $w(c, m)$ be the length- m prefix of $w(c)$. Note that this implies $w(c) = w(c, 2^n)$. If a word x does not encode a circuit, then $w(x, m)$ is defined to be the empty word. We say that (c, m) is a succinct instance of $w(c, m)$. The succinct version $S(A)$ of a language A is the set of all succinct instances of all words in A .

When working with succinct versions of languages, instead of standard polynomial time reducibility we will use polylogarithmic time reducibility \leq^{PLT} . A Turing machine that works in polylogarithmic time cannot read even the whole input. To overcome this we will use a known modification—a random access Turing machine, that can in one step access any bit on its input tape. Another problem we have to overcome is the length of the output, a function that works in polylogarithmic time can produce at most polylogarithmic number of output bits. To overcome this we ask that each output bit is computable in polylogarithmic time.

Definition 7. *A language A is polylogarithmic time reducible to a language B ($A \leq^{PLT} B$) if there are two functions $R : \Sigma^* \times \mathbb{N} \rightarrow \{0, 1\}$ and $l : \Sigma^* \rightarrow \mathbb{N}$ computable in polylogarithmic time with random access to their input such that*

$$x \in A \leftrightarrow R(x, 0)R(x, 1)\dots R(x, l(x)) \in B$$

It is easy to see that \leq^{PLT} reducibility is reflexive and transitive and implies standard polynomial time many-one reducibility. The main result that we will need is following:

Theorem 8. [13] *Let $f(n)$ be a nondecreasing bound. Then, if A is \leq^{PLT} hard for $DTIME(f(n))$, $NTIME(f(n))$, $DSPACE(f(n))$ or $NSPACE(f(n))$ then $S(A)$ is \leq^{PLT} hard for $DTIME(f(2^n))$, $NTIME(f(2^n))$, $DSPACE(f(2^n))$ or $NSPACE(f(2^n))$, respectively.*

Notice that the circuit representation of an automaton and a succinct instance of an automaton are closely related. The circuit representation of an automaton consists of two circuits—one for the transition function and one for the acceptance table while the succinct version of an automaton consists of one circuit describing both transition and acceptance tables bit by bit. These representations can easily be obtained one from another.

Let (F, G) be a circuit representation of an automaton, then by adding one more argument that tells which bit to extract this can easily be changed to a succinct instance of an automaton. On the other hand given a succinct instance of an automaton where state is encoded as b binary bits, one can concatenate b such circuits to obtain a transition function of an automaton and one more succinct representation one can turn into an acceptance circuit. This transformation can easily be done in polynomial time. Without going into details we will state that, given a succinct instance of DFA A of size n , one can in polynomial time construct a circuit representation of A with BC-complexity $poly(n)$ and vice versa.

Let REACH be a language that consists of all pairs (s, A) where s is a reachable state in DFA A (given in standard form). By L and NL we denote (as usual) the class of languages computable in logarithmic space with a deterministic or nondeterministic Turing machines, respectively.

Theorem 9. *For a fixed input alphabet $|\Sigma| \geq 2$, REACH is NL-complete under \leq^{PLT} reduction. For a unary alphabet it is L-complete under \leq^{PLT} reduction.*

Proof. First we have to show that $\text{REACH}(s, A) \in \text{NL}$. We can guess the input for which s is reachable nondeterministically, to do this we just need to keep the counter, current state and the next state on the working tape, it is easy to see that it takes no more than logarithmic space.

Now let's prove that any language $B \in \text{NL}$ is \leq^{PLT} reducible to $\text{REACH}(s, A)$. Consider a nondeterministic TM M that recognizes B using no more than $c \log n$ space and consider its configuration graph V on input x . It is a directed graph, each vertex of which has two (non-deterministic), one (deterministic) or 0 (end-state) outgoing edges. The states of DFA A will be the vertices of V (configurations of M), transitions of A will be the edges of V . For each state (configuration) there are at most two outgoing transitions that we can label with two (or less) input letters, for the DFA to be completely specified we can set all other transitions to stay in the same state. The state s of our interest will correspond to a configuration of M in its end-state having "1" on its worktape (accepting configuration). It is easy to see that $x \in B$ iff s is reachable in A .

As M works in logspace its configuration graph is polynomial on the size of its input. The length of each configuration is logarithmic in n and all the transitions of A (edges of the configuration graph) can be computed in polylogarithmic time using the definition of M .

For unary alphabet one just have to notice that each vertex of a configuration graph of a deterministic TM has at most one outgoing edge. \square

Denote by REACH_{CR} the language consisting of all pairs $(s, (F, G))$ where (F, G) is a circuit representation of a DFA and s is a state that is reachable in A .

Theorem 10. *For a fixed input alphabet, REACH_{CR} is PSPACE-complete.*

Proof. First let's see why REACH_{CR} is in PSPACE. Similarly as for REACH we can nondeterministically guess the input which leads to state s from the start state, no more than polynomial space is needed for that.

A direct consequence of Theorems 9 and 8 is that $S(\text{REACH})$ is PSPACE-hard. But $S(\text{REACH})$ can be reduced to REACH_{CR} in polynomial time as noted before. \square

Using this result we can show that many more problems on DFA in their circuit representation are PSPACE-complete.

Theorem 11. *The following problems are PSPACE-complete:*

1. *Given a circuit representation of a DFA and (an encoding of) two its states determine if these states are equivalent*
2. *Given a circuit representation of a DFA determine if the language it accepts is the empty language*
3. *Given two circuit representations of DFAs determine if these DFAs are equivalent*
4. *Given a circuit representation of a DFA and a number c determine if there is an equivalent DFA with BC-complexity at most c*

Proof. For the first problem if the given two states are not equivalent then we can nondeterministically guess the input word which leads them to states one of which is accepting, but the other not. Thus testing non-equivalence (and also equivalence) is in $\text{NPSPACE} = \text{PSPACE}$. Now let's see how REACH_{CR} can be reduced to state equivalence testing. Assume that we are given (F, G) and an encoding of state s , and we want to test if s is reachable. Change (F, G) in the following way: add one more state s' (as a separate state bit) that leads to itself for any input letter and set the only accepting state of A to be s . It is easy to see that start state s_0 and s' are equivalent iff s is not reachable.

For the second (emptiness) problem we have to test if all the accepting states are not reachable, what can easily be done in PSPACE . To show that it is PSPACE -complete we will again reduce REACH_{CR} to it. Let (F, G) be a given circuit representation and s be a state. Change the acceptance circuit so that s is the only accepting state. Now this DFA accepts empty language iff s is not reachable.

For the third problem to be in PSPACE we have to show that testing equivalence of initial states of two given automata can be done in PSPACE , what can be obtained the same way as for the first problem. To show that it is PSPACE -complete we note that even testing if a circuit representation of a given automaton is equivalent to a circuit representation of the empty language is PSPACE -complete.

Finally, for the fourth problem we can nondeterministically guess a circuit representation of DFA with no more than k states that is equivalent to a given one and then test their equivalence (that is in PSPACE). That puts this problem into $\text{NPSPACE} = \text{PSPACE}$. Next we will reduce emptiness problem to it. The only circuit representations whose BC-complexity is 0 are those of all-accepting and all-rejecting automata. Given a circuit representation of a DFA to test its emptiness we first test if the start state is an accepting state. If yes, then the automaton is not empty. If no, then we check if it has an equivalent DFA with zero BC-complexity, and it has iff it is an empty automaton. \square

The last problem in Theorem 11 is the minimization problem of DFA in their circuit representation stated as a decision problem. It shows that despite the fact that automata minimization in standard representation is easily solvable in polynomial time, in circuit representation this problem is PSPACE complete. But this is not the most natural setting of the problem. In the synthesis of sequential machines an automaton usually is given in its standard state table representation and the task is to find an optimal circuit representation for it. Thus we can ask a different question: Given a DFA (as a state table) and a number c determine if there exists an equivalent DFA with BC-complexity at most c .

This problem is definitely in PSPACE , one can construct a circuit representation of a given automaton and then apply Theorem 11. But is it really so hard or may be there are more effective methods to solve it remains an interesting open question.

On the other hand if the given automaton is an NFA (instead of DFA) then this problem is PSPACE-complete. Indeed, it is well known that determining if a given NFA is equivalent to all-accepting NFA is PSPACE-complete [14]. Hence this question is PSPACE-complete even for $c = 0$.

7 Conclusions

In this paper we continued the research (started in [5,6]) on the complexity properties of circuit representations of automata.

In the first part of the paper upper and lower bounds on BC-complexity were obtained for languages with a given nondeterministic state complexity, which differ just by a constant factor. We can conclude that if we want to express an automaton with s states as a Boolean circuit then it does not matter if it is a DFA or NFA, the size difference of the resulting circuit representation is polynomial ($(k - 1)s$ versus $\frac{ks^2}{\log s}$).

In the second part of the paper the algorithmic complexity of circuit representations of automata was studied. It was shown that if a circuit representation of DFA is given then many simple questions (is given state reachable, is this automaton minimal) are PSPACE-complete. On the other hand the exact complexity of the natural problem: given DFA find its minimal circuit representation, has left as an open question.

References

1. Kohavi, Z., Jha, N.K.: Switching and Finite Automata Theory. Cambridge University Press, Cambridge (2009)
2. Calazans, N.L.: State minimization and state assignment of finite state machines: their relationship and their impact on the implementation. PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium (1993)
3. Benini, L., De Micheli, G.: State assignment for low power dissipation. IEEE J. Solid-State Circuits **30**(3), 258–268 (1995)
4. Kajstura, K., Kania, D.: Low power synthesis of finite state machines—state assignment decomposition algorithm. J. Circuits Syst. Comput. **27**, 1850041 (2017)
5. Valdats, M.: Transition function complexity of finite automata. In: Holzer, M., Kutrib, M., Pighizzini, G. (eds.) DCFS 2011. LNCS, vol. 6808, pp. 301–313. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22600-7_24
6. Valdats, M.: Boolean circuit complexity of regular languages. In: International Conference on Automata and Formal Languages. EPTCS, vol. 151, pp. 342–354 (2014)
7. Domaratzki, M., Kisman, D., Shallit, J.: On the number of distinct languages accepted by finite automata with n states. J. Automata Lang. Comb. **7**(4), 469–486 (2002)
8. Wegener, I.: The Complexity of Boolean Functions. Wiley, New York (1987)
9. Lupanov, O.: Asymptotic estimates of the complexity of control systems. Moscow State University (1984). (in Russian)
10. Tarjan, R.E.: Complexity of monotone networks for computing conjunctions. Ann. Discret. Math. **2**, 121–133 (1978)

11. Lozano, A., Balcázar, J.L.: The complexity of graph problems for succinctly represented graphs. In: Nagl, M. (ed.) WG 1989. LNCS, vol. 411, pp. 277–286. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52292-1_20
12. Balcázar, J.L., Lozano, A., Torán, J.: The complexity of algorithmic problems on succinct instances. In: Baeza-Yates, R., Manber, U. (eds.) Computer Science, pp. 351–377. Springer, Boston (1992). https://doi.org/10.1007/978-1-4615-3422-8_30
13. Borchert, B., Lozano, A.: Succinct circuit representations and leaf languages are basically the same concept. Technical report, Universitat Politecnica de Catalunya (1996). <http://upcommons.upc.edu/handle/2117/97245>
14. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, New York (1979)



Disturbance Decoupling in Finite Automata

Alexey Zhirabok^{1,3}(✉) and Alexey Shumsky^{1,2}

¹ Far Easter Federal University, Vladivostok 690950, Russia
zhirabok@mail.ru, a.e.shumsky@yandex.com

² Institute of Applied Mathematics,
Far Eastern Branch of Russian Academy of Sciences, Vladivostok 690041, Russia
³ Institute of Marine Technology Problem,
Far Eastern Branch of Russian Academy of Sciences, Vladivostok 690950, Russia

Abstract. The paper addresses the disturbance decoupling problem by dynamic measurement feedback for finite automata. The mathematical technique called the pair algebra of partitions is used. The paper gives sufficient solvability conditions and a procedure to construct the required feedback.

Keywords: Finite automata · Disturbance decoupling
Measurement feedback · Pair algebra of partitions

1 Introduction

Disturbances in finite automata (FA) may appear as malfunctions (or faults) in technical elements of the system, caused either by external or internal perturbations. The solution of the disturbance decoupling problem (DDP) aims to construct a feedback controller in such a manner that the output-to-be-controlled in the closed-loop system is not influenced by the disturbance anymore. The solution of the DDP is important for FA since malfunctions happen with high probability in different technical systems constructed from a large number of elements. The DDP by output feedback control has been addressed for timed event graphs, that is a special case of discrete-event systems in Lhommeau et al. [1] and in Shumsky and Zhirabok [2] for the same class of systems as in this paper (see also Kotta and Mullari [3]). Finally note that the problem has been investigated in Cheng [4] and Yang et al. [5] for Boolean control networks using state feedback. In Katz [6] the method to compute the controlled-invariant sets for discrete-event systems over the max-plus algebra has been given.

The paper Shumsky and Zhirabok [2] sketched the unified approach for solution of the DDP in three separate cases - for continuous systems, discrete-time systems and discrete-event systems. The focus of [2] was on the unification aspect

The present paper investigates solvability of the disturbance decoupling problem via the dynamic measurement feedback for finite automata and provides the sufficient solvability conditions. The paper develops the results obtained in [7, 8].

The rest of the paper is organized as follows. In Sect. 2, the problem statement is formulated. Section 3 recalls some facts from the algebra of partitions, which are necessary for proving the results of this paper. Section 4 contains some preliminary results. Section 5 presents the problem solution. Section 6 concludes the paper.

2 Problem Formulation

Consider a finite automaton

$$\begin{aligned} x^+ &= \delta(x, u, w), \\ y &= \lambda(x), \\ y' &= \lambda'(x), \end{aligned} \tag{1}$$

where $X = \{x_1, \dots, x_n\}$, $x^+ \in X$ is the new state after transition from the state $x \in X$, initiated by the inputs $U = \{u_1, \dots, u_m\}$, $u \in U$, $W = \{w_1, \dots, w_p\}$, $w \in W$ is the unmeasurable disturbance, $Y = \{y_1, \dots, y_l\}$, $y \in Y$ is the measured output, and $y' \in Y' = \{y'_1, \dots, y'_{l'}\}$, $y' \in Y'$ is the output-to-be-controlled. The functions δ , λ , and λ' are determined by the table of state transitions and output functions (see Example 8).

In this paper we study the dynamic disturbance decoupling problem via measurement feedback. We are looking for dynamic feedback of the form

$$\begin{aligned} x_*^+ &= \delta_*(x_*, y, u), \\ u &= \lambda_*(x_*, y, u_*), \end{aligned} \tag{2}$$

where $X_* = \{x_{*1}, \dots, x_{*n_*}\}$, $x_* \in X_*$ and the new inputs $U_* = \{u_{*1}, \dots, u_{*m_*}\}$, $u_* \in U_*$ such that the values of the outputs-to-be-controlled y' of the closed-loop automaton are independent of the disturbance w . Note that cardinality of U_* is a design object. The functions δ_* and λ_* will be searched in the form of appropriate tables given δ , λ and λ' in (1). Closed-loop dynamics (1), (2) are shown in Fig. 1. Denote the dynamics in (2) by automaton S_* .

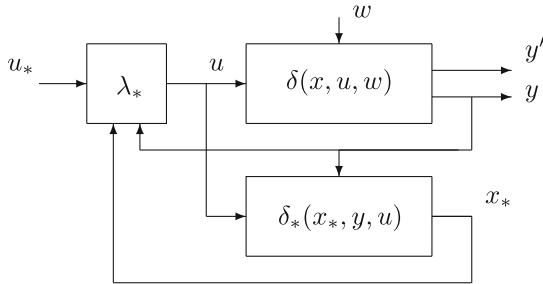


Fig. 1. Closed-loop dynamics

3 Mathematical Technique

The main ingredients of pair algebra of partitions are:

1. relation of partial order, denoted by \leq ,
 2. binary operations, denoted by \times and $+$,
 3. binary relation, denoted by Δ ,
 4. operators \mathbf{m} and \mathbf{M} .

Partitions. Let π and σ be partitions on X . One says that the partition π is less than or equal to σ , denoted by $\pi \leq \sigma$, if for every block $B_{\pi i} \in \pi$ there exists block $B_{\sigma j} \in \sigma$ such that $B_{\pi i} \subseteq B_{\sigma j}$, or $x \equiv x'(\pi) \Rightarrow x \equiv x'(\sigma)$. If we know the partition π on X , then we have knowledge about the states of the automaton with accuracy up to the partition π ; in particular, if $\pi \leq \sigma$, then the partition π contains the same amount or more information about states than σ .

There exist two special partitions denoted by **0** and **1**. Each block of the partition **0** contains only a single element of the set X ; the partition **1** has a single block containing all elements of X . Obviously, for arbitrary partition π on X , $\mathbf{0} \leq \pi \leq \mathbf{1}$.

Lattice. A set of all partitions on X with relation of partial order is a lattice [9]. Therefore, for each pair of partitions (π, σ) on X one can find two partitions $\inf(\pi, \sigma)$ and $\sup(\pi, \sigma)$. It is a common practice to replace these partitions by $\pi \times \sigma$ and $\pi + \sigma$, respectively and speak about the multiplication and addition instead of taking \inf and \sup . The partitions π , σ , and γ satisfy the following conditions:

$$(\pi \times \sigma \leq \pi, \pi \times \sigma \leq \sigma), (\gamma \leq \pi, \gamma \leq \sigma \Rightarrow \gamma \leq \pi \times \sigma); \\ (\pi \leq \pi + \sigma, \sigma \leq \pi + \sigma), (\pi \leq \gamma, \sigma \leq \gamma \Rightarrow \pi + \sigma \leq \gamma). \quad (3)$$

To calculate the product and sum, one may use the following simple rules. Each block of the partition $\pi \times \sigma$ is the intersection of some blocks of the partitions π and σ , and each block of the partition $\pi + \sigma$ is the union of all intersected blocks of the partitions π and σ .

Partition pairs. Let π and σ be partitions on X . The ordered pair (π, σ) is a partition pair on X , denoted as $(\pi, \sigma) \in \Delta$, iff the blocks of π are mapped by the state transition function of the automaton (1) into the blocks of σ :

$$x \equiv x'(\pi) \Rightarrow \delta(x, u, w) \equiv \delta(x', u, w)(\sigma) \quad \forall(u, w) \in U \times W.$$

Given a partition π , some partitions σ may exist such that $(\pi, \sigma) \in \Delta$; in particular, $(\pi, \mathbf{1}) \in \Delta$ is valid for arbitrary π . Alternatively, given a partition σ , some partitions π may exist such that $(\pi, \sigma) \in \Delta$; in particular, $(\mathbf{0}, \sigma) \in \Delta$ is valid for arbitrary σ .

Operators \mathbf{m} and \mathbf{M} . Define the operator \mathbf{m} as follows. Given partition π , $\mathbf{m}(\pi)$ is the smallest partition such that $(\pi, \mathbf{m}(\pi)) \in \Delta$. That is, for any other partition σ ,

$$(\pi, \sigma) \in \Delta \Rightarrow \mathbf{m}(\pi) \leq \sigma.$$

Because $(\pi, \mathbf{1}) \in \Delta$, the partition $\mathbf{m}(\pi)$ always exists, in extreme case, $\mathbf{m}(\pi) = \mathbf{1}$.

Define the operator \mathbf{M} as follows. Given a partition σ , $\mathbf{M}(\sigma)$ is the largest partition such that $(\mathbf{M}(\sigma), \sigma) \in \Delta$. That is, for any other partition π ,

$$(\pi, \sigma) \in \Delta \Rightarrow \pi \leq \mathbf{M}(\sigma).$$

Because $(\mathbf{0}, \sigma) \in \Delta$, the partition $\mathbf{M}(\sigma)$ always exists, in extreme case, $\mathbf{M}(\sigma) = \mathbf{0}$. Loosely speaking, the partition $\mathbf{m}(\pi)$ describes the largest amount of information regarding the state x^+ from the knowledge of π . Similarly, the partition $\mathbf{M}(\sigma)$ describes the least amount of information one has to know about x to compute σ for x^+ .

The following formulas can be used for calculation:

$$\begin{aligned} \mathbf{m}(\pi) &= \sum_{c \in U \times W} (\sigma_c \mid \sigma_c \text{ is minimal such that} \\ &\quad x \equiv x'(\pi) \Rightarrow \delta(x, c) \equiv \delta(x', c)(\sigma_c)), \\ \mathbf{M}(\sigma) &= \prod_{c \in U \times W} (\pi_c \mid \pi_c \text{ is maximal such that} \\ &\quad x \equiv x'(\pi_c) \Rightarrow \delta(x, c) \equiv \delta(x', c)(\sigma)). \end{aligned}$$

Lemma 1 [9]. *Let π , σ and γ be some partitions on X . Then*

1. $\pi \leq \sigma \Rightarrow \mathbf{m}(\pi) \leq \mathbf{m}(\sigma)$, $\mathbf{M}(\pi) \leq \mathbf{M}(\sigma)$;
2. $\pi \leq \sigma \Rightarrow \pi \times \gamma \leq \sigma \times \gamma$, $\pi + \gamma \leq \sigma + \gamma$;
3. $\pi \leq \sigma \Leftrightarrow \pi \times \sigma = \pi \Leftrightarrow \pi + \sigma = \sigma$.
4. $\mathbf{M}(\pi) \times \mathbf{M}(\sigma) = \mathbf{M}(\pi \times \sigma)$.
5. $\mathbf{m}(\mathbf{M}(\pi)) \leq \pi$, $\sigma \leq \mathbf{M}(\mathbf{m}(\sigma))$.

4 Preliminaries

To solve the problem, we will use both partitions and functions. It is known that if β is a function $\beta : X \rightarrow X_\beta$, then there exists partition π_β on X , defined by β such that

$$x \equiv x'(\pi_\beta) \Leftrightarrow \beta(x) = \beta(x'). \quad (4)$$

On the other hand, if π_β is a partition on X and $X_\beta = \{B_{\pi_\beta i}\}$ is the respective set of blocks of π_β , then there exists the function $\beta : X \rightarrow X_\beta$, corresponding to the partition π_β such that $\beta(x) = B_{\pi_\beta j}$ for $x \in B_{\pi_\beta j}$.

Lemma 2. *Let functions α and β correspond to the partitions π_α and π_β , respectively. Then $\pi_\alpha \leq \pi_\beta$ if and only if a function γ exists such that $\gamma(\alpha)(x) = \beta(x)$ for all $x \in X$.*

The proof of this lemma is obvious.

Consider a function $\varphi : X \rightarrow X_*$ such that

$$\varphi(\delta(x, u, w)) = \delta_0(\varphi(x), \lambda(x), u, w) \quad (5)$$

for some function δ_0 and for all $(x, u, w) \in X \times U \times W$. Note that the function φ always exists; in the extreme cases φ is the identity function with $\delta_0 = \delta$ or $\varphi = \text{const}$ with $\delta_0 = \text{const}$. The function φ can be found from the tables of transitions and outputs of automata (1) and (2). For that, introduce the partitions π_φ , π_λ , and $\pi_{\lambda'}$ on the set X , given by the functions φ , λ , and λ' respectively, according to the rule (4). That is, the states x and x' are in the same block of the partition π_φ (π_λ or $\pi_{\lambda'}$) if their images for the function φ (λ or λ') coincide. By analogy with Nijmeijer and van der Schaft [14] the function φ and the partition π_φ are called (λ, δ) -invariant. If we take in the above definition $\lambda(x) = \text{const}$ (that corresponds to $\pi_\lambda = \mathbf{1}$), then we end up with the concepts of δ -invariant function φ and partition π_φ .

It follows from (5) that if $\varphi(x) = \varphi(x')$ and $\lambda(x) = \lambda(x')$, then $\varphi(\delta(x, u, w)) = \varphi(\delta(x', u, w))$ for all $(u, w) \in U \times W$. Taking into account the relations between the functions φ , λ and their respective partitions π_φ , π_λ , one may alternatively write

$$(x \equiv x'(\pi_\varphi)) \& (x \equiv x'(\pi_\lambda)) \Rightarrow \delta(x, u, w) \equiv \delta(x', u, w)(\pi_\varphi). \quad (6)$$

This relation and definition of the binary relation Δ yields $(\pi_\varphi \times \pi_\lambda, \pi_\varphi) \in \Delta$. According to the definitions of operators \mathbf{m} and \mathbf{M} , the last inclusion is equivalent to the inequalities

$$\mathbf{m}(\pi_\varphi \times \pi_\lambda) \leq \pi_\varphi, \quad \pi_\varphi \times \pi_\lambda \leq \mathbf{M}(\pi_\varphi). \quad (7)$$

Note that these inequalities may be considered as an alternative definition of (λ, δ) -invariant partition π_φ , and will be used later in the proofs and computations.

Based on relation (5), one may say that the function $\varphi(\delta(x, u, w))$ can be expressed via $\varphi(x)$, $\lambda(x)$, u , and w . The same is true for the relations (6) and (7) in terms of partitions.

Lemma 3. Let π and σ be minimal (λ, δ) -invariant and δ -invariant partitions on X satisfying the conditions $\mu \leq \pi$ and $\mu \leq \sigma$ for arbitrary partition μ , respectively. Then $\pi \leq \sigma$.

Proof. By assumptions of the lemma and (7) the inequalities $\mathbf{m}(\pi \times \pi_\lambda) \leq \pi$ and $\mathbf{m}(\sigma) \leq \sigma$ hold (taking $\pi_\lambda = \mathbf{1}$ in (7)). By the 1st property of Lemma 1, $\mathbf{M}(\mathbf{m}(\pi \times \pi_\lambda)) \leq \mathbf{M}(\pi)$ and $\mathbf{M}(\mathbf{m}(\sigma)) \leq \mathbf{M}(\sigma)$; next, by the 5th property of Lemma 1, $\pi \times \pi_\lambda \leq \mathbf{M}(\pi)$ and $\sigma \leq \mathbf{M}(\sigma)$. Multiplication of these inequalities componentwise yields $\pi \times \sigma \times \pi_\lambda \leq \mathbf{M}(\pi) \times \mathbf{M}(\sigma)$. By the 4th property of Lemma 1, $\mathbf{M}(\pi) \times \mathbf{M}(\sigma) = \mathbf{M}(\pi \times \sigma)$; therefore $\pi \times \sigma \times \pi_\lambda \leq \mathbf{M}(\pi \times \sigma)$. The latter means that $\pi \times \sigma$ is (λ, δ) -invariant partition. Since $\mu \leq \pi$ and $\mu \leq \sigma$, then $\mu \leq \pi \times \sigma$ by (3). Because π is a minimal (λ, δ) -invariant partitions satisfying the condition $\mu \leq \pi$, then $\pi \leq \pi \times \sigma$. Since the inverse inequality $\pi \times \sigma \leq \pi$ is evident, one has $\pi \times \sigma = \pi$, then by the 3rd property of Lemma 1, $\pi \leq \sigma$. \square

The function α (and the appropriate partition π_α) is said to be a controlled-invariant if there exists a static state feedback $u = \lambda_0(x, u_*)$ such that the function α (the partition π_α) is δ -invariant for the closed-loop automaton.

Lemma 4. If the partition π is (λ, δ) -invariant (δ -invariant), then it is (λ, δ) -invariant (δ -invariant) in the closed-loop automaton.

Proof. Let the partition π be (λ, δ) -invariant, then $(x \equiv x'(\pi)) \& (x \equiv x'(\pi_\lambda)) \Rightarrow \delta(x, u, w) \equiv \delta(x', u, w)(\pi)$ for all u and w according to (6). If ψ is a function corresponding to π according to (4), then one may say that the composition $\psi(\delta(x, u, w))$ can be expressed via $\psi(x)$, $\lambda(x)$, u , and w . In the closed-loop automaton the input u is replaced with $u = \lambda_*(x_*, y, u_*)$ which can be expressed via $\psi(x)$, $\lambda(x)$, and u_* as well. Therefore, the partition π is (λ, δ) -invariant in the closed-loop automaton. In case $\pi_\lambda = \mathbf{1}$ one obtains δ -invariance. \square

5 Problem Solution

5.1 Automaton S_* Design

To construct the automaton S_* , we have to find the function φ that satisfies the additional property

$$\varphi(\delta(x, u, w)) = \delta_*(\varphi(x), \lambda(x), u) \quad (8)$$

for some function δ_* independent of w and for all $(x, u, w) \in X \times U \times W$. To construct (λ, δ) -invariant function φ with the property (8), introduce the smallest partition π_W on X that satisfies the condition

$$\delta(x, u, w) \equiv \delta(x, u, w')(\pi_W) \quad (9)$$

for all $(x, u) \in X \times U$ and for all $w, w' \in W$. Observe that the block of the partition π_W , containing the state $\delta(x, u, w)$ contains also all the states $\delta(x, u, w')$.

It follows from (8) and the definition of partitions, that π_φ satisfies (9) too, and since π_W is the smallest partition satisfying (8), then

$$\pi_W \leq \pi_\varphi. \quad (10)$$

Thus, the partition π_φ satisfies inequalities (7) and (10). To construct the automaton S_* , one has to find the smallest partition π_φ , satisfying the above properties, since this will yield the automaton S_* with the largest number of states.

Theorem 5. *Let*

$$\pi_0 = \pi_W, \quad \pi_{j+1} = \pi_j + \mathbf{m}(\pi_j \times \pi_\lambda), \quad j = 0, 1, \dots \quad (11)$$

Then there is k such that $\pi_k = \pi_{k+1}$. This π_k , from now onwards denoted as π_φ , is the smallest partition satisfying the conditions (7) and (10) simultaneously.

*Proof.*¹ Notice that by construction $\pi_{j+1} \geq \pi_j$ and thus $\pi_\varphi = \pi_k \geq \pi_W$. Since the set X is finite, the integer k exists such that $\pi_{k+1} = \pi_k$. It follows from (11) that $\pi_\varphi = \pi_k = \pi_{k+1} \geq \pi_k + \mathbf{m}(\pi_k \times \pi_\lambda)$. Therefore $\pi_\varphi = \pi_\varphi + \mathbf{m}(\pi_\varphi \times \pi_\lambda)$. Thus $\mathbf{m}(\pi_\varphi \times \pi_\lambda) \leq \pi_\varphi$. That is, the partition π_φ satisfies both conditions (7) and (10). Suppose that another partition π_α satisfies these conditions, i.e. $\mathbf{m}(\pi_\alpha \times \pi_\lambda) \leq \pi_\alpha$ and $\pi_W \leq \pi_\alpha$. Then $\mathbf{m}(\pi_0) \leq \mathbf{m}(\pi_\alpha)$ and $\mathbf{m}(\pi_0 \times \pi_\lambda) \leq \mathbf{m}(\pi_\alpha \times \pi_\lambda)$ by the 1st and 2nd properties of Lemma 1. From $\pi_0 \leq \pi_\alpha$, one obtains $\pi_0 + \mathbf{m}(\pi_0 \times \pi_\lambda) \leq \pi_\alpha + \mathbf{m}(\pi_\alpha \times \pi_\lambda)$. Since $\mathbf{m}(\pi_\alpha \times \pi_\lambda) \leq \pi_\alpha$ by (7), the above inequality can be rewritten in the form $\pi_1 = \pi_0 + \mathbf{m}(\pi_0 \times \pi_\lambda) \leq \pi_\alpha$. By induction, it can be shown that $\pi_2 \leq \pi_\alpha, \dots, \pi_\varphi = \pi_k \leq \pi_\alpha$. \square

Theorem 6. *The output-to-be-controlled y' of automaton (1) is disturbance decoupled iff there exists δ -invariant function α such that $\pi_W \leq \pi_\alpha \leq \pi_{\lambda'}$.*

Proof. Necessity. Using Theorem 5 ($\pi_\lambda = \text{const}$), find the minimal δ -invariant partition π_α satisfying the condition $\pi_W \leq \pi_\alpha$. By definition, δ -invariance yields $(\pi_\alpha, \pi_\alpha) \in \Delta$. Let α be a function corresponding to the partition π_α by (4). Define $z_0 = \alpha(x)$; because of the inequality $\pi_W \leq \pi_\alpha$, z_0 is independent of the disturbance w . Construct the automaton

$$z_0^+ = \alpha(\delta(x, u, w)) =: \delta_z(z_0, u).$$

This automaton has maximal possible number of states since π_α is a minimal δ -invariant partition. Since y' is disturbance decoupled, then maximality of the automaton above means that the output y' may be considered as an output of this automaton, therefore, $y' = \lambda_0(z_0)$ for some function λ_0 . By definition, $y' = \lambda'(x)$, hence $\lambda'(x) = (\lambda_0(\alpha))(x)$ for all $x \in X$, meaning that $\pi_\alpha \leq \pi'_\lambda$ by Lemma 2.

¹ For the specific case $\pi_\lambda = \mathbf{1}$, Theorem 5 has been proved in [9], in this case the appropriate partition and the corresponding function are δ -invariant.

Sufficiency. Since π_α is δ -invariant, then $(\pi_\alpha, \pi_\alpha) \in \Delta$. Define $z^0 := \alpha(x)$; because of the inequality $\pi_W \leq \pi_\alpha$, z^0 is independent of w . Construct the automaton

$$z^{0+} = \alpha(\delta(x, u, w)) =: \delta^z(z^0, u).$$

Since $\pi_\alpha \leq \pi'_\lambda$, by Lemma 2, the function λ^0 exists such that $\lambda^0(\alpha(x)) = \lambda'(x)$ for $x \in X$. From above, $y' = \lambda'(x) = \lambda^0(z^0)$; because z^0 is independent of the disturbance w , then y' is independent of w as well. \square

Note that the state x_* of the compensator (2) is defined by $x_* = \varphi(x)$. Since the dynamics of the compensator (2) is free of the disturbance w and π_W is the minimal partition which does not depend on w , the following condition must be satisfied: $\pi_W \leq \pi_\varphi$. Recall that some partition π_α is controlled-invariant if the partition π_α is δ -invariant for the closed-loop automaton.

Theorem 7. *Automaton (1) can be disturbance decoupled by feedback (2) iff there exist a controlled-invariant partition π_ξ and a (λ, δ) -invariant partition π_φ (satisfying $\pi_W \leq \pi_\varphi$) such that*

$$\pi_W \leq \pi_\varphi \leq \pi_\xi \leq \pi_{\lambda'}. \quad (12)$$

Proof. Necessity. Assume that there exists a feedback (2) that solves the disturbance decoupling problem. Then, by Theorem 6, there exists δ -invariant (in the closed-loop automaton) partition π_ξ such that $\pi_W \leq \pi_\xi \leq \pi_{\lambda'}$. Since the partition π_ξ is δ -invariant in the closed-loop automaton, then it is controlled-invariant. In (2), the function φ is clearly (λ, δ) -invariant and the condition $\pi_W \leq \pi_\varphi$ is satisfied. Because π_φ is (λ, δ) -invariant, then by Lemma 4, π_φ is (λ, δ) -invariant in the closed-loop automaton. Since π_ξ is δ -invariant in the closed-loop automaton, then $\pi_\varphi \leq \pi_\xi$ by Lemma 3.

Sufficiency. Since the partition π_ξ is controlled-invariant, there exists a static state feedback $u = \lambda_*(x, u_*)$ such that $x \equiv x'(\pi_\xi) \Rightarrow \delta^*(x, u_*) \equiv \delta^*(x', u_*)(\pi_\xi)$ for all u_* and some δ^* . Since $\pi_W \leq \pi_\xi \leq \pi_{\lambda'}$, then by Theorem 6, the closed-loop automaton is disturbance decoupled. It remains to show that the function λ_* depends only on the variables x_* , y and u_* . Since $\pi_\varphi \leq \pi_\xi$, then $\mathbf{M}(\pi_\varphi) \leq \mathbf{M}(\pi_\xi)$. Due to (λ, δ) -invariance of the partition π_φ and transitivity of the relation \leq , one has $\pi_\varphi \times \pi_\lambda \leq \mathbf{M}(\pi_\varphi)$ and $\pi_\varphi \times \pi_\lambda \leq \mathbf{M}(\pi_\xi)$. The last inequality is equivalent to $(x \equiv x'(\pi_\varphi)) \& (x \equiv x'(\pi_\lambda)) \Rightarrow \delta^*(x, u_*) \equiv \delta^*(x', u_*)(\pi_\xi)$ for all u_* . This means that δ^* can be written in terms of x_* , y , and u_* and then the function λ_* depends also only on x_* , y , and u_* . \square

Note that relations similar to (12) were obtained by Isidori [15] in terms of distributions and in Kaldm   et al. [7] in terms of vector functions.

Define the dynamics of automaton S_* by the equation

$$x_*^+ = \delta_*(x_*, y, u) \quad (13)$$

where the function δ_* , given by (8), can be obtained from the table of transitions of automaton (1) by combining the states, contained in the same blocks of the partition π_φ , and denoting them as the state x_* of the automaton S_* .

Example 8. Consider the automaton, described by Table 1. The automaton has six states, three control inputs u and two disturbance inputs w , three measurement outputs y and two outputs to-be-controlled y' .

From Table 2, $\pi_\lambda = \{(1, 4), (2, 6), (3, 5)\}$, $\pi_{\lambda'} = \{(1, 4, 5), (2, 3, 6)\}$, and $\pi_W = \{(1), (2, 3), (4), (5), (6)\}$. Since $\pi_\lambda \times \pi_W = \mathbf{0}$, from (11) one obtains $\pi_1 = \pi_0$ and, as a result (see Theorem 5), $\pi_\varphi = \{(1), (2, 3), (4), (5), (6)\}$. Description of the automaton S_* is given by Table 2 where x_{*1} corresponds to the block (1), x_{*2} to (2, 3), x_{*3} to (4), x_{*4} to (5), x_{*5} to (6).

Table 1. Given automaton

x	x^+						y	y'		
	u_1		u_2		u_3					
	w_1	w_2	w_1	w_2	w_1	w_2				
x_1	x_1	x_1	x_2	x_3	x_5	x_5	a	A		
x_2	x_2	x_2	x_4	x_4	x_3	x_3	b	B		
x_3	x_6	x_6	x_5	x_5	x_3	x_3	c	B		
x_4	x_4	x_4	x_1	x_1	x_4	x_4	a	A		
x_5	x_5	x_5	x_1	x_1	x_5	x_5	c	A		
x_6	x_2	x_2	x_5	x_5	x_3	x_3	b	B		

Table 2. Automaton S_*

x_*	x_*^+		
	u_1	u_2	u_3
x_{*1}	x_{*1}	x_{*2}	x_{*4}
$x_{*2}, y = b$	x_{*2}	x_{*3}	x_{*2}
$x_{*2}, y = c$	x_{*5}	x_{*4}	x_{*2}
x_{*3}	x_{*3}	x_{*1}	x_{*3}
x_{*4}	x_{*4}	x_{*1}	x_{*4}
x_{*5}	x_{*2}	x_{*4}	x_{*2}

5.2 Construction of Function λ_*

Construction of the function λ_* is based on the partition π_ξ . To obtain the controlled-invariant partition π_ξ such that $\pi_\varphi \leq \pi_\xi$, one may use Theorem 7 after replacing π_W by π_φ to construct δ -invariant partition π_ξ , which, by Lemma 4, is δ -invariant in the closed-loop automaton, i.e. is controlled-invariant.

The alternative approach for obtaining the partition π_ξ is based on the partition $\pi_{\lambda'}$ and the following result. Find the sequence of partitions:

$$\pi_\theta^0 := \pi_{\lambda'}, \quad \pi_\theta^j = \pi_{\lambda'} \times \mathbf{M}(\pi_{\lambda'}) \times \dots \times \mathbf{M}^j(\pi_{\lambda'}), \quad j = 0, 1, \dots \quad (14)$$

When $\pi_\theta^c = \pi_\theta^{c+1}$ for some c , the partition $\pi_\xi = \pi_\theta^c$ is maximal δ -invariant satisfying the condition $\pi_\xi \leq \pi_{\lambda'}$ Hartmanis and Stearns [9].

Example 9. Consider Example 8. Set $\pi_0 := \pi_\varphi = \{(1), (2, 3), (4), (5), (6)\}$ and use Theorem 7. Compute $\mathbf{m}(\pi_0) = \{(1), (2, 6), (3), (4, 5)\}$, $\pi_1 = \pi_0 + \mathbf{m}(\pi_0) = \{(1), (2, 3, 6), (4, 5)\}$, $\pi_2 = \pi_1 + \mathbf{m}(\pi_1) = \{(1), (2, 3, 6), (4, 5)\}$. As a result, $\pi_\xi = \{(1), (2, 3, 6), (4, 5)\}$. Since $\pi_\xi \leq \pi_{\lambda'}$, automaton (1) can be disturbance decoupled.

The alternative approach based on (14) gives the same result as before: $\pi_\xi = \{(1), (2, 3, 6), (4, 5)\}$. Since $\pi_\varphi \leq \pi_\xi$, automaton (1) can be disturbance decoupled.

Next, define the auxiliary automaton S_0 with the state $x_0 = \xi(x)$ by the equation

$$x_0^+ = \delta_0(x_0, u)$$

where δ_0 is the transition function of the quotient automaton of automaton (1) with equivalence relation π_ξ . It can be obtained from the table of transitions of automaton (1) by replacing the states contained in some block of the partition π_ξ , by the appropriate state x_{0i} . The automaton S_0 is given in Table 3 where x_{01} corresponds to the block (1), x_{02} to (2, 3, 6), and x_{03} to (4, 5).

Table 3. Auxiliary automaton S_0

x_0	x_0^+		
	u_1	u_2	u_3
x_{01}	x_{01}	x_{02}	x_{03}
x_{02}	x_{02}	x_{03}	x_{02}
x_{03}	x_{03}	x_{01}	x_{03}

Recall that the dynamics of S_* depends on the disturbance w only through the measured output y . To avoid this, our goal is to use the control u to compensate the possible dependence. To achieve this goal, the auxiliary automaton S_0 is used.

Set

$$x_0^+ = \delta_0(x_0, u) = u_*, \quad (15)$$

$u_* \in U_*$, where U_* is chosen in such a way that its cardinality is equal to the number of states of the automaton S_0 . Solve the Eq. (15) for u_* to obtain the function $\lambda_*(x_*, y, u_*)$. This function is the inverse of δ_0 in (15) with respect to variables u and can be obtained by interchanging the values of u and u_* in table of the automaton S_ξ : $u = \lambda_*(x_*, y, u_*)$.

To solve (15), introduce the set of partitions $\{\rho_{x_0}\}$ on the set U as follows:

$$u \equiv u'(\rho_{x_0}) \Leftrightarrow \delta_0(x_0, u) = \delta_0(x_0, u').$$

The set $\{\rho_{x_0}\}$ may be constructed by analyzing the rows of the table of transitions of the automaton S_0 corresponding to x_0 .

The unique solution of (15) is possible iff $\rho_{x_0} = \mathbf{0}$ for all $x_0 \in X_0$ since then different inputs u correspond to different inputs u_* for each x_0 . Such a solution can be found as follows. The table of transitions of S_0 is rewritten in such a manner that rows correspond to the states x_0 and columns correspond to the inputs u . Then the states $x_0^+ = \delta_0(x_0, u)$ are replaced by the inputs u_* according to (15).

If $\rho_{x_0} \neq \mathbf{0}$ for some x_0 , the state x_0^+ is replaced by the respective block $B_{\rho_{x_0}}$ of the partition ρ_{x_0} (instead of u values). In this case the function λ_* is partially defined.

The necessary condition for $\rho_{x_0} = \mathbf{0}$ for all x_0 is the inequality $|\pi_\varphi| \geq |U|$ where by $|\pi_\varphi|$ is denoted the number of blocks of the partition π_φ , $|U|$ means the cardinality of the set U . If $|\pi_\varphi| < |U|$, then $\rho_{x_0} \neq \mathbf{0}$ for some x_0 .

Example 10. Consider the automaton S_0 described by Table 3. Compute $\rho_{x_{01}} = \mathbf{0}$, $\rho_{x_{02}} = \rho_{x_{03}} = \{(u_1, u_3), (u_2)\}$. Since $\pi_\varphi \leq \pi_\xi$, then to construct the function λ_* , states x_{0j} of the automaton S_0 can be replaced by states x_{*j} . As a result, the function λ_* is given in Table 4.

Table 4. Function λ_*

x_0	u		
	u_{*1}	u_{*2}	u_{*3}
x_{*1}	u_1	u_2	u_3
x_{*2}	—	(u_1, u_3)	u_2
x_{*3}	u_2	—	(u_1, u_3)
x_{*4}	u_2	—	(u_1, u_3)
x_{*5}	—	(u_1, u_3)	u_2

6 Conclusions

A sufficient solvability condition has been given for the DDP by dynamic measurement feedback for finite automata as well as the algorithm to compute the feedback whenever it exists. The solution is obtained for automata, whose dynamics is described by tables. In the solution the concept of (λ, δ) -invariant partition plays a key role. This concept extends the notion of (h, f) -invariant distribution Nijmeijer and van der Schaft [14] for automata.

If we compare the results of this paper to those of Kaldm   et al. [7] that addresses the DDP in case of discrete-time systems, one can say the following. Discrete-time systems are described via difference equations whereas automata are given in the form of tables and this determines the choice of the mathematical techniques, the algebra of functions or the pair algebra of partitions,

respectively. These objects – functions and partitions – are, however, related; see (4). Because of the latter, many constructions in this paper correspond to those in [7]. In particular, partition π_φ corresponds to function α , partition π_ξ to function ξ , etc. However, the function λ_* in (2) of this paper, unlike those in [7] are not uniquely defined. Some results of this paper – Theorems 6 and 7 – are analogues of the results in Isidory [15] and [7]; they supplement the paper Kaldm   et al. [8].

Acknowledgments. This work was supported by the Russian Scientific Foundation (project 16-19-00046).

References

1. Lhommeau, M., Hardouin, L., Cottenceau, B.: Disturbance decoupling of timed event graphs by output feedback controller. In: Proceedings of the 6th Workshop on DES, Zaragoza, Spain, vol. 6, pp. 203–208 (2002)
 2. Shumsky, A., Zhirabok, A.: Unified approach to the problem of full decoupling via output feedback. *Eur. J. Control* **16**, 313–325 (2010)
 3. Kotta, U., Mullari, T.: Discussion on: unified approach to the problem of full decoupling via output feedback. *Eur. J. Control* **16**, 326–328 (2010)
 4. Cheng, D.: Disturbance decoupling of Boolean control networks. *IEEE Trans. Autom. Control* **56**, 2–10 (2011)
 5. Yang, M., Li, R., Chu, T.: Controller design for disturbance decoupling of Boolean control networks. *Automatica* **49**, 273–277 (2013)
 6. Katz, R.: Max-plus (a,b)-invariant spaces and control of timed discrete-event systems. *IEEE TAC* **52**(2), 229–241 (2007)
 7. Kaldmäe, A., Kotta, U., Shumsky, A., Zhirabok, A.: Measurement feedback disturbance decoupling in discrete-time nonlinear systems. *Automatica* **49**, 2887–2891 (2013)
 8. Kaldmäe, A., Kotta, U., Shumsky, A., Zhirabok, A.: Measurement feedback disturbance decoupling in discrete-event systems. *Int. J. Robust Nonlinear Control* **25**, 3330–3348 (2015)
 9. Hartmanis, J., Stearns, R.: The Algebraic Structure Theory of Sequential Machines. Prentice-Hall, Upper Saddle River (1966)
 10. Berdjag, D., Cocquempot, V., Christophe, C., Shumsky, A., Zhirabok, A.: Algebraic approach for model decomposition: application to fault detection and isolation in discrete-event systems. *Int. J. Appl. Math. Comput. Sci.* **21**, 109–125 (2011)
 11. Danilov, V., Kolesov, N., Podkopaev, B.: Algebraic model of hardware monitoring of automata. *Autom. Remote Control* **36**, 118–125 (1975)
 12. Kaldmäe, A., Kotta, U., Shumsky, A., Zhirabok, A.: Faulty plant reconfiguration based on disturbance decoupling methods. *Asian J. Control* **8**, 858–867 (2016)
 13. Zhirabok, A., Shumsky, A.: Method of fault accommodation in finite automata. *Autom. Remote Control* **71**, 122–132 (2010)
 14. Nijmeijer, H., van der Schaft, A.: Nonlinear Dynamical Control Systems. Springer, Heidelberg (1990). <https://doi.org/10.1007/978-1-4757-2101-0>
 15. Isidori, A.: Nonlinear Control Systems. Springer, Heidelberg (1995). <https://doi.org/10.1007/978-1-84628-615-5>



Default Logic and Bounded Treewidth

Johannes K. Fichte^{1(✉)}, Markus Hecher^{1(✉)}, and Irina Schindler^{2(✉)}

¹ Technische Universität Wien, Vienna, Austria

{fichte,hecher}@dbai.tuwien.ac.at

² Leibniz Universität Hannover, Hannover, Germany

schindler@thi.uni-hannover.de

Abstract. In this paper, we study Reiter’s propositional default logic when the treewidth of a certain graph representation (semi-primal graph) of the input theory is bounded. We establish a dynamic programming algorithm on tree decompositions that decides whether a theory has a consistent stable extension (EXT). Our algorithm can even be used to enumerate all generating defaults (ENUMSE) that lead to stable extensions. We show that our algorithm decides EXT in linear time in the input theory and triple exponential time in the treewidth (so-called *fixed-parameter linear* algorithm). Further, our algorithm solves ENUMSE with a pre-computation step that is linear in the input theory and triple exponential in the treewidth followed by a linear delay to output solutions.

Keywords: Parameterized algorithms · Tree decompositions
Dynamic programming · Reiter’s default logic · Propositional logic

1 Introduction

Reiter’s *default logic* (*DL*) is one of the most fundamental formalisms to non-monotonic reasoning where reasoners draw tentative conclusions that can be retracted based on further evidence [1, 2]. *DL* augments classical logic by rules of default assumptions (*default rules*). Intuitively, a default rule expresses “in the absence of contrary information, assume ...”. Formally, such rule is a triple $p : j \rightarrow c$ of formulas p , j , and c expressing “if prerequisite p can be deduced and justification j is never violated then assume conclusion c ”. For an initial set of facts, beliefs supported by default rules are called an extension of this set of facts. If the default rules can be applied consistently until a fixed point, the extension is a maximally consistent view (*consistent stable extension*) with respect to the facts together with the default rules. In *DL* stable extensions involve the construction of the deductive closure, which can be generated from

The work has been supported by the Austrian Science Fund (FWF), Grants Y698 and P26696, and the German Science Fund (DFG), Grant ME 4279/1-1. The first two authors are also affiliated with the Institute of Computer Science and Computational Science at University of Potsdam, Germany. An extended self-archived version of the paper can be found [online](#).

the conclusions of the defaults and the initial facts by means of so-called generating defaults. However, not every generating default leads to a stable extension. If a generating default leads to a stable extension, we call it a stable default set. Our problems of interest are deciding whether a default theory has a consistent stable extension (EXT), output consistent stable default sets (COMPSE), counting the number of stable default sets (#SE), and enumerating all stable default sets (ENUMSE). All these problems are of high worst case complexity, for example, the problem EXT is Σ_2^P -complete [3].

Parameterized algorithms [4] have attracted considerable interest in recent years and allow to tackle hard problems by directly exploiting certain structural properties present in input instances (the *parameter*). For example, EXT can be solved in polynomial time for input theories that allow for small backdoors into tractable fragments of DL [5]. Another parameter is treewidth, which intuitively measures the closeness of a graph to a tree. EXT can also be solved in linear time for input theories and a (non-elementary) function that depends on the treewidth of a certain graph representation of the default theory (incidence graph) [6]. This result relies on logical characterization in terms of a so-called MSO-formula and Courcelle's theorem [7]. Unfortunately, the non-elementary function can become extremely huge and entirely impractical [8]. More precisely, the result by Meier et al. [6] yields a function that is at least quintuple exponential in the treewidth and the size of the MSO-formula. This opens the question whether one can significantly improve these runtime bounds. A technique to obtain better worst-case runtime bounds that often even allows to practically solve problem instances, which have small treewidth, are dynamic programming (DP) algorithms on tree decompositions [9–11]. In this paper, we present such a DP algorithm for DL, which uses a slightly simpler graph notation of the theory (semi-primal graph).

Contributions. We introduce DP algorithms that exploit small treewidth to solve EXT and COMPSE in time triple exponential in the semi-primal treewidth and linear in the input theory. Further, we can solve #SE in time triple exponential in the semi-primal treewidth and quadratic in the input theory. Our algorithm can even be used to enumerate all stable default sets (ENUMSE) with a pre-computation step that is triple exponential in the semi-primal treewidth and linear in the input theory followed by a linear delay for outputting the solutions (Delay-FPT [12]).

2 Default Logic

A *literal* is a (propositional) variable or its negation. The *truth evaluation* of (propositional) formulas is defined in the standard way [2]. In particular, $\theta(\perp) = 0$ and $\theta(\top) = 1$ for an assignment θ . Let f and g be formulas and $X = \text{Vars}(f) \cup \text{Vars}(g)$. We write $f \vDash g$ if and only if for all assignments $\theta \in 2^X$ it holds that if the assignment θ satisfies f , then θ also satisfies g . Further, we define the *deductive closure* of f as $\text{Th}(f) := \{g \in \mathcal{P} \mid f \vDash g\}$ where \mathcal{P} is the family that contains all formulas. In this paper, whenever it is clear from the context, we may use sets of formulas and a conjunction over formulas equivalently. In particular,

we let for formula f and a family \mathcal{M} of sets of variables be $\text{Mod}_{\mathcal{M}}(f) := \{M \mid M \in \mathcal{M}, M \models f\}$. We denote with SAT the problem that asks whether a given formula f is satisfiable.

We define for formulas p , j , and c a *default rule* d as a triple $p : j \rightarrow c$; p is called the *prerequisite*, j is called the *justification*, and c is called the *conclusion*; we set $\alpha(d) := p$, $\beta(d) := j$, and $\gamma(d) := c$. The mappings α , β and γ naturally extend to sets of default rules. We follow the definitions by Reiter [1]. A *default theory* $\langle W, D \rangle$ consists of a set W of propositional formulas (knowledge base) and a set of default rules.

Definition 1. Let $\langle W, D \rangle$ be a default theory and E be a set of formulas. Then, $\Gamma(E)$ is the smallest set of formulas such that: (i) $W \subseteq \Gamma(E)$ (ii) $\Gamma(E) = \text{Th}(\Gamma(E))$, and (iii) for each $p : j \rightarrow c \in D$ with $p \in \Gamma(E)$ and $\neg j \notin E$, it holds that $c \in \Gamma(E)$. E is a stable extension of $\langle W, D \rangle$, if $E = \Gamma(E)$. An extension is inconsistent if it contains \perp , otherwise it is called consistent. The set $G = \{d \mid \alpha(d) \in E, \neg\beta(d) \notin E, d \in D\}$ is called the set of generating defaults of extension E and default theory D .

The definition of stable extensions allows inconsistent stable extensions. However, inconsistent extensions only occur if the set W is already inconsistent where $\langle W, D \rangle$ is the theory of interest [2, Corollary 3.60]. In consequence, (i) if W is consistent, then every stable extension of $\langle W, D \rangle$ is consistent, and (ii) if W is inconsistent, then $\langle W, D \rangle$ has a stable extension. For Case (ii) the stable extension consists of all formulas. Therefore, we consider only consistent stable extensions. For default theories with consistent W , we can trivially transform every formula in W into a default rule. Hence, in this paper we generally assume that $W = \emptyset$ and write a default theory simply as set of default rules. Moreover, we refer by $\text{SE}(D)$ to the set of all consistent stable extensions of D .

Example 1. Let the default theories D_1 and D_2 be given as $D_1 := \{d_1 = \top : a \rightarrow a \vee b, d_2 = \top : \neg a \rightarrow \neg b\}$ and $D_2 := \{d_1 = c : a \rightarrow a \vee b, d_2 = c : \neg a \rightarrow \neg b, d_3 = \top : c \rightarrow c, d_4 = \top : \neg c \rightarrow \neg c\}$. D_1 has no stable extension, while D_2 has only one stable extension $E_1 = \{\neg c\}$. ■

In our paper, we use an alternative characterization of stable extension beyond fixed point semantics, which is inspired by Reiter's stage construction [1].

Definition 2. Let D be a default theory and $S \subseteq D$. Further, we let $E(S) := \{\gamma(d) \mid d \in S\}$. We call a default $d \in D$ p -satisfiable in S , if $E(S) \cup \neg\alpha(d)$ is satisfiable; and j -satisfiable in S , if $E(S) \cup \beta(d)$ is unsatisfiable; c -satisfiable in S , if $d \in S$. The set S is a satisfying default set, if each default $d \in D$ is p -satisfiable in S , or j -satisfiable in S , or c -satisfiable in S .

The set S is a stable default set, if (i) S is a satisfying default set and (ii) there is no S' where $S' \subsetneq S$ such that for each default d it holds that d is p -satisfiable in S' , or j -satisfiable in S , or c -satisfiable in S' . We refer by $\text{SD}(D)$ to the set of all stable default sets of D .

The following lemma establishes that we can simply use stable default sets to obtain stable extensions of a default theory.

Lemma 1 (\star^1). *Let D be a default theory. Then,*

$$\text{SE}(D) = \bigcup_{S \in SD(D)} \text{Th}(\{\gamma(d) \mid d \in S\}).$$

In particular, $S \in SD(D)$ is a generating default of extension $\text{Th}(\{\gamma(d) \mid d \in S\})$.

Given a default theory D we are interested in the following problems: EXT asks whether D has a consistent stable extension. COMPSE asks to output a stable default set D . #SE asks to output the number of stable default sets of D . ENUMSE asks to enumerate all stable default sets of D .

3 Dynamic Programming on TDs for Default Logic

In this section, we present the basic methodology and definitions to solve our problems more efficiently for default theories that have small treewidth. Our algorithms are inspired by earlier work for another non-monotonic framework [10]. However, due to much more evolved semantics of DL, we require extensions of the underlying concepts.

Before we provide details, we give an intuitive description. The property treewidth was originally introduced for graphs and is based on the concept of a tree decomposition (TD). Given a graph, a TD constructs a tree where each node consists of sets of vertices of the original graph (bags) such that additional conditions hold. Then, we define a dedicated graph representation of the default theory and our algorithms work by dynamic programming (DP) along the tree decomposition (post-order) where at each node of the tree, information is gathered in tables. The size of these tables is triple exponential in the size of the bag. Intuitively, the TD fixes an order in which we evaluate the default theory. Moreover, when we evaluate the default theory for one node, we can restrict the theory to a sub-theory and parts of prerequisites, justifications, and conclusions that depends only on the content of the currently considered bag.

Tree Decompositions. Let $G = (V, E)$ be a graph, $T = (N, F)$ be a tree (N, F) with root n , and $\chi : N \rightarrow 2^V$ be a mapping. We call the sets $\chi(\cdot)$ bags and N the set of nodes. The pair $\mathcal{T} = (T, \chi)$ is a *tree decomposition* (TD) of G if the following conditions hold: (i) for every vertex $v \in V$ there is a node $t \in N$ with $v \in \chi(t)$; (ii) for every edge $e \in E$ there is a node $t \in N$ with $e \subseteq \chi(t)$; and (iii) for any three nodes $t_1, t_2, t_3 \in N$, if t_2 lies on the unique path from t_1 to t_3 ,

¹ Statements or descriptions whose proofs or details are omitted due to space limitations are marked with “ \star ”. These statements are sketched in an extended version.

then $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$. The *width* of the TD is the size of the largest bag minus one. The *treewidth* $tw(G)$ is the minimum width over all possible TDs of G . For $k \in \mathbb{N}$ we can compute a TD of width k or output that no exists in time $2^{\mathcal{O}(k^3)} \cdot |V|$ [13].

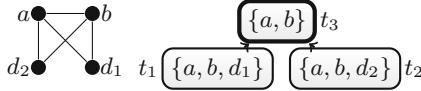


Fig. 1. Graph G (left) and an TD T (right) of G .

Next, we restrict the TD T such that we have only nice case distinctions for our DP algorithm later. Therefore, we define a *nice TD* in the usual way as follows. For a node $t \in N$ we say that $\text{type}(t)$ is *leaf* if t has no children; *join* if t has children t' and t'' with $t' \neq t''$ and $\chi(t) = \chi(t') = \chi(t'')$; *int* (“introduce”) if t has a single child t' , $\chi(t') \subseteq \chi(t)$ and $|\chi(t)| = |\chi(t')| + 1$; *rem* (“removal”) if t has a single child t' , $\chi(t) \subseteq \chi(t')$ and $|\chi(t')| = |\chi(t)| + 1$. If every node $t \in N$ has at most two children, $\text{type}(t) \in \{\text{leaf}, \text{join}, \text{int}, \text{rem}\}$, and bags of leaf nodes and the root are empty, then the TD is called *nice*. For every TD, we can compute a nice TD in linear time without increasing the width [13]. In our algorithms we will traverse a TD bottom up, therefore, let $\text{post-order}(T, t)$ be the sequence of nodes in post-order of the induced subtree $T' = (N', \cdot, t)$ of T rooted at t .

Example 2. Fig. 1 (left) depicts a graph G together with a TD of width 2 of G . Further, the TD T in Fig. 2 sketches main parts of a nice TD of G (obvious parts are left out). ■

Graph Representations of Default Theories. For a default theory D , its *incidence graph* $I(G)$ is the bipartite graph, where the vertices are of variables of D and defaults $d \in D$, and there is an edge da between a default $d \in D$ and a corresponding variable $a \in \text{Vars}(d)$. The *semi-primal graph* $S(D)$ of D is the graph, where the vertices are variables $\text{Vars}(D)$ and defaults of D . For each default $d \in D$, we have an edge ad if variable $a \in \text{Vars}(d)$ occurs in d . Moreover, there is an edge ab if either $a, b \in \text{Vars}(\alpha(d))$, or $a, b \in \text{Vars}(\beta(d))$, or $a, b \in \text{Vars}(\gamma(d))$ ². Observe the following connection. For any default theory D , we have that $twI(D) \leq twS(D)$. Note that earlier work [6] uses a special version of the incidence graph $I'(D)$. The graph $I'(D)$ is a supergraph of $I(D)$ and still a bipartite graph, which contains an additional vertex for each subformula of every occurring formula, and corresponding edges between subformulas and variables. Consequently, we obtain the bound $tw(I(D)) \leq tw(I'(D))$.

Example 3. Recall default theory D_1 of Example 1. We observe that graph G in the left part of Fig. 1 is the semi-primal graph of D_1 . ■

² Note that these formulas may also be \top or \perp , which we “simulate” by means of the same formula $v \vee \neg v$ or $v \wedge \neg v$, where variable v does not occur in the default theory.

In our DP algorithms for default logic we need to remember when we can evaluate a formula (prerequisite, justification, or conclusion) for a default, i.e., we have a default and all the variables of the formula in a bag. To that end, we introduce labels of nodes. Since we work along the TD and want a unique point where to evaluate, we restrict a label to the first occurrence when working along the TD. A *labeled tree decomposition (LTD)* \mathcal{T} of a default theory D is a tuple $\mathcal{T} = (T, \chi, \delta)$ where (T, χ) is a TD of $S(D)$ and $\delta : N \rightarrow 2^{\{\alpha, \beta, \gamma\} \times D}$ is a mapping where for any (f, d) in $\{\alpha, \beta, \gamma\} \times D$ it holds that (i) if $(f, d) \in \delta(t)$, then $\{d\} \cup f(d) \subseteq \chi(t)$; and (ii) if $\{d\} \cup f(d) \subseteq \chi(t)$ and there is no descendent t' of t such that $(f, d) \in \delta(t')$, then $(f, d) \in \delta(t)$.

We need special case distinctions for DL. Therefore, we restrict an LTD as follows. For a node $t \in N$ that has exactly one child t' where $\chi(t) = \chi(t')$ and $\delta(t) \neq \emptyset$, we say that $\text{type}(t)$ is *label*. If every node $t \in N$ has at most two children, $\text{type}(t) \in \{\text{leaf}, \text{join}, \text{int}, \text{label}, \text{rem}\}$, bags of leaf nodes and the root are empty, $|\delta(t)| \leq 1$, and $\delta(t) = \emptyset$ for $\text{type}(t) \neq \text{label}$ then the LTD is called *pretty*. It is easy to see that we can construct in linear time a pretty LTD without increasing the width from a nice TD, simply by traversing the tree of the TD and constructing the labels and duplicating nodes t where $\delta(t) \neq \emptyset$. Assume in the following, that we use pretty LTDs, unless mentioned otherwise.

Listing 1: Algorithm $\mathcal{DP}(\mathcal{T})$ for Dynamic Programming on TD \mathcal{T} for DL, cf. [10].

In: Pretty LTD $\mathcal{T} = (T, \chi, \delta)$ with $T = (N, \cdot, n)$ of the semi-primal graph $S(D)$.
Out: A table for each node $t \in T$ stored in a mapping $\text{Tables}[t]$.

```

1 for iterate  $t$  in post-order( $T, n$ ) do
2   Child-Tabs := {Tables[ $t'$ ] |  $t'$  is a child of  $t$  in  $T$ }
3   Tables[ $t$ ]  $\leftarrow$  SPRIM( $t, \chi(t), \delta(t), D_t, \text{Child-Tabs}$ )
4 return Tables[ $\cdot$ ]

```

Next, we briefly present the methodology and underlying ideas of our DP algorithms on TDs. The basis for our Algorithm is given in Listing 1 (\mathcal{DP}), which traverses the underlying tree of the given LTD (T, χ, δ) in post-order and runs an algorithm SPRIM at each node $t \in T$. SPRIM computes a new table τ_t based on the tables of the children of t . It has only a “local view” on *bag-defaults*, which are simply the “visible” defaults, i.e., $D_t := D \cap \chi(t)$. Intuitively, we store in each table information such as partial assignments of D_t , that is necessary to locally decide the default theory without storing information beyond variables that belong to the bag $\chi(t)$. Further, the *default theory below* t is defined as $D_{\leq t} := \{d \mid d \in D_{t'}, t' \in \text{post-order}(T, t)\}$, and the *default theory strictly below* t is $D_{< t} := D_{\leq t} \setminus D_t$. For root n of T , it holds that $D_{\leq n} = D_{< n} = D$.

Example 4. Intuitively, the LTD of Fig. 1 enables us to evaluate D by analyzing sub-theories ($\{d_1\}$ and $\{d_2\}$) and combining results agreeing on a, b . Indeed, for the given LTD of Fig. 1, $D_{\leq t_1} = \{d_1\}$, $D_{\leq t_2} = \{d_2\}$ and $D = D_{\leq t_3} = D_{< t_3} = D_{\leq t_1} \cup D_{\leq t_2}$. ■

The next section deals with the details of SPRIM. Before, we need a notion to talk about the result of sequences of a computation. For a node t , the Algorithm SPRIM stores tuples in a table τ_t based on a computation that depends

on tuples (*originating tuples*) that are stored in the table(s) of the child nodes. In order to talk in informal explanations about properties that tuples or parts of tuples have when looking at the entire computation from the relevant leaves up to the node t in the post-order, we need a notion similar to a default theory below t for parts of tuples. Assume for now that our tuples in tables are only tuples of sets. Then, we collect recursively in pre-order along the induced subtree T' of T rooted at t a sequence s of originating tuples $(\mathbf{u}, \mathbf{u}_1, \dots, \mathbf{u}_m)$. If the set T occurs in position i of tuple \mathbf{u} , our notion $T^{\leq t}(s)$ takes the union over all sets T, T_1, \dots, T_m at position i in the tuples $\mathbf{u}_1, \dots, \mathbf{u}_m$. Since a node of type *rem* will typically result in multiple originating tuples, we have multiple sequences s_1, \dots, s_m of originating tuples in general. This results in a family $\mathcal{T}^{\leq t} := \{T^{\leq t}(s) \mid s \in \{s_1, \dots, s_m\}\}$ of such sets. However, when stating properties, we are usually only interested in the fact that each $S \in \mathcal{T}^{\leq t}$ satisfies the property. To this end, we refer to $T^{\leq t}$ as any arbitrary $S \in \mathcal{T}^{\leq t}$. Further, we let $T^{< t} := T^{\leq t} \setminus T$. The definition vacuously extends to nested tuples and families of sets. A more formal compact definition provide so-called extension pointers [14].

Example 5. Recall the given TD in Fig. 1 (right). For illustrating notation, we remove node t_2 , since we only care about nodes t_1 and t_3 and thereby obtain a simpler TD $\mathcal{T} = (T, \chi, \delta)$ (of some simpler graph). Assume that for both nodes t in T we store a table of tuples, say of the form $\langle X, Y \rangle$, where X is a subset of the bag $\chi(t)$ and Y is a set of subsets of $\chi(t)$. Further, let the tables τ_i for the two nodes in this example be as follows: $\tau_1 := \{\mathbf{u}_{1.1} = \{\{d_1\}, \{\emptyset, \{d_1\}, \{d_1, b\}\}\}, \mathbf{u}_{1.2} = \{\{\{a\}, \{\{b\}\}\}\}$, and $\tau_3 := \{\mathbf{u}_{3.1} = \{\emptyset, \{\{a\}\}\}\}$. Then, we let tuple $\mathbf{u}_{3.1}$ originate from tuple $\mathbf{u}_{1.1}$ of child table τ_1 and not from $\mathbf{u}_{1.2}$. We discuss only the Y part of tuple $\mathbf{u}_{3.1}$ (referred to by $Y_{3.1}$). In order to talk about any “extension” $Y_{3.1.1}^{\leq t} \supseteq Y_{3.1.1}$ of $Y_{3.1.1} = \{a\}$ in \mathcal{T} , we write $Y_{3.1.1}^{\leq t}$, which can be one of $\{a\}$, $\{a, d_1\}$, or $\{a, d_1, b\}$. ■

4 Computing Stable Default Sets

In this section, we present our table algorithm SPRIM. Therefore, let D be a given default theory and $\mathcal{T} = (T, \chi, \delta)$ a pretty LTD of $S(D)$.

Our table algorithm follows Definition 2, which consists of two parts: (i) finding sets of satisfying default sets of the default theory and (ii) generating smaller sets of conclusions for these satisfying default sets in order to invalidate subset minimality. Since, SPRIM has only a “local view” on default theory D , we are only allowed to store parts of satisfying default sets. However, we guarantee that, if for the “visible” part Z of a set of satisfying defaults for any node t of T there is no smaller set of satisfying defaults, then Z can be extended to a stable default set of $D_{<t}$. However, in general Z alone is not sufficient, we require auxiliary information to decide the satisfiability of defaults. We need a way to prove that Z witnessed a satisfying default set $Z^{\leq t}$. In particular, even though each $d \in Z^{\leq t}$ is vacuously c-satisfiable, we have to verify that each default $d \in D \setminus Z^{\leq t}$ is indeed

p -satisfiable or j -satisfiable. In turn, we require a set \mathcal{M} of (partial) assignments of $Z^{\leq t}$. To this end, we store in table τ_t tuples that are of the form $\langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle$, where $Z \subseteq D_t$ and $\mathcal{M} \subseteq 2^X$ for $X = \chi(t) \cap \text{Vars}(D)$. The first three tuple positions cover Part (i) and can be seen as the *witness* part. The last position consists of a set of tuples $\mathcal{C} = \langle \rho, \mathcal{AC}, \mathcal{BC} \rangle$ to handle Part (ii) and can be seen as the *counter-witness* part.

In the following, we describe more details of our tuples. We call Z the *witness set*, since Z *witnesses* the existence of a satisfying default set $Z^{\leq t}$ for a *subtheory* S . Each element M in the set \mathcal{M} of *witness models* witnesses the existence of a model of $F_{\leq t} := \bigwedge_{d \in Z^{\leq t}} \gamma(d)$. For our assumed witness set Z , we require a set \mathcal{P} of *witness proofs*. The set \mathcal{P} consists of tuples of the form $\langle \sigma, \mathcal{A}, \mathcal{B} \rangle$, where $\sigma : D_t \rightarrow \{p, j, c\}$ and $\mathcal{A}, \mathcal{B} \subseteq 2^X$ for $X = \chi(t) \cap \text{Vars}(D)$. The function σ , which we call *states function*, maps each default $d \in D_t$ to a decision state $v \in \{p, j, c\}$ representing the case where d is v -satisfiable. The set \mathcal{A} , which we call the *required p-assignments*, contains an assignment $A \in 2^X$ for *each* default d that is claimed to be p -satisfiable. More formally, there is an assignment $A \in \mathcal{A}$ for each default $d \in \sigma^{-1}(p) \cup D_{<t}$ where $\sigma^{\leq t}(d) = p$ such that there is an assignment $A^{\leq t}$ that satisfies $F_{\leq t} \wedge \neg\alpha(d)$. The set \mathcal{B} , which we call the *refuting j-assignments*, contains an assignment $B \in 2^X$ for certain defaults. Intuitively, for each $B \in \mathcal{B}$ there is a default d in the current bag $\chi(t)$ or was in a bag below t such that there is an assignment $B^{\leq t}$ where the justification is not fulfilled. More formally, there is a $B \in \mathcal{B}$ if there is an assignment $B^{\leq t}$ that satisfies $F_{\leq t} \wedge \beta(d)$ for some default $d \in \sigma^{-1}(j) \cup D_{<t}$ where $\sigma^{\leq t}(d) = j$. In the end, if Z proves the existence of a satisfying default set $Z^{\leq t}$ of theory $D_{<t}$, then there is at least one tuple $\langle \cdot, \cdot, \mathcal{B} \rangle \in \mathcal{P}$ with $\mathcal{B} = \emptyset$. Hence, we require that $\mathcal{B} = \emptyset$ in order to guarantee that each default $d \in D_{<t}$ is j -satisfiable where $\sigma^{\leq t}(d) = j$. To conclude, if table τ_n for (empty) root n contains $\mathbf{u} = \langle Z, \cdot, \mathcal{P}, \mathcal{C} \rangle$ where \mathcal{P} contains $\langle \cdot, \cdot, \emptyset \rangle$, then $Z^{\leq t}$ is a satisfying default set of the default theory D . The main aim of \mathcal{C} is to invalidate the subset-minimality of $Z^{\leq t}$, and will be covered later.

Next, we briefly discuss important cases of Listing 2 for Part (i), which consists only of the first three tuple positions (colored red and green) and ignores the remaining parts of the tuple. We call the resulting table algorithm SCONS, which only concerns about computing satisfying default sets. Let $t \in T$ and $\mathbf{u}' = \langle Z, \mathcal{M}, \mathcal{P}, \cdot \rangle$ a tuple of table τ' for a child node of t and $\langle \sigma, \mathcal{A}, \mathcal{B} \rangle$ a tuple in \mathcal{P} . We describe informally how we transform \mathbf{u}' tuples into one or more tuples for the table in node t .

If t is of type *int* and a default d is introduced in t , Line 3 guesses whether d is p -satisfiable, j -satisfiable, or c -satisfiable. To this end, $\text{SGuess}_{d,S}(\mathcal{P})$ adds potential proofs to \mathcal{P} where the satisfiability state of d is within S . Lines 5, 7 and 9 cover nodes of type *label* as follows: In Line 5, if (γ, d) is the label and $\sigma(d) = c$, we enforce that each $M \in \mathcal{M}$ is also a model of $\gamma(d)$. $\text{PCond}_d(\mathcal{P})$ only keeps tuples in \mathcal{P} where each $A \in \mathcal{A}$ is a model of $\gamma(d)$. In Line 7, if (α, d) is the label and $\sigma(d) = p$, $\text{PPred}_d(\mathcal{P}, \mathcal{M})$ enforces that each $A \in \mathcal{A}$ within \mathcal{P} is a model of $\neg\alpha(d)$.

Listing 2 (*): Table algorithm SPRIM($t, \chi_t, \delta_t, D_t, \text{Child-Tabs}$).

In: Bag χ_t , label mapping δ_t , bag-theory D_t , and child tables Child-Tabs of t .
Out: Table τ_t .

```

1 if type( $t$ ) = leaf then  $\tau_t \leftarrow \{\langle \emptyset, \{\emptyset\}, \{\langle \emptyset, \emptyset, \emptyset \rangle\}, \emptyset \rangle\}$  /* Abbreviations below. */
2 else if type( $t$ ) = int,  $d \in D_t$  is the introduced default, and  $\tau' \in \text{Child-Tabs}$  then
3    $\tau_t \leftarrow \{ \langle Z_d^+, M, \text{SGuess}_{d, \{c\}}(\mathcal{P}), \text{SGuess}_{d, \{p, j, c\}}(\mathcal{C}) \cup \text{SGuess}_{d, \{p, j\}}(\mathcal{P}, \mathcal{M}) \rangle,$ 
    $\langle Z, M, \text{SGuess}_{d, \{p, j\}}(\mathcal{P}), \text{SGuess}_{d, \{p, j\}}(\mathcal{C}) \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$ 
4 else if type( $t$ ) = label,  $\{(\gamma, d)\} = \delta_t$  is the label of  $t$ ,  $d \in D_t$ , and  $\tau' \in \text{Child-Tabs}$  then
5    $\tau_t \leftarrow \{ \langle Z, \text{Mod}_M(\gamma(d)), \text{PCon}_d(\mathcal{P}), \text{CCon}_d(\mathcal{C}) \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau', d \in Z \} \cup$ 
    $\{ \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau', d \notin Z \}$ 
6 else if type( $t$ ) = label,  $\{(\alpha, d)\} = \delta_t$  is the label of  $t$ ,  $d \in D_t$ , and  $\tau' \in \text{Child-Tabs}$  then
7    $\tau_t \leftarrow \{ \langle Z, M, \text{PPred}_d(\mathcal{P}, \mathcal{M}), \text{CPred}_d(\mathcal{C}) \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$ 
8 else if type( $t$ ) = label,  $\{(\beta, d)\} = \delta_t$  is the label of  $t$ ,  $d \in D_t$ , and  $\tau' \in \text{Child-Tabs}$  then
9    $\tau_t \leftarrow \{ \langle Z, M, \text{PJust}_d(\mathcal{P}, \mathcal{M}), \text{PJust}_d(\mathcal{C}, \mathcal{M}) \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$ 
10 else if type( $t$ ) = int,  $a \in \chi_t$  is the introduced variable, and  $\tau' \in \text{Child-Tabs}$  then
11   $\tau_t \leftarrow \{ \langle Z, M \cup M_a^\omega, \text{PGuess}_a(\mathcal{P}), \text{PGuess}_a(\mathcal{C}) \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$ 
12 else if type( $t$ ) = rem,  $d \notin D_t$  is the removed default, and  $\tau' \in \text{Child-Tabs}$  then
13   $\tau_t \leftarrow \{ \langle Z_d^-, M, \text{SProj}_d(\mathcal{P}), \text{SProj}_d(\mathcal{C}) \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$ 
14 else if type( $t$ ) = rem,  $a \notin \chi_t$  is the removed variable, and  $\tau' \in \text{Child-Tabs}$  then
15   $\tau_t \leftarrow \{ \langle Z, M_a^\omega, \text{AProj}_a(\mathcal{P}), \text{AProj}_a(\mathcal{C}) \rangle \quad | \quad \langle Z, M, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$ 
16 else if type( $t$ ) = join and  $\tau', \tau'' \in \text{Child-Tabs}$  with  $\tau' \neq \tau''$  then
17   $\tau_t \leftarrow \{ \langle Z, M' \cap M'', \mathcal{P}' \bowtie_{M', M''} \mathcal{P}'', (\mathcal{C}' \bowtie_{M', M''} \mathcal{C}'') \cup (\mathcal{P}' \bowtie_{M', M''} \mathcal{C}'') \cup$ 
    $(\mathcal{C}' \bowtie_{M', M''} \mathcal{P}'') \quad | \quad \langle Z, M', \mathcal{P}', \mathcal{C}' \rangle \in \tau', \langle Z, M'', \mathcal{P}'', \mathcal{C}'' \rangle \in \tau'' \}$ 
18 return  $\tau_t$ 
```

$S_e^- := S \setminus \{e\}$, $S_e^\sim := \{S_e^- \mid S \in \mathcal{S}\}$, $S_e^+ := S \cup \{e\}$, and $S_e^\omega := \{S_e^+ \mid S \in \mathcal{S}\}$.

In Line 9, if (β, d) is the label and $\sigma(d) = j$, $\text{PJust}_d(\mathcal{P}, \mathcal{M})$ adds assignments of \mathcal{M} to \mathcal{B} that are also models of $\beta(d)$.

Next, we cover the case, where a variable a is introduced. In Line 11, we increase the existing witness set $M \cup \{a\}$ for each $M \in \mathcal{M}$. $\text{PGuess}_a(\mathcal{P})$ works analogously for \mathcal{B} and computes all potential combinations of every $A \in \mathcal{A}$, where a is either set to true or to false.

In Line 13, we remove default d from Z and $\text{SProj}_d(\mathcal{P})$ removes d from the domain of the mapping σ , since d is not considered anymore. In Line 15, we remove variable a from each $M \in \mathcal{M}$ and $\text{AProj}_a(\mathcal{P})$ works analogously for each assignment of \mathcal{A} and \mathcal{B} .

Finally, if the node is of type *join*, we have a second child and its table τ'' as well as a tuple $\mathbf{u}'' \in \tau''$. Intuitively, tuples \mathbf{u}' and \mathbf{u}'' represent intermediate results of two different branches in T . To combine these results, we have to join the tuples on the witness extension, witness states, and the witness models. The join operation \bowtie can be seen as a combination of inner and outer joins, used in database theory [15]. Note that for instance for an assignment $B \in \mathcal{B}$ to endure within \mathcal{P} of τ_t , it suffices that B is a corresponding witness model in \mathbf{u}'' .

Example 6. Consider default theory D from Example 1 and in Fig. 2 (left) pretty LTD $\mathcal{T} = (\cdot, \chi, \delta)$ of the semi-primal graph $S(D)$ and the tables τ_1, \dots, τ_{18} illustrating computation results obtained during post-order traversal of \mathcal{T} by \mathcal{DP} using SCONS instead of SPRIM in Line 3. We omit the last position of the

tuples, since those are only relevant for SPRIM. Note that we discuss only selected cases, and we assume for presentation that each tuple in a table τ_t is identified by a number, i.e., the i -th tuple corresponds to $\mathbf{u}_{t,i} = \langle Z_{t,i}, \mathcal{M}_{t,i}, \mathcal{P}_{t,i}, \mathcal{C}_{t,i} \rangle$. The numbering naturally extends to sets in witness proofs and counter-witnesses.

We obtain table $\tau_1 = \{\langle \emptyset, \{\emptyset\}, \{\langle \emptyset, \emptyset, \emptyset \rangle\} \rangle\}$ as $\text{type}(t_1) = \text{leaf}$ (see Line 1).

Since $\text{type}(t_2) = \text{int}$ and a is the introduced variable, we construct table τ_2 from τ_1 by modifying $\mathcal{M}_{2,1}$ and $\mathcal{P}_{2,1} = \{\langle \sigma_{1,1}, \mathcal{A}_{1,1}, \mathcal{M}_{2,1} \rangle\}$, where $\mathcal{M}_{2,1}$ contains $M_{1,1,k}$ and $M_{1,1,k} \cup \{a\}$ for each $M_{1,1,k}$ ($k \leq 1$) in τ_1 . This corresponds to a guess on a . Precisely, $\mathcal{M}_{2,1} := \{\emptyset, \{a\}\}$ (Line 11).

Then, t_3 introduces default d_1 , which results in two tuples. In tuple $\mathbf{u}_{3,1}$ default d_1 is p -satisfiable or j -satisfiable due to $\alpha(d_1)$ or $\beta(d_1)$ (see $\mathcal{P}_{3,1}$, Line 3).

In tuple $\mathbf{u}_{3,2}$ default d_1 is c -satisfiable and we have that $Z_{3,2} = \{d_1\}$ and $\mathcal{P}_{3,2} = \{\{d_1 \mapsto c\}, \emptyset, \emptyset\}$.

Node t_4 introduces label (β, d_1) and modifies $\mathcal{P}_{4,1,2}$. In particular, it chooses among \mathcal{M} candidates, which might contradict that d_1 is j -satisfiable (see Line 9). Obviously, we have that $\mathcal{B}_{4,1,2} = \{\{a\}\}$, since $\beta(d_1) = a$.

In table τ_5 , we present the case where default d_1 should be p -satisfiable. In this case since $\alpha(d_1) = \top$, we do not find any model of \perp . In consequence, there is no corresponding successor of $\mathcal{P}_{4,1,1}$ in τ_5 , i.e., in τ_5 it turns out that d_1 can not be p -satisfiable.

Table τ_7 concerns the conclusion $\gamma(d_1)$ of a default. It updates every assignment occurring in the table, such that the models satisfy $\gamma(d_1)$ if d_1 is c -satisfiable. The remaining cases work similarly.

In the end, join node t_{16} just combines witnesses agreeing on its content. ■

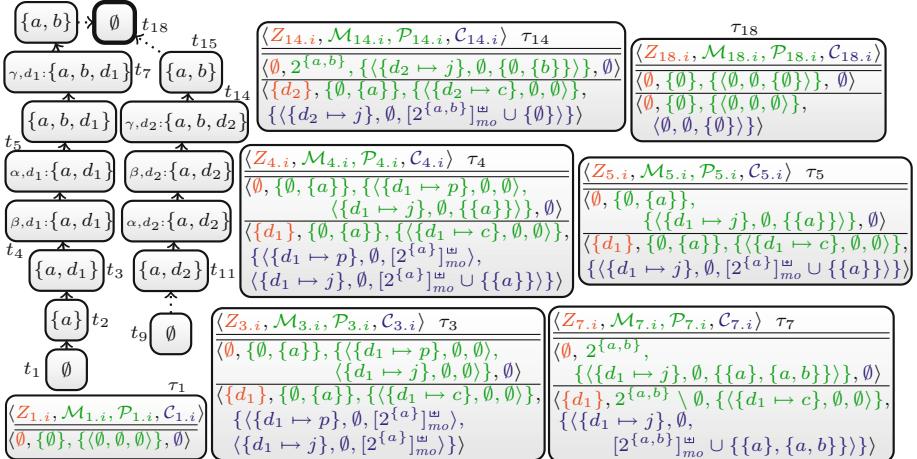


Fig. 2. Selected DP tables of SPRIM for pretty LTD \mathcal{T} .

Next, we briefly discuss the handling of counter-witnesses, which completes Algorithm SPRIM. The handling of counter-witnesses \mathcal{C} is quite similar to the

witness proofs \mathcal{P} . The tuples $\langle \rho, \mathcal{AC}, \mathcal{BC} \rangle \in \mathcal{C}$ consist of a states function $\rho : D_{\leq t} \mapsto \{p, j, c\}$, required p -assignments $\mathcal{AC} \subseteq 2^X$ and refuting j -assignments $\mathcal{BC} \subseteq 2^{(X \cup \{mo\})}$ for $X = \text{Vars}_{\leq t} \cap \chi(t)$. In contrast to the refuting j -assignments in \mathcal{B} , \mathcal{BC} may in addition contain an assignment $B \in \mathcal{BC}$ with a marker mo . The marker indicates that $B^{\leq t}$ is actually not refuting, but only a model of $\gamma(d)$ for each default below t that is c -satisfiable, i.e., $\bigwedge_{d \in D_{\leq t}, \rho^{\leq t}(d)=c} \gamma(d)$. In other words, those assignments setting mo to true are the counter-witness assignments that do not refute c -assignments (comparable to witness assignments in \mathcal{M} for Part (ii)).

The existence of a certain counter-witness tuple for a witness in a table τ_t establishes that the corresponding witness can *not* be extended to a stable default set of $D_{\leq t}$. In particular, there exists a stable extension for D if the table τ_n for root n contains a tuple of the form $\langle \emptyset, \{\emptyset\}, \mathcal{P}, \mathcal{C} \rangle$, where $\mathcal{P} \neq \emptyset$ and contains tuples of the form $\langle \cdot, \cdot, \emptyset \rangle$. Moreover, for each $\langle \rho, \mathcal{AC}, \mathcal{BC} \rangle \in \mathcal{C}$ there is $\emptyset \in \mathcal{BC}$ indicating a true refuting j -assignment for the empty root n . Intuitively, this establishes that there is no actual counter-witness, which contradicts that the corresponding satisfying default $Z^{\leq t}$ is subset-minimal and hence indeed a stable default set.

Due to space limitations, we omit a full description of both Parts (i) and (ii) together for our algorithm. A major difference of Part (ii) is that we need a special function $\text{CCon}_d(\mathcal{C})$ to establish that a default d is j -satisfiable, which is defined with respect to fixed set S , c.f., Case (ii) of Definition 2. Then, $\text{CCon}_d(\mathcal{C})$ additionally adds potential proofs involving counter-witnesses and mo models, where $\rho(d) \neq c$, but $\sigma(d) = c$.

In the following, we state the correctness of the algorithm \mathcal{DP} .

Theorem 1 (*). *Given a default theory D , algorithm \mathcal{DP} correctly solves EXT.*

Proof (Idea). The correctness proof of this algorithm needs to investigate each node type separately. We have to show that a tuple at a node t guarantees existence of a stable default set for a sub-theory of theory $D_{\leq t}$, which proves soundness. Conversely, one can show that each stable default set is indeed evaluated while traversing the pretty LTD, which establishes completeness.

Next, we establish that we can extend \mathcal{DP} to enumerate stable default sets. The algorithm on top of \mathcal{DP} is relatively straight forward, which can be found in an extended version. The idea is to compute a first stable default set in linear time, followed by systematically enumerating subsequent solutions with linear delay. One can even further extend \mathcal{DP} to solve #SE, similar to related work [10] in a slightly different context.

Theorem 2 (*). *Given a default theory D , algorithm SPRIM can be used as a preprocessing step to construct tables from which we can solve problem ENUMSE.*

Proof (Idea). The correctness proof requires to extend the previous results to establish a one-to-one correspondence when traversing the tree of the TD and such that we can reconstruct each solution as well as we do not get duplicates. The proof proceeds similar to Theorem 1.

The following theorem states that we obtain threefold exponential runtime in the treewidth.

Theorem 3 (★). *Algorithm DP runs in time $\mathcal{O}(2^{2^{2^k+4}} \cdot \|S(D)\|)$ for a given default theory D , where $k := \text{tw}(S(D))$ is the treewidth of semi-primal graph $S(D)$.*

5 Conclusion

In this paper, we established algorithms that operate on tree decompositions of the semi-primal graph of a given default theory. Our algorithms can be used to decide whether the default theory has a stable extension or to enumerate all stable default sets. The algorithms assume small treewidth and run in linear time and with linear delay, respectively. Even though already linear time results for checking the existence of a stable extension are known, we are able to establish runtime that is only triple exponential in the treewidth of the semi-primal graph.

In order to simplify the presentation, we mainly covered the semi-primal graph. However, we believe that our algorithms can be extended to tree decompositions of the incidence graph. Then we need additional states to handle the cases where prerequisite, justification, and conclusion do not occur together in one bag. Consequently, such an algorithm will likely be very complex. Further, we also believe that our algorithm can be extended to disjunctive defaults [16], where we have to guess which of the conclusion parts is to apply. An interesting research question is whether we can improve our runtime bounds. Still it might be worth implementing our algorithms to enumerate stable default sets for DL, as previous work showed that a relatively bad worst-case runtime may anyways lead to practical useful results [9].

References

1. Reiter, R.: A logic for default reasoning. *AIJ* **13**, 81–132 (1980)
2. Marek, V.W., Truszczyński, M.: Nonmonotonic Logic: Context-dependent Reasoning. *Artificial Intelligence*. Springer, Berlin (1993). <https://doi.org/10.1007/978-3-662-02906-0>
3. Gottlob, G.: Complexity results for nonmonotonic logics. *JLC* **2**(3), 397–425 (1992)
4. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Cham (2015). <https://doi.org/10.1007/978-3-319-21275-3>
5. Fichte, J.K., Meier, A., Schindler, I.: Strong backdoors for default logic. In: Creignou, N., Le Berre, D. (eds.) *SAT 2016*. LNCS, vol. 9710, pp. 45–59. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_4
6. Meier, A., Schindler, I., Schmidt, J., Thomas, M., Vollmer, H.: On the parameterized complexity of non-monotonic logics. *Arch. Math. Logic* **54**(5–6), 685–710 (2015)
7. Courcelle, B.: Graph rewriting: an algebraic and logic approach. In: van Leeuwen, J. (ed.) *Handbook of theoretical computer science*. Volume Formal Models and Semantics, vol. B, pp. 193–242. Elsevier Science Publishers, North-Holland (1990)

8. Kneis, J., Langer, A.: A practical approach to Courcelle's theorem. *Electron. Notes Theor. Comput. Sci.* **251**, 65–81 (2009)
9. Charwat, G., Woltran, S.: Dynamic programming-based QBF solving. In: Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF 2016), pp. 27–40 (2016)
10. Fichte, J.K., Hecher, M., Morak, M., Woltran, S.: Answer set solving with bounded treewidth revisited. In: Balduccini, M., Janhunen, T. (eds.) LPNMR 2017. LNCS (LNAI), vol. 10377, pp. 132–145. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61660-5_13
11. Fichte, J.K., Hecher, M., Morak, M., Woltran, S.: DynASP2.5: dynamic programming on tree decompositions in action. In: Proceedings of the 12th IPEC (2017)
12. Creignou, N., Meier, A., Müller, J.S., Schmidt, J., Vollmer, H.: Paradigms for parameterized enumeration. *Th. Comput. Syst.* **60**(4), 737–758 (2017)
13. Bodlaender, H., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. *Comput. J.* **51**(3), 255–269 (2008)
14. Bliem, B., Charwat, G., Hecher, M., Woltran, S.: D-FLAT² subset minimization in dynamic programming on tree decompositions made easy. *FI* **147**, 27–34 (2016)
15. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases: The Logical Level, 1st edn. Addison-Wesley, Boston (1995)
16. Gelfond, M., Lifschitz, V., Przymusinska, H., Truszczynski, M.: Disjunctive Defaults, pp. 230–237. Morgan Kaufmann, Burlington (1991)



A General Class of Monoids Supporting Canonisation and Minimisation of (Sub)sequential Transducers

Stefan Gerdjikov^{1,2(✉)}

¹ Faculty of Mathematics and Informatics, Sofia University,
5, James Bourchier Blvd., 1164 Sofia, Bulgaria
sstefangerdzhikov@fmi.uni-sofia.bg

² Institute of Information and Communication Technologies,
Bulgarian Academy of Sciences, 25A, Acad. G. Bonchev Str., 1113 Sofia, Bulgaria

Abstract. In this paper we consider the problems of canonisation and minimisation of subsequential transducer with output in an arbitrary monoid. We show that these problems can be efficiently solved for a large class of monoids that includes the free monoids, tropical monoid, and groups, and is closed under Cartesian Product. We describe this class of monoids in terms of five simple axioms. The first four of them seem to be natural. For the last one, we show that it is also necessary.

Keywords: Sequential transducers · Minimisation · Canonisation
Monoids

1 Introduction

Subsequential transducers provide a natural and effective way to represent functions that map words to words, to real numbers, or, more generally, to elements of an arbitrary monoid, [2–4, 13]. They also facilitate an efficient on-line processing of words which makes them an attractive formalism for Natural Language Modelling [8–12]. In this paper we consider two classical problems related to subsequential transducers: *canonisation* and *minimisation*.

The canonical subsequential transducers provide the maximal piece of the output that is currently unambiguously defined. In the special case of free monoids and tropical monoid this problem has been considered in [1, 11] where also efficient algorithms for its solution have been proposed. In this paper we generalise the notion of canonical transducer to arbitrary monoids. Specifically, we consider the preorder on monoidal elements $a \leq b$ iff $b = ac$ for some c . In this notion, the longest common beginning of several elements of the monoid translates to an infimum of those elements.

The minimisation of subsequential transducers is the problem to find a subsequential transducer with as few states as possible that is equivalent to a given one. The problem in the case of free monoids and real numbers with addition

has been also thoroughly studied and efficient algorithms in these cases are well known, [4, 11]. These algorithms can be considered as refinement of the classical minimisation [7] of deterministic automata applied on canonical subsequential transducers. Thus, it is not surprising that being able to canonise a subsequential transducer, it is also possible to minimise it. This is what we do. Yet, the invertible elements in the monoid lead to complications that we show how to resolve.

In this paper we define a class of monoids for which the canonisation and minimisation problem are algorithmically tractable. This is the class of monoids with the following five properties: (i) left and (ii) right cancellation; (iii) lower semi-lattice under \leq ; (iv) pairs of elements with upper bound have also a least upper bound; (v) $b \leq c$ and $b \leq ac$ implies $b \leq ab$.

After some preliminaries in Sect. 2, we formally state and discuss the above five properties in Sect. 3. In Sect. 4 we describe a general construction for canonisation, and Sect. 5 presents a construction for minimisation of subsequential transducers. In Sect. 6 we argue that property (v) is also necessary. We conclude in Sect. 7.

2 Preliminaries

A *monoid* $\mathcal{M} = \langle M, \circ, e \rangle$ is a semigroup $\langle M, \circ \rangle$ with a unit element e . A special class of monoids are the free monoids, Σ^* , generated by a finite set Σ . The support of Σ^* is the set of *words* over Σ , i.e. the finite sequences of elements of Σ . The product is the concatenation of words, and the unit element is the empty word. Groups represent another class of monoids. Another example of a monoid is the set of non-negative real numbers with addition, $\langle \mathbb{R}^+, +, 0 \rangle$. We refer to this monoid as *tropical monoid*.

For monoids $\mathcal{M}_i = \langle M_i, \circ_i, e_i \rangle$ for $i = 1, 2$, the Cartesian Product, $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$, is defined as $\mathcal{M} = \langle M_1 \times M_2, \circ, \langle e_1, e_2 \rangle \rangle$, where:

$$\langle a_1, a_2 \rangle \circ \langle b_1, b_2 \rangle = \langle a_1 \circ_1 b_1, a_2 \circ_2 b_2 \rangle.$$

We note that the Cartesian Product of monoids is a monoid.

For a monoid \mathcal{M} and a set S we denote with $\mathbf{e}_S : S \rightarrow M$ the identity function, i.e. $\mathbf{e}_S(s) = e$ for $s \in S$. For a singleton, $S = \{s\}$ we use \mathbf{e}_s for $\mathbf{e}_{\{s\}}$.

An *automaton*¹ over a monoid \mathcal{M} is a tuple $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta, \iota, \Psi \rangle$ where Q is a finite set of *states*, $I, F \subseteq Q$ are sets of *initial* and *final* states, respectively, $\Delta \subseteq Q \times M \times Q$ is a finite set of *transitions*, and $\iota : I \rightarrow M$ and $\Psi : F \rightarrow M$ are *initial* and *final* functions.

A *path* in an automaton \mathcal{A} is a possibly empty sequence of transitions:

$$\pi = \langle p_0, m_1, p_1 \rangle \dots \langle p_{n-1}, m_n, p_n \rangle.$$

¹ We adopt the definition from [4] so that (sub)sequential transducers can be considered as a special kind of automata.

We call $\sigma(\pi) = p_0$ *initial state* of π and $\tau(\pi) = p_n$ – *terminal state* of π . $|\pi| = n$ is the *length* of π and the product $\ell(\pi) = \prod_{i=1}^n m_i$ is the *label* of the path π . By default, the trivial path is of length 0 and its label is the unit element, e .

A path π in \mathcal{A} is called *successful* if $\sigma(\pi) \in I$ and $\tau(\pi) \in F$. In these terms, the *language* of an automaton $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta, \iota, \Psi \rangle$ is:

$$\mathcal{L}(\mathcal{A}) = \{\iota(\sigma(\pi)) \circ \ell(\pi) \circ \Psi(\tau(\pi)) \mid \pi \text{ is a successful path in } \mathcal{A}\}.$$

For a state $p \in Q$, we define the automaton \mathcal{A}_p as $\mathcal{A}_p = \langle \mathcal{M}, Q, \{p\}, F, \Delta, \mathbf{e}_p, \Psi \rangle$. We shall often write $\mathcal{L}(p)$ as a short hand for $\mathcal{L}(\mathcal{A}_p)$.

A state p is called *accessible* in \mathcal{A} if there is a path π with $\sigma(\pi) \in I$ and $\tau(\pi) = p$. A state p is called *co-accessible* in \mathcal{A} if there is a path π with $\sigma(\pi) = p$ and $\tau(\pi) \in F$. An automaton is called *trimmed* if every state of the automaton is both accessible and co-accessible.

We shall also use the standard notions Δ^* and $\Delta^{\leq n}$. Formally:

$$\begin{aligned}\Delta^* &= \{\langle \sigma(\pi), \ell(\pi), \tau(\pi) \rangle \mid \pi \text{ is a path}\} \\ \Delta^{\leq n} &= \{\langle \sigma(\pi), \ell(\pi), \tau(\pi) \rangle \mid \pi \text{ is a path with } |\pi| \leq n\}.\end{aligned}$$

A $\Sigma - \mathcal{M}$ -*transducer* is an automaton: $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta, \iota, \Psi \rangle$, where Σ is a finite set. Thus, the language of \mathcal{T} is a relation, $\mathcal{L}(\mathcal{T}) \subseteq \Sigma^* \times \mathcal{M}$. A transducer is called *one-letter* transducer if, additionally, $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times M \times Q$, $Rng(\iota) \subseteq \{\varepsilon\} \times M$, and $Rng(\Psi) \subseteq \{\varepsilon\} \times M$. Clearly, for a one-letter transducers we can identify ι and Ψ with their second projections $\iota_2 : I \rightarrow M$ and $\Psi_2 : F \rightarrow M$, respectively. To simplify the notation, we shall do so when no confusion is possible.

A $\Sigma - \mathcal{M}$ -transducer is called *(sub)sequential* if:

1. $I = \{s\}$ is a singleton,
2. there are (partial) functions $\delta : Q \times \Sigma \rightarrow Q$ and $\lambda : Q \times \Sigma \rightarrow Q$ with $Dom(\delta) = Dom(\lambda)$, such that:

$$\Delta = \{\langle p, \langle a, \lambda(p, a) \rangle, \delta(p, a) \rangle \mid \langle p, a \rangle \in Dom(\delta)\}.$$

To stress these particularities of the (sub)sequential transducers we denote them as: $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$.

The functions, $\delta^* : Q \times \Sigma^* \rightarrow Q$ and $\lambda^* : Q \times \Sigma^* \rightarrow M$ are the natural extensions of δ and λ , respectively. In particular, $Dom(\lambda^*) = Dom(\delta^*)$ and:

$$\Delta^* = \{\langle p, \langle w, \lambda^*(p, w) \rangle, \delta^*(p, w) \rangle \mid \langle p, w \rangle \in Dom(\delta^*)\}.$$

Clearly, a (sub)sequential transducer, \mathcal{T} , recognises a graph of a function mapping Σ^* to M , as opposed to general (rational) relation of Σ^* and M . We denote this function with $\mathcal{O}_{\mathcal{T}} : \Sigma^* \rightarrow M$. One can easily see that for each $w \in \Sigma^*$:

$$\mathcal{O}_{\mathcal{T}}(w) = \iota(s) \circ \lambda^*(s, w) \circ \Psi(\delta^*(s, w)) \text{ iff } \delta^*(s, w) \in F.$$

In our considerations we shall denote with $\mathcal{O}_{\mathcal{T}}^{(q)} = \mathcal{O}_{\mathcal{T}_q}$ the function that corresponds to the (sub)sequential transducer \mathcal{T}_q .

3 Monoids

In this section we define the subclass of monoids that we shall be interested in. We also introduce the notion of *infimum* of a subset of a monoid that generalises the notion of longest common prefix for languages and infimum for real numbers. Interestingly, all the notions and properties that we introduce make sense and are valid for free monoids, tropical monoid, and, more generally, sequentiable structures [5], and also for groups. Furthermore, all the properties we are looking at are closed under Cartesian Product of monoids.

Definition 1. For a monoid \mathcal{M} and elements $a, b \in M$ we say that $a \leq_M b$ if there is an element $c \in M$ with $a \circ c = b$.

Remark 1. It should be clear that \leq_M defines a pre-order on \mathcal{M} . Furthermore, if \mathcal{M} is a group any two elements a and b satisfy $a \leq_M b$. On the other hand for a free monoid \mathcal{M} , $a \leq_M b$ means that a is a prefix of b . Finally, for the tropical monoid, $(\mathbb{R}^+, +, 0)$, $a \leq_M b$ means $a \leq b$ as real numbers.

Definition 2. For a subset $S \subseteq M$ of a monoid \mathcal{M} we introduce the lower and upper bounds for S as:

$$\text{low}(S) = \{l \mid \forall s \in S (l \leq_M s)\} \quad \text{up}(S) = \{u \mid \forall s \in S (s \leq_M u)\}.$$

The infimum and supremum sets for S are defined as:

$$\inf S = \text{low}(S) \cap \text{up}(\text{low}(S)) \text{ and } \sup S = \text{up}(S) \cap \text{low}(\text{up}(S)).$$

Remark 2. For all S , $e \in \text{low}(S)$. However, $\text{up}(S)$ can be empty even if $S \neq \emptyset$, consider the free monoid on two incomparable words. Even if $\text{low}(S) \neq \emptyset$, it should not be the case that $\inf S \neq \emptyset$ – one may consider $(\mathbb{Q}^+, +, 0)$ and arbitrary decreasing sequence tending to an irrational number. The same holds for $\sup S$.

Remark 3. Note, that in the case where \mathcal{M} is a group, for any $S \subseteq M$ we have $\text{low}(S) = \text{up}(S) = \inf S = \sup S = M$.

Definition 3 (LSL-axiom). We say that a monoid \mathcal{M} satisfies the Lower Semi-Lattice axiom (LSL-axiom), if for any two elements $a, b \in M$, $\inf\{a, b\} \neq \emptyset$.

If \mathcal{M} satisfies the LSL-axiom we shall denote with $a \sqcap b$ an arbitrary, but fixed, element in $\inf\{a, b\}$. For a finite sequence of elements $\{a_i\}_{i=1}^n$ we shall use $\prod_{i=1}^n a_i := (\dots((a_1 \sqcap a_2) \sqcap a_3) \dots \sqcap a_{n-1}) \sqcap a_n$.

Remark 4. The groups, free monoids (longest common prefix), and the tropical monoid (minimum), satisfy the LSL-axiom.

Lemma 1. If monoids \mathcal{M}_1 and \mathcal{M}_2 obey the LSL-axiom, then so does $\mathcal{M}_1 \times \mathcal{M}_2$. \square

The symmetric requirement, i.e. $\sup\{a, b\} \neq \emptyset$, would impose on the monoid structure similar to a lattice. However, it is too strong and natural monoids, say the free monoids, violate this condition. We go for a much weaker requirement:

Definition 4 (RMGE-axiom). We say that a monoid \mathcal{M} satisfies the Right Most General Equaliser Axiom (RMGE-axiom) if for any two elements $a, b \in M$ it holds:

$$\text{up}(\{a, b\}) \neq \emptyset \Rightarrow \sup\{a, b\} \neq \emptyset.$$

If \mathcal{M} satisfies the RMGE-axiom we shall denote with $a \vee b$ an arbitrary, but fixed, element in $\sup\{a, b\}$, if such exists. For a finite sequence of elements $\{a_i\}_{i=1}^n$ we shall use $\bigvee_{i=1}^n a_i := (\dots ((a_1 \vee a_2) \vee a_3) \dots \vee a_{n-1}) \vee a_n$.

Remark 5. Trivially, any group satisfies the RMGE-axiom. Further, in a free monoid Σ^* words α and β have an upper bound if and only if one of the words is a prefix of the other. In this case, the longer word is easily seen to be a supremum for α and β . Finally, for the tropical monoid, the maximum of two real numbers a and b is clearly their supremum.

Lemma 2. If monoids \mathcal{M}_1 and \mathcal{M}_2 obey the RMGE-axiom, then so does $\mathcal{M}_1 \times \mathcal{M}_2$. \square

The next two properties that we shall consider are the *left* and *right* cancellation properties of a monoid. Formally:

Definition 5 (LC-axiom). The Left Cancellation Axiom (LC-axiom) for a monoid \mathcal{M} states that for every $a, b, c \in M$: $ca = cb \Rightarrow a = b$.

For elements $c, d \in M$ with² $c \leq_M d$, we denote with $\frac{d}{c}$ the unique element in M s.t.: $c \circ \frac{d}{c} = d$.

Definition 6 (RC-axiom). The Right Cancellation Axiom (RC-axiom) for a monoid \mathcal{M} states that for every $a, b, c \in M$: $ac = bc \Rightarrow a = b$.

If $c, d \in M$ and there is an invertible element $a \in M$ such that $ac = d$ we denote with $d \ominus c$ the unique element a with this property.

Remark 6. Clearly, groups satisfy the LC and RC-axioms. Further, free monoids (think of suffixes and prefixes) satisfy the LC and RC-axioms. Finally, the tropical monoid also satisfies the LC and RC-axioms (consider the difference of real numbers).

Definition 7. A monoid \mathcal{M} satisfying the RMGE-, LC-, and RC-axioms is called a most general equaliser (mge) monoid.

By the above discussion, it should be clear that groups, free monoids, and tropical monoid are mge monoid. Furthermore, it also follows that mge monoids are closed under Cartesian Product.

Next two lemmata describe some interesting properties of mge monoids.

Lemma 3. Let \mathcal{M} be an mge monoid. If $a, b \in M$ have an upper bound, then:

$$1. a \frac{a \vee b}{a} = b \frac{a \vee b}{b}.$$

² Note that $c \leq_M d$ is equivalent to the existence of an element a with $ca = d$.

2. if $ax = by$ for some $x, y \in M$, then $x = \frac{a \vee b}{a} z$ and $y = \frac{a \vee b}{b} z$ for some $z \in M$.

Lemma 4. Let \mathcal{M} be an mge monoid, $S \subseteq M$ be a nonempty subset of M and $m \in M$ be arbitrary. Then $\inf(mS) = m \inf S$.

The last property of monoids that we shall consider is the following:

Definition 8 (LCB-axiom). We say that a monoid \mathcal{M} satisfies the Greatest Common Left Factor Axiom (GCLF-axiom) if for any $a, b, c \in M$ it holds:

$$b \leq_M c \& b \leq_M ac \Rightarrow b \leq_M ab.$$

For free monoids the GCLF-axiom can be expressed in the following way. If b is a prefix of c , and b is a prefix of ac , then b is a prefix of ab . For words this implication is obvious, because by the premise ab is a prefix of ac and ab is longer than b . Thus, since b is a prefix of ac , it is also a prefix of ab .

Remark 7. The groups (trivially), free monoids (by above), and the tropical monoid (nonnegative reals) satisfy the GCLF-axiom.

Lemma 5. If monoids \mathcal{M}_1 and \mathcal{M}_2 satisfy the GCLF-axiom, then so does $\mathcal{M}_1 \times \mathcal{M}_2$. \square

In the next two sections we shall consider mge monoids with LSL- and GCLF-axioms, i.e. monoids \mathcal{M} satisfying all the axioms defined in this section: LSL-, RMGE-, LC-, RC-, and GCLF-axioms. In the last section we construct a monoid satisfying the first four axioms violates the GCLF-axiom for which the implications of our results are also false.

4 Canonisation of Transducers

Throughout this section we assume that \mathcal{M} is an mge monoid satisfying the LSL- and GCLF-axioms.

Definition 9. For a Σ - \mathcal{M} -transducer $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta, \iota, \Psi \rangle$ and a state $p \in Q$ we define the range of the state p as: $\mathcal{R}(p) = \text{Rng}(\mathcal{T}_p)$. We say that \mathcal{T} is onward if for every $p \in Q$ it holds that $e \in \inf \mathcal{R}(p)$.

By *input-transitions* of \mathcal{T} , we mean $\Delta_{in} = \{\langle p, a, q \rangle \mid \exists m (\langle p, \langle a, m \rangle, q \rangle \in \Delta)\}$, i.e. the set of transitions of \mathcal{T} by dropping their output. The result in this section is:

Theorem 1. For every Σ - \mathcal{M} one-letter transducer, \mathcal{T} , there is an equivalent onward transducer, \mathcal{T}_c , with the same states and the same input-transitions.

The proof of this result for free monoids relies on the computation of the longest common prefix for regular languages, [1, 10, 11]. Afterwards longest common prefixes of the output languages are pushed forward along the transitions. Our approach follows along these lines. However, while we shall have little problems with the second step, it is not quite obvious how to generalise the algorithms

for computation of longest common prefix to an algorithm for a greatest common left factor in the general setting of the monoid \mathcal{M} .

In the sequel, we first describe an algorithm for the computation of Greatest Common Left Factor (GCLF) Problem. Then, we complete the canonisation algorithm of a transducer in a standard way.

Recall that $\mathcal{L}(p) = \mathcal{L}(\mathcal{A}_p)$, $\mathcal{A}_p = \langle \mathcal{M}, Q, \{p\}, F, \Delta, \mathbf{e}_p, \Psi \rangle$, is the right language of the state p . We define the GCLF Problem as:

Given: $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta, \iota, \Psi \rangle$ trimmed automaton.

Output: $\mu(p) \in \inf \mathcal{L}(p)$ for all $p \in Q$.

To solve this problem, we suggest the following standard dynamic programming construction:

1. for each $p \in Q$ select an element $\mu_0(p) \in \mathcal{L}(p)$, s.t. $\mu_0(f) = \Psi(f)$ for $f \in F$.
2. for $k = 0$ to $k = 2|Q| - 2$ for each $p \in Q$ compute:

$$\mu_{k+1}(p) = \mu_k(p) \sqcap \bigcap_{\langle p, m, q \rangle \in \Delta} (m \circ \mu_k(q)).$$

3. Output $\mu(p) = \mu_{2|Q|-1}(p)$.

For a transducer \mathcal{T} , a state p of the \mathcal{T} and an integer k we denote:

$$\mathcal{L}^{\leq k}(p) = \{m \circ \Psi(f) \mid f \in F \text{ and } \langle p, m, f \rangle \in \Delta^{\leq k}\}.$$

We denote with $\mathcal{L}^{\leq k}(\mathcal{A}) = \bigcup_{i \in I} \mathcal{L}^{\leq k}(i)$. The correctness of the suggested construction relies on the following lemma:

Lemma 6. Let $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta, \mathbf{e}_I, \Psi \rangle$ be a trimmed automaton and let $n = 2|Q| - 1$. Then, $\text{low}(\mathcal{L}(\mathcal{A})) = \text{low}(\mathcal{L}^{\leq n}(\mathcal{A}))$.

Proof. Let $L = \text{low}(\mathcal{L}(\mathcal{A}))$ and $L_n = \text{low}(\mathcal{L}^{\leq n}(\mathcal{A}))$. Since $\mathcal{L}^{\leq n}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A})$, $L \subseteq L_n$. We prove that the converse is also true.

To this end, let us fix an element $l \in L_n$. For an element $m \in \mathcal{L}(\mathcal{A})$ we set $\kappa = \kappa(m)$ to be the minimal k s.t. $m \in \mathcal{L}^{\leq k}(\mathcal{A})$. Let $P(k)$ be the statement:

$$P(k) : \forall m \in \mathcal{L}(\mathcal{A}) (\kappa(m) = k \Rightarrow l \leq_M m).$$

We prove that $P(k)$ holds true for every $k \in \mathbb{N}$ by induction on k . For $k \leq n$ we have that the premise of $P(k)$ already implies that $m \in L_n$ and therefore $l \leq_M m$ because $l \in L_n$.

Let $k \in \mathbb{N}$ be such that $k \geq n + 1$ and $P(k')$ is true for all $k' < k$. Let $m \in \mathcal{L}(\mathcal{A})$ and $\kappa(m) = k$. Let π be any successful path with $m = \ell(\pi) \circ \Psi(\tau(\pi))$ and $|\pi| = k$. By the definition of $\kappa(m)$ such a path exists. Let π be given as:

$$\pi : \langle p_0, m_1, p_1 \rangle \langle p_1, m_2, p_2 \rangle \dots \langle p_{k-1}, m_k, p_k \rangle.$$

Since $k \geq n + 1 = 2|Q|$ there is a state $p \in Q$ and indices $0 \leq i_1 < i_2 < i_3 \leq k$ such that $p_{i_j} = p$ for $j = 1, 2, 3$. Setting $i_0 = 0$ and $i_4 = k$ the three fixed occurrences of p split the path π into four (sub)paths:

$$\pi_j : \langle p_{i_{j-1}}, m_{i_{j-1}+1}, p_{i_{j-1}+1} \rangle \dots \langle p_{i_j-1}, m_{i_j}, p_{i_j} \rangle \text{ for } j = 1, 2, 3, 4.$$

Since $i_1 < i_2 < i_3$, π_2 and π_3 are non-trivial. Let $m^{(j)} = \ell(\pi_j)$ be the label of the path π_j for $j = 1, 2, 3, 4$. Thus: $\pi_1\pi_4$, $\pi_1\pi_2\pi_4$, $\pi_1\pi_3\pi_4$, and $\pi = \pi_1\pi_2\pi_3\pi_4$ are all successful paths and furthermore the first three have length strictly less than k . Let: $x = m^{(1)}m^{(4)}\Psi(p_k)$, $y = m^{(1)}m^{(2)}m^{(4)}\Psi(p_k)$, and $z = m^{(1)}m^{(3)}m^{(4)}\Psi(p_k)$. Thus, by the discussion above we have that $x, y, z \in \mathcal{L}(\mathcal{A})$, $\kappa(x), \kappa(y), \kappa(z) < k$ and by the induction hypothesis: $l \leq_M x$, $l \leq_M y$, and $l \leq_M z$.

This shows that $x, y, z \in up(\{l, m^{(1)}\})$. Hence $l \vee m^{(1)}$ is defined. Let $v = \frac{l \vee m^{(1)}}{m^{(1)}}$. Hence: $v \leq_M m^{(4)}\Psi(p_k)$, $v \leq_M m^{(2)}m^{(4)}\Psi(p_k)$, and $v \leq_M m^{(3)}m^{(4)}\Psi(p_k)$.

By the GCLF-axiom, the first two inequalities imply that $v \leq_M m^{(2)}v$. Now, taking into account also the last one, we get: $v \leq_M m^{(2)}v \leq_M m^{(2)}m^{(3)}m^{(4)}\Psi(p_k)$. Multiplying both sides of the inequality by $m^{(1)}$ we obtain: $l \vee m^{(1)} = m^{(1)}v \leq_M m^{(1)}m^{(2)}m^{(3)}m^{(4)}\Psi(p_k) = m$. Since $l \leq_M l \vee m^{(1)}$ we get $l \leq_M m$. This completes the inductive step. Now the result is immediate. \square

Now, it is rather standard to verify the correctness of the algorithm outlined in the beginning of the section. The following lemma is technical:

Lemma 7. *Throughout the construction, for any $k = 0, \dots, 2|Q| - 1$ and any $p \in Q$ it holds that there is a set $S_k(p) \subseteq M$ with the following two properties:*

1. $\mu_k(p) \in \inf S_k(p)$,
2. $\mathcal{L}^{\leq k}(p) \subseteq S_k(p) \subseteq \mathcal{L}(p)$.

Corollary 1. *For all $p \in Q$ it holds that $\mu_{2|Q|-1}(p) \in \inf \mathcal{L}(p)$.*

Proof. Let $n = 2|Q| - 1$. By the previous lemma we have that $\mathcal{L}^{\leq n}(p) \subseteq S_n(p) \subseteq \mathcal{L}(p)$. Furthermore by Lemma 6 we know that $\inf \mathcal{L}(p) = \inf \mathcal{L}^{\leq n}(p)$, therefore $\inf S_n(p) = \inf \mathcal{L}(p)$ and hence $\mu_n(p) \in \inf \mathcal{L}(p)$. \square

The following lemma is now straightforward. Similar results for the deterministic case and free monoids or tropical monoid can be found in [8, 11].

Lemma 8. *Let $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, I, F, \Delta, \iota, \Psi \rangle$ be a trimmed one-letter transducer and \mathcal{M} be an mge-monoid. Assume that $\mu(p) \in \inf \mathcal{R}_{\mathcal{T}}(p)$ for all $p \in Q$, then:*

1. *for any transition $t = \langle p, \langle a, m \rangle, p' \rangle \in \Delta$ it holds $\mu(p) \leq_M m \circ \mu(p')$. In particular, $m(t) = \frac{m \circ \mu(p')}{\mu(p)}$ is well-defined.*
2. *for any $f \in F$, $\mu(f) \leq_M \Psi(f)$*
3. *the transducer $\mathcal{T}_c = \langle \Sigma, \mathcal{M}, Q, I, F, \Delta_c, \iota_c, \Psi_c \rangle$, where $\iota_c(s) = \iota(s) \circ \mu(s)$, $\Psi_c(f) = \frac{\Psi(f)}{\mu(f)}$, and $\Delta_c = \{\langle p, \langle a, m(t) \rangle, p' \rangle \mid t = \langle p, \langle a, m \rangle, p' \rangle \in \Delta\}$ has the properties:*
 - (a) $\mathcal{L}(\mathcal{T}) = \mathcal{L}(\mathcal{T}_c)$.
 - (b) *for all $p \in Q$: $e \in \inf \mathcal{R}_{\mathcal{T}_c}(p)$.*

Now we can complete the proof of Theorem 1.

Proof. First, assume that \mathcal{T} is trimmed. Then we consider the projection of \mathcal{T} w.r.t. M and compute $\mu(p) \in \inf \mathcal{R}_{\mathcal{T}}(p)$. Now, the result follows by Lemma 8. In the general case, we first construct the onward \mathcal{T}_c for the trimmed part of \mathcal{T} and afterwards we add the transitions that do not lie on a successful path. This preserves the correctness, since those transitions are irrelevant for the language. Furthermore, $\text{low}(\emptyset) = M$ and hence $e \in \inf \emptyset$. \square

5 Minimisation of (Sub)sequential Transducers

In this section, again, we consider an mge monoid \mathcal{M} with LSL- and GCLF-axioms. We set out to develop a minimisation construction for subsequential transducers. Even though we follow the well-known technique, we have to take into account invertible elements. This is why we give the following syntactic definition:

Definition 10. Let $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$ be a complete subsequential transducer. We define the syntactic relation $\sim_{\mathcal{T}} \subseteq Q \times Q$ as:

$$q_1 \sim_{\mathcal{T}} q_2 \iff \exists u \in \mathcal{M} (u \text{ is invertible and } \mathcal{O}_{\mathcal{T}}^{(q_1)} = u \mathcal{O}_{\mathcal{T}}^{(q_2)}).$$

We shall write $q_1 \sim_{\mathcal{T}}^u q_2$ to stress that $\mathcal{O}_{\mathcal{T}}^{(q_1)} = u \mathcal{O}_{\mathcal{T}}^{(q_2)}$.

We start with the following technical lemma:

Lemma 9. Let $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$ be a complete subsequential transducer. Then $\sim_{\mathcal{T}}$ is an equivalence relation. Furthermore, if \mathcal{T} is onward, then $p_1 \sim_{\mathcal{T}}^u p_2$ implies $\delta(p_1, a) \sim_{\mathcal{T}} \delta(p_2, a)$ for any character $a \in \Sigma$. Moreover, if $\delta(p_2, a) \sim_{\mathcal{T}}^v \delta(p_1, a)$ then $\lambda(p_1, a) = u \lambda(p_2, a) v$.

With this result it is standard to prove that the following construction yields a minimal subsequential transducer.

Lemma 10. Let $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$ be an onward complete subsequential transducer such that each state $q \in Q$ is accessible. Let $n = \text{ind}(\sim_{\mathcal{T}})$ and $Q_M = \{q_1 = s, q_2, \dots, q_n\}$ be a set of pairwise non-equivalent states in Q . Let $F_M = Q_M \cap F$ and $\Psi_M = \Psi \upharpoonright Q_M$ and δ_M and λ_M be defined as:

$$\begin{aligned} \delta_M(q_i, a) &= q_j \iff \delta(q_i, a) \sim_{\mathcal{T}} q_j \\ \lambda_M(q_i, a) &= \lambda(q_i, a)v \text{ if } q_j \sim_{\mathcal{T}}^v \delta(q_i, a) \text{ for some } j \text{ and } v \in \mathcal{M}. \end{aligned}$$

Then $\mathcal{T}_M = \langle \Sigma, \mathcal{M}, Q_M, s, F_M, \delta_M, \lambda_M, \iota, \Psi_M \rangle$ is well-defined and $\mathcal{O}_{\mathcal{T}_M} = \mathcal{O}_{\mathcal{T}}$. Furthermore, \mathcal{T}_M is minimal, i.e. any complete (sub)sequential transducer \mathcal{T}' with $\mathcal{O}_{\mathcal{T}'} = \mathcal{O}_{\mathcal{T}}$ has at least $|Q_M|$ states.

In view of Lemma 10 the question arises whether and how to compute $\sim_{\mathcal{T}}$ for onward subsequential transducers. We generalise the classical minimisation algorithm, [7]. We assume that $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$ is onward.

The problem with applying directly the standard minimisation algorithm for subsequential transducers arises from the necessity to account for the witnesses involved in the definition of the equivalence $\sim_{\mathcal{T}}$. To surmount this difficulty we start with the following simple observation:

Lemma 11. If $p_1 \sim_T^u p_2$ and $p_1 \sim_T^v p_2$ then $\text{Dom}(\mathcal{T}_{p_1}) = \text{Dom}(\mathcal{T}_{p_2}) = \emptyset$, or $u = v$.

Proof. If $\gamma \in \text{Dom}(\mathcal{T}_{p_1})$, then by $p_1 \sim_T^u p_2$ it follows that $\mathcal{O}_T^{(p_1)}(\gamma) = u\mathcal{O}_T^{(p_2)}(\gamma)$ and by $p_1 \sim_T^v p_2$, $\mathcal{O}_T^{(p_1)}(\gamma) = v\mathcal{O}_T^{(p_2)}(\gamma)$. Hence by the RC-axiom $u = v$. \square

Now, the idea is the following. We are going to introduce a linear order on the words in Σ^* and for each co-accessible state $p \in Q$ we are going to compute $\langle \alpha_{\min}(p), m_{\min}(p) \rangle \in \mathcal{L}(p)$ such that $\alpha_{\min}(p)$ is the least element in the domain of p . Using these pairs we are going to compute an initial approximation for \sim_T that we will subsequently refine as in the standard minimisation algorithm.

Definition 11. We define the relation $\preceq_{deg-lex} \subseteq \Sigma^* \times \Sigma^*$ as:

$$\alpha \preceq_{deg-lex} \beta \iff |\alpha| < |\beta| \text{ or } (|\alpha| = |\beta| \text{ and } \alpha \preceq_{lex} \beta)$$

where \preceq_{lex} is some fixed lexicographic ordering on Σ^* .

Definition 12. For a subsequential transducer $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$ we define the mapping: $\mu : Q \rightarrow (\Sigma^* \times \mathcal{M}) \cup \{\perp\}$ as follows:

$$\mu(p) = \arg \min_{\preceq_{deg-lex}} \{\alpha \mid \langle \alpha, m \rangle \in \mathcal{L}(p)\}.$$

Lemma 12. Given a subsequential transducer $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$ over an mge monoid \mathcal{M} we can construct the mapping μ .

Proof. We can adopt a standard BFS procedure to yield the values $\mu(p)$:

1. set $Q_0 = F$, $\mu(p) = \langle \varepsilon, \Psi(p) \rangle$ for each $p \in F$. Let $k = 0$ and $Q_{-1} = \emptyset$.
2. while $Q_k \neq Q_{k-1}$ do:
 - (a) set $Q_{k+1} = Q_k$, $\mu_{k+1} = \mu_k$.
 - (b) for each $q \in Q_k \setminus Q_{k-1}$ and each transition $\delta(p, a) = q$ add p to Q_{k+1} .
 - (c) for $p \in Q_{k+1} \setminus Q_k$ let $a \in \Sigma$ be the lex. least s.t. $\delta(p, a) \in Q_k$. Set:
 $\mu_{k+1}(p) = \langle a, \lambda(p, a) \rangle \circ \mu_k(\delta(p, a))$.
 - (d) set $k = k + 1$.
3. complete μ_k with \perp to μ .

The correctness of the algorithm follows by an inductive argument. \square

Next, we proceed to show how to compute the relation \sim_T . First we define the relation $\sim_0 \subseteq Q \times Q$ as:

$$\begin{aligned} p_1 \sim_0 p_2 \iff & \mu(p_1) = \mu(p_2) = \perp \text{ or} \\ & \mu_1(p_1) = \mu_1(p_2) \text{ and } \mu_2(p_1) \ominus \mu_2(p_2) \neq \perp. \end{aligned}$$

If $p_1 \sim_0 p_2$ we write $p_1 \sim_0^\perp p_2$ if $\mu(p_1) = \perp$, or $p_1 \sim_0^u p_2$ where $u = \mu_2(p_1) \ominus \mu_2(p_2)$ to indicate the witness for $p_1 \sim_0 p_2$.

Remark 8. It should be clear that \sim_0 is an equivalence relation.

Assume that $\sim_n \subseteq Q \times Q$ is defined. We refine \sim_n to \sim_{n+1} as follows. For states $p_1, p_2 \in Q$, $p_1 \sim_{n+1} p_2$ if and only if the following conditions hold:

1. $p_1 \sim_n p_2$ and $p_1 \sim_0^u p_2$ for some $u \in \mathcal{M} \cup \{\perp\}$,
2. for each $a \in \Sigma$, $\delta(p_1, a) \sim_n \delta(p_2, a)$, $\delta(p_2, a) \sim_0^v \delta(p_1, a)$, and

$$v \neq \perp \Rightarrow \lambda(p_1, a) = u\lambda(p_2, a)v.$$

The following standard lemma shows that computing the relations \sim_n suffices to compute $\sim_{\mathcal{T}}$:

Lemma 13. *For all $n \in \mathbb{N}$ it holds: (i) $\sim_{n+1} \subseteq \sim_n$; (ii) $\sim_{\mathcal{T}} \subseteq \sim_n$; (iii) $\bigcap \sim_n \subseteq \sim_{\mathcal{T}}$; (iv) if $\sim_n = \sim_{n+1}$ then $\sim_n = \sim_{\mathcal{T}}$; (v) $\sim_{|Q|} = \sim_{\mathcal{T}}$.*

From the discussion above we obtain the following result:

Theorem 2. *Let \mathcal{M} be an mge monoid with LSL- and GCLF-axioms. Then there is a construction that converts any (complete) (sub)sequential transducer: $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, s, F, \delta, \lambda, \iota, \Psi \rangle$ into an equivalent minimal (sub)sequential transducer.* \square

6 Example for the Necessity of the GCLF-axiom

Whereas the LC and RC-axioms are natural algebraic axioms, and RMGE- and LSL- are, in a sense, also natural semi-lattice properties, a question arises what the influence of the GCLF-axiom is. Clearly, from the proof of Lemma 6, it allows us to eliminate cycles in the transducer or, equivalently, eliminate Kleene-stars in the definition of regular languages over the monoid \mathcal{M} . But, can we drop it and still have infimums of regular sets? The answer to this question is negative:

Theorem 3. *There is a monoid \mathcal{M} such that:*

1. \mathcal{M} satisfies the RMGE-, LC-, RC-, and LSL-axioms,
2. \mathcal{M} violates the GCLF-axiom,
3. \mathcal{M} admits a (non-empty) regular language with an empty infimum set.

We construct such an example as follows. We consider the alphabet $\Sigma = \{a, b, c\}$ with three distinct elements. Then, we factorise the free monoid Σ^* w.r.t. the equality $abc = b$. In terms of grammars we can describe this as follows. Let G be a grammar with a single production $abc \rightarrow b$. For a word $w \in \Sigma^*$ we denote with $[w]$ the deg-lex minimal word in Σ^* such that: $w \Rightarrow_G^* [w]$. We set: $M = \{[w] \mid w \in \Sigma^*\}$. Next, we introduce the operation $\circ : M \times M \rightarrow M$ in a standard way:

$$[u] \circ [v] = [[u][v]],$$

where $[u][v]$ refers to the usual concatenation of words in Σ^* . It is standard combinatorics on words to check that:

Lemma 14. *The structure $\mathcal{M} = \langle M, \circ, [\varepsilon] \rangle$ is a monoid and further it supports left and right cancellation and satisfies RMGE and LSL-axioms.* \square

Remark 9. Note that $[abc] = [b] = b$, $[a] = a$, $[bc] = bc$ and $[ab] = ab$. Therefore, we have $[b] \leq_M [bc]$ and $[b] \leq_M [a] \circ [bc]$. On the other hand $[b] \not\leq_M [a] \circ [b] = ab$. This shows that \mathcal{M} violates the GCLF-axiom.

Finally, it holds that:

Lemma 15. *In $\mathcal{M} = \langle M, \circ, [\varepsilon] \rangle$ the language a^*b has an empty infimum set.*

Proof (Sketch). It is easy to see that $[\alpha] \in \text{low}(a^*b)$ implies $[\alpha] \in a^* \cup a^*b$. Next, since $a^{n+1}b \leq_M [a^{n+1}b] \circ [c] = a^n b$ and $a^n b \not\leq_M a^{n+1}b$, we get that $\text{low}(a^*b) = a^*$. But since $a^{n+1} \not\leq_M a^n$ we get that no element of a^* can be in $\inf(a^*b)$. \square

7 Conclusion

In this paper we defined a general class of monoids for which the canonisation and minimisation of subsequential transducers are tractable. We showed constructive proofs of our results, which, under reasonable assumptions, can be turned into efficient algorithms. All but one of the axioms that describe the considered class of monoids correspond to natural algebraic properties from the theory of semi-groups and lattices, respectively. For the last one we have shown that it holds if the canonisation should be always possible. It is interesting if these results may open new application areas for the learning algorithms developed in [6, 8].

References

1. Breslauer, D.: The suffix tree of a tree and minimizing sequential transducers. *Theor. Comput. Sci.* **191**(1–2), 131–144 (1998)
2. Choffrut, C.: Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theor. Comput. Sci.* **5**, 325–338 (1977)
3. Daviaud, L., Reynier, P.A., Talbot, J.M.: A generalised twinning property for minimisation of cost register automata. In: Proceedings - Symposium on Logic in Computer Science, pp. 857–866 (2016)
4. Eilenberg, S.: Automata, Languages and Machines. Academic Press, New York (1974)
5. Gerdjikov, S., Mihov, S.: Over which monoids is the transducer determinization procedure applicable? In: Drewes, F., Martín-Vide, C., Truthe, B. (eds.) LATA 2017. LNCS, vol. 10168, pp. 380–392. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53733-7_28
6. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, Cambridge (2010)
7. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (2001)
8. Jardine, A., Chandlee, J., Eyraud, R., Heinz, J.: Very efficient learning of structured classes of subsequential functions from positive data. In: Proceedings of the 12th International Conference on Grammatical Inference, vol. 34, pp. 94–108 (2014)
9. Mohri, M.: On some applications of finite-state automata theory to natural language processing. *J. Nat. Lang. Eng.* **2**, 1–20 (1996)

10. Mohri, M.: Finite-state transducers in language and speech processing. *Comput. Linguist.* **23**(2), 269–311 (1997)
11. Mohri, M.: Minimization algorithms for sequential transducers. *Theor. Comput. Sci.* **234**, 177–201 (2000)
12. Roche, E., Schabes, Y.: Introduction. In: Roche, E., Schabes, Y. (eds.) *Finite-State Language Processing*, pp. 1–66. MIT Press, Cambridge (1997)
13. Sakarovitch, J.: *Elements of Automata Theory*. Cambridge University Press, Cambridge (2009)



Deciding Regular Intersection Emptiness of Complete Problems for PSPACE and the Polynomial Hierarchy

Demen Güler^(✉), Andreas Krebs, Klaus-Jörn Lange, and Petra Wolf

Wilhelm-Schickard-Institut, Universität Tübingen,
Sand 13, 72076 Tübingen, Germany

{gueler,krebs,lange,wolfp}@informatik.uni-tuebingen.de

Abstract. For a regular set R of quantified Boolean formulae we decide whether R contains a true formula. We conclude that there is a PSPACE-complete problem for which emptiness of intersection with a regular set is decidable. Furthermore, by restricting depth and order of quantification we obtain complete problems for each level of the polynomial hierarchy with this decidability as well.

Keywords: Automata and logic · Emptiness of regular intersection
Quantified Boolean formula · PSPACE · Polynomial hierarchy

1 Introduction and Motivation

The original motivation of this work is to distinguish *families of formal languages* like the context-free or regular ones with iteration or pumping lemmata and various decidabilities from *complexity classes* where no such combinatorial arguments are available. In particular, the existence of families of formal languages being densely complete in the classes NP, SAC¹ and NSPACE($\log n$) [5, 6] drives this line of research by highlighting the disparities between formal languages and complexity classes. The distinction between these two kinds of language classes currently relies only on examples. Out of the lengthy list of differences (see [7]) we analyzed the decidability of the *emptiness of intersection with regular languages* as a candidate for a precise criterion. In contrast to complexity classes, families of *formal languages* (as Reg, CFL and their subclasses) have a decidable emptiness problem. Since they are closed under intersection with regular sets, the following property of a formal language is decidable.

Definition 1. Let $\text{int}_{\text{Reg}}(L)$ be the problem of deciding, given a finite automaton, whether its language has a non-empty intersection with L .

Since all known families of *formal languages* (up to the OI-hierarchy as one of the largest known family of formal languages [3]) are contained in NP we void this criterion by identifying problems (most probably) outside of NP for which int_{Reg} is decidable. We exhibit languages L of true quantified Boolean

formulae (again, likely) outside of NP where $\text{int}_{\text{Reg}}(L)$ is decidable. This implies that int_{Reg} is unsuitable as a characterization of formality. Van Leeuwen [10] showed that the satisfiability problem SAT in an appropriate *variable-free* coding is an ET0L (and hence indexed) language, which implies the decidability of $\text{int}_{\text{Reg}}(\text{SAT}_{\text{var-free}})$ [1].

This might be contrasted by the NP-complete *machine language* $L_{\text{NP}} := \{\langle M \rangle, x, a^n \mid M \text{ is NTM accepting } x \text{ in } n \text{ steps}\}$. Here, $\text{int}_{\text{Reg}}(L_{\text{NP}})$ is undecidable since for an arbitrary Turing machine M_0 with $L(M_0) \subseteq \Sigma^*$ the language L_{NP} intersected with the regular set $\{\langle M_0 \rangle, x, a^n \mid x \in \Sigma^*, n \geq 0\}$ is non-empty if and only if $L(M_0)$ is non-empty, which is undecidable.

In this article we consider the complexity classes PSPACE and the levels of the polynomial hierarchy (PH). *Machine versions* of languages complete for these classes would be defined by using polynomially time bounded Turing machines of unbounded and respectively bounded alternation depth. In both cases int_{Reg} would be undecidable. As SAT is the canonical NP-complete problem, languages of (true) quantified Boolean formulae (TQBF) are canonical complete languages for PSPACE and PH. Arbitrary true quantified Boolean formulae form a PSPACE-complete language, where constraining quantification depth and order yields languages for the classes of the polynomial hierarchy. We show for the Σ_k^P -, Π_k^P - and respectively PSPACE-complete languages L_{Σ_k} , L_{Π_k} and L_{TQBF} that $\text{int}_{\text{Reg}}(L_{\Sigma_k})$, $\text{int}_{\text{Reg}}(L_{\Pi_k})$ and $\text{int}_{\text{Reg}}(L_{\text{TQBF}})$ are decidable. In the case of languages where int_{Reg} is decidable the question of coding the input (e.g. a Boolean formula) may be of importance. We adopt an encoding for QBFs that only differentiates from the commonly used notation by omitting explicit quantifiers. Instead, literals carry information about their quantification depth as a unary string.

A converse viewpoint of this problem is to consider a regular set of encoded quantified Boolean formulae and to decide whether at least one does evaluate to true. For finite sets this problem is, from a decidability perspective, trivial. On the contrary, the question whether an arbitrary (not necessarily regular) infinite set contains a true quantified Boolean formula is not per se decidable. In our proof we make use of the finiteness of the state set of finite automata which assure us the repetitions of certain subwords in words accepted by the automaton. This way we are able to reduce the set of possibilities to a finite number of candidates when we search for true quantified Boolean formulae in infinite regular sets.

This article is structured as follows: We start with preliminaries and define the complete problems we need. The main results are listed in Sect. 3, where a short overview of the proof ideas are given. The detailed elaboration of the proofs is content of Sects. 4 and 5. Finally, in the last section we discuss our results and list some open problems.

2 Preliminaries

We assume the reader to be acquainted with the basics and standard results of complexity theory as they are presented in any text book of this topic. We

use the common notation for Boolean formulae with 0 and 1 as truth values. For readability reasons we extend regular expressions by operations $E^{\leq n} := (E|E^2|\dots|E^n)$ and $E^{\geq n} := E^n E^*$ for a fixed $n \in \mathbb{N}$ and E a regular expression and write $w \in E$ instead of $w \in L(E)$. In particular we write $E^{\geq 1}$ instead of E^+ because signs are part of the alphabet we use. For $w \in \Gamma^*$ and $\gamma \in \Gamma$ let $\#\gamma(w)$ denote the number of γ s in w .

In the following we will describe our encoding of quantified Boolean formulae. We will avoid the explicit use of quantifiers. Instead, variables will contain their quantification depth. Our convention will be that variables on even levels of quantification are universally quantified, while variables on odd levels are existentially quantified. We allow missing levels of quantification, i.e. the occurring levels of quantification are not necessarily consecutive. As a consequence, the minimal quantification level of a formula decides whether it is a Σ - or Π -formula. E.g. the absence of a first level, together with existence of the second level implies that a Π formula is encoded. In addition we assume the propositional part (the matrix) to be in conjunctive normal form.

Existentially and universally quantifying a propositional formula over $\{0, 1\}$, yields a *quantified Boolean formula*: $\mathcal{Q}_{i_1} \mathbf{x}_{i_1} \mathcal{Q}_{i_2} \mathbf{x}_{i_2} \dots \mathcal{Q}_{i_m} \mathbf{x}_{i_m} \phi(\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \dots, \mathbf{x}_{i_m})$, where $m < n \Rightarrow i_m < i_n$ and $\mathcal{Q}_{i_j} = \exists$ for odd i_j and $\mathcal{Q}_{i_j} = \forall$ for even i_j and \mathbf{x}_{i_j} are finite vectors of Boolean variables and ϕ is a propositional Boolean formula with $\sum_{j=1}^k |\mathbf{x}_{i_j}|$ free variables.

Wrathall [11] showed that true quantified Boolean formulae with k alternating quantifiers and suitable propositional structure yield complete languages for the k -th level of the polynomial hierarchy. The shape of the propositional formula depends on the innermost quantifier. If the innermost quantifier is existential (universal), it is in CNF (DNF). The set of true quantified Boolean formulae with k alternating quantifiers, where $\mathcal{Q}_1 = \exists$ ($\mathcal{Q}_1 = \forall$) is complete for Σ_k^P (Π_k^P). If the propositional formula is kept in CNF regardless of the innermost quantifier, then for each odd (even) k and $\mathcal{Q}_1 = \exists$ ($\mathcal{Q}_1 = \forall$) the true quantified formulas with k alternating quantifiers yield Σ_k^P (Π_k^P)-complete language. If the number of quantifier alternations in formulae is not bounded the set of true quantified Boolean formulae yields the PSPACE-complete set TQBF [9].

Throughout the article we will describe regular sets of quantified Boolean formulae. We will use a natural encoding (similar to Stockmeyer [8]) of quantified Boolean formulae, where literals $\pm b^* a^*$ contain three kinds of information: A sign will indicate whether the variable is negated or not, followed by a factor b^* which indicates the quantification depth and the name or index of the variable used in this literal as suffix in a^* . Odd length quantification levels denote that variables are existentially quantified and analogously even levels that they are universally quantified. Also, we will assume the propositional structure to be in 3-CNF.

Example 2. If two consecutive vectors are both indexed odd/even they will be quantified identically. Let ψ with range over $\mathbf{x}_1 = (x_{1,1})$, $\mathbf{x}_3 = (x_{3,2}, x_{3,4})$, $\mathbf{x}_4 = (x_{4,2})$ and $\mathbf{x}_8 = (x_{8,1}, x_{8,2})$ be

$$\exists \mathbf{x}_1 \exists \mathbf{x}_3 \forall \mathbf{x}_4 \forall \mathbf{x}_8 (x_{1,1} \vee x_{8,1} \vee x_{3,2}) \wedge (x_{3,4} \vee x_{4,2} \vee \neg x_{4,2}) \wedge (\neg x_{3,4} \vee x_{8,2} \vee x_{3,4})$$

Then the encoding of ψ reads

$$\langle +ba \vee +b^8a \vee +b^3aa \rangle \wedge \langle +b^3a^4 \vee +b^4aa \vee -b^4aa \rangle \wedge \langle -b^3a^4 \vee +b^8aa \vee +b^3a^4 \rangle$$

Definition 3. Let $\Gamma = \{a, b, \langle, \rangle, \wedge, \vee, +, -\}$ and \pm be the regular expression of $\{+, -\}$. As the regular set of encoded quantified Boolean formulae in 3-CNF we define $L_{k-QBF} :=$

$$L \left(\langle \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \rangle \left(\wedge \langle \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \rangle \right)^* \right)$$

Furthermore, let $L_{QBF} := \bigcup_{k \geq 1} L_{k-QBF}$ be the set of encoded quantified Boolean formulae without bound of quantifier alternation depth.

Definition 4. Let L_{Σ_k} (L_{Π_k}) $\subseteq L_{k-QBF}$ be the set of all true quantified Boolean formulae in our encoding and 3-CNF, where the first quantifier is existential (universal).

$$\text{Let } L_{k-TQBF} := L_{\Sigma_k} \cup L_{\Pi_k} \text{ and } L_{TQBF} := \bigcup_{k \geq 1} L_{k-TQBF}.$$

Fact 5. For odd k , the language L_{Σ_k} is Σ_k^P -complete and for even k the language L_{Π_k} is Π_k^P -complete [11]. The set $L_{TQBF} \subseteq L_{QBF}$ of encoded true quantified Boolean formulae in 3-CNF is PSPACE-complete [9].

3 Results

In this section we present our two main theorems for the Σ_k^P -complete language L_{Σ_k} , the Π_k^P -complete language L_{Π_k} and the PSPACE-complete language L_{TQBF} , where the formal proofs are contained in Sects. 4 and 5.

Theorem 6. Let R be a regular language. For each $k \in \mathbb{N}$ it is decidable whether $L_{\Sigma_k} \cap R = \emptyset$ and $L_{\Pi_k} \cap R = \emptyset$, i.e. $\text{int}_{\text{Reg}}(L_{\Sigma_k})$ and $\text{int}_{\text{Reg}}(L_{\Pi_k})$ are decidable.

Idea of proof: Let A be a DFA recognizing R . For every pair of states in A we compute the (possibly empty) regular set of end-to-end literals that can be read in-between them. Each such literal set is then assigned a finite set of *representing literals*. We define an automaton $\text{condense}(A)$ based on the finitely many representatives and show that $\text{condense}(A)$ recognizes a true quantified Boolean formula if and only if A accepts one. Finally we show that for each $k \in \mathbb{N}$ the emptiness of $\text{condense}(A) \cap L_{\Sigma_k}$ and $\text{condense}(A) \cap L_{\Pi_k}$ is decidable, which in total proves Theorem 6.

Theorem 7. Let R be a regular language. It is decidable whether $L_{TQBF} \cap R = \emptyset$, i.e. $\text{int}_{\text{Reg}}(L_{TQBF})$ is decidable.

Idea of proof: Let R be given as a DFA A . Formulae recognized by A can be unbounded in their quantification depth. We construct a new automaton $\text{restrict}(A)$ which only accepts formulae of quantification depth up to d , where d is only dependent on the size of A . We show that A accepts a *true* quantified Boolean formula if and only if $\text{restrict}(A)$ accepts one. Following Theorem 6 it is decidable whether $L(\text{restrict}(A))$ contains a true quantified Boolean formula with at most d alternating quantifiers and thus $L_{\text{TQBF}} \cap R = \emptyset$ is decidable, too.

4 Proof of Theorem 1

Let $R \subseteq L_{k-\text{QBF}}$ be a regular language. If R is finite, checking whether R contains at least one true quantified Boolean formula can be achieved by decoding and evaluating every single word in R .

For infinite R this procedure is obviously not possible. Instead, we will show that we only have to test finitely many quantified Boolean formulae in R for each (infinite) regular language to decide the intersection emptiness with L_{Σ_k}/L_{Π_k} .

The idea is to extract from R a finite subset $\{w_1, \dots, w_n\} \subseteq R$ such that $R \cap L_{\Sigma_k} \neq \emptyset$ if and only if $w_i \in L_{\Sigma_k}$ for some i . In order to do so, we look in a word $x \wedge \langle l_1 \vee l_2 \vee l_3 \rangle \wedge y \in R$ for literals l_i which are existentially quantified and make the following case distinction. If the finite automaton A accepting R has a loop while reading l_i we can single out a uniquely referenced variable for l_i . Thus, l_i can be existentially satisfied without effecting other literals/clauses. Otherwise, if A has no loop while reading l_i leading from some state q to some state q' we conclude that there can only exist finitely many different literals l' leading from q to q' . All of these are put (after some massage) in a set $\text{rep}(A_{q,q'})$ for joint treatment of all $\text{rep}(A_{q,q'})$. The case of universally quantified literals needs more care. For every pair (q, q') of states in A we compute a set $\Lambda_{q,q'}$ of literals leading from q to q' . From the powerset of all Λ sets we identify combinations of Λ sets, sharing a common variable, which is chosen as a representative. In a manner this procedure minimizes the number of universally quantified variables, which makes the formula overall easier to be true.

4.1 Construction of the Condensed Automaton

Definition 8. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{k-\text{QBF}}$. For every pair of states $q, q' \in Q$, $s \in \{+, -\}$ and $1 \leq d \leq k$ we define

$$\Lambda_{q,q'}^{d,s} := \{w \in ((|\varepsilon\rangle sb^da^{\geq 1}(\vee\rangle)) \mid \delta^*(q, w) = q'\},$$

the literal transition set from q to q' with sign s and quantifier depth d . Furthermore, let

$$\Lambda_{q,q'} := \bigcup_{s \in \{+, -\}, 1 \leq d \leq k} \Lambda_{q,q'}^{d,s}$$

be the union of all literal transition sets from q to q' . For easier readability, we will sometimes refer to $\Lambda_{q,q'}^{d,s}$ as Λ if d, s and q, q' are understood.

Each Λ is recognized by a *sub-automaton* of A and therefore is a regular set.

Definition 9. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton with $L(A) \subseteq L_{k\text{-QBF}}$. Let $\Upsilon^d := \left\{ \Lambda_{q,q'}^{d,s} \mid q, q' \in Q, s \in \{+, -\} \right\}$ be the set containing all universally quantified literal transition sets with quantifier depth d , for all even d with $1 \leq d \leq k$.

Definition 10. Let $\text{trunc}: \Gamma = \{a, b, \langle, \rangle, +, -, \vee, \wedge\}^* \rightarrow \{a, b\}^*$ be the homomorphism with

$$\text{trunc}(\gamma) := \begin{cases} \gamma & \text{if } \gamma \in \{a, b\}, \\ \varepsilon & \text{otherwise.} \end{cases}$$

Define the operation *extend* such that $\text{extend}(w, \Lambda) := \text{trunc}^{-1}(w) \cap \Lambda$ for $w \in \text{trunc}(\Lambda)$ and language Λ .

Intuitively, function *trunc* returns the variable referenced by a literal forgetting its sign and its position in a clause, while *extend* provides us with all possible occurrences of a variable as a literal leading from state q to state q' .

Definition 11. Let $P^d := \{p \in \mathcal{P}(\Upsilon^d) \mid \cap_{\Lambda \in p} \text{trunc}(\Lambda) \neq \emptyset\}$ for all even d with $1 \leq d \leq k$ be the subset of the powerset of all literal transition sets with quantifier depth d , which only contains sets of languages with a common variable.

In the following we construct for each Λ a finite set $\text{rep}(\Lambda) \subseteq \Lambda$ of representatives, which will carry the essential information whether the language contains a true quantified Boolean formula.

Definition 12. For a language $L \subseteq \Gamma^*$ let $\min_{\text{lex}}(L)$ denote the lexicographically minimal element of L .

Since an existentially quantified variable which only occurs once in a quantified Boolean formula can always be satisfied, we try to *separate* all existentially quantified variables when assigning representatives to each literal transition set. Conversely, having many different universally quantified variables makes a quantified Boolean formula harder to be true. The elements in P^d state which literal transition sets contain universally quantified literals, which can reference the same variable.

Definition 13. For every $\Lambda_{q,q'}^{d,s}$ define a finite set of representatives $\text{rep}(\Lambda_{q,q'}^{d,s})$ with the following case distinction:

1. If d is even (universally quantified) use Algorithm 1 to compute the representatives of $\Lambda_{q,q'}^{d,s}$.
2. If d is odd (existentially quantified) and $|\Lambda_{q,q'}^{d,s}| < \infty$, then $\text{rep}(\Lambda_{q,q'}^{d,s}) = \Lambda_{q,q'}^{d,s}$.
3. If d is odd (existentially quantified) and $|\Lambda_{q,q'}^{d,s}| = \infty$, then $\text{rep}(\Lambda_{q,q'}^{d,s}) = \text{extend}(b^d a^l, \Lambda_{q,q'}^{d,s})$, such that the truncated $b^d a^l$ is in no other set of representatives for any $\Lambda_{p,p'}^{d',s'}$, with $p, p' \in Q$, quantifier depth d' and sign s' .

```

forall  $\Lambda \in \Upsilon^d$  do
|    $rep(\Lambda) \leftarrow \emptyset$ 
end
forall  $p \in P^d$  do
|    $label(p) \leftarrow \min_{lex} \left( \bigcap_{\Lambda \in p} trunc(\Lambda) \right)$ 
end
forall  $\Lambda \in p$  do
|    $rep(\Lambda) \leftarrow rep(\Lambda) \cup extend(label(p), \Lambda)$ 
end

```

Algorithm 1. Computation of $rep(\Lambda)$ for even d . For every combination of literal transition sets sharing common variables, unique representatives are chosen and assigned to all respective literal transition sets.

Remark 14. The lexicographic minimal element of each $p \in P^d$ is chosen to make the algorithm deterministic. Any other element of $\bigcap_{\Lambda \in p} trunc(\Lambda)$ could be picked as possible label.

Lemma 15. For all $q, q' \in Q$, quantifier depth $d \leq k$ and $s \in \{+, -\}$ the set $rep(\Lambda_{q,q'}^{d,s})$ is finite.

Proof. For odd d the claim is easily verified. So, assume d to be even. The set Υ^d is finite since d is bounded by k , Q is finite and s can only assume two values. Hence the powerset $\mathcal{P}(\Upsilon^d)$ is finite and therefore P^d is finite, too. Each $p \in P^d$ has exactly one label. So for finitely many p , finitely many elements are added to $rep(\Lambda)$ for any Λ . Hence $rep(\Lambda)$ is finite for any $\Lambda \in \Upsilon^d$. \square

Definition 16. For $q, q' \in Q$ let

$$rep(\Lambda_{q,q'}) := \bigcup_{s \in \{+, -\}, d \leq k} rep(\Lambda_{q,q'}^{d,s})$$

be the set of representatives from q to q' of arbitrary quantification and sign s .

Definition 17. Based on literal transition sets (see Definition 8), we define for every $q, q' \in Q$ clause transition sets

$$C_{q,q'} := \bigcup_{q_1, q_2 \in Q, \Lambda_{q,q_1} \neq \emptyset, \Lambda_{q_1, q_2} \neq \emptyset, \Lambda_{q_2, q'} \neq \emptyset} rep(\Lambda_{q,q_1}) \cdot rep(\Lambda_{q_1, q_2}) \cdot rep(\Lambda_{q_2, q'}),$$

where we require that all words in Λ_{q,q_1} start with \langle , the words in Λ_{q_1, q_2} contain neither \langle nor \rangle and the words in $\Lambda_{q_2, q'}$ end with \rangle .

Lemma 18. For every $q, q' \in Q$ the clause transition set $C_{q,q'}$ is finite.

Proof. For every $q, q' \in Q$ the set of representatives $rep(\Lambda_{q,q'})$ is finite, and hence finite concatenation yields again a finite set. \square

Definition 19. Let $R \subseteq L_{k\text{-QBF}}$ be a regular language and $A = (Q, \Gamma, \delta, q_0, F)$ be a DFA with $L(A) = R$. Define the condensed automaton $\text{condense}(A) = (Q, \Gamma', \delta', q_0, F)$ where

$$\begin{aligned} \Gamma' &= \bigcup_{q, q' \in Q} C_{q, q'} \cup \{\wedge\} \\ \text{and } \forall q, q' \in Q : (q, w, q') \in \delta' &\text{ if } w \in C_{q, q'} \\ \forall q, q' \in Q : (q, \wedge, q') \in \delta' &\text{ if } (q, \wedge, q') \in \delta . \end{aligned}$$

The automaton $\text{condense}(A)$ accepts a subset of $L(A)$ such that the words in $L(\text{condense}(A))$ are built over a finite set of clauses.

The next section shows that $L(A) \cap L_{\Sigma_k} \neq \emptyset \iff L(\text{condense}(A)) \cap L_{\Sigma_k} \neq \emptyset$. Thus, we can decide the emptiness of the intersection by inspecting the finitely many elements which are accepted by loop-free paths in $\text{condense}(A)$ one-by-one.

4.2 Condensation Preserves Regular Intersection (Non-)Emptiness

Lemma 20. Let $R \subseteq L_{k\text{-QBF}}$ be a regular language and A be a DFA with $L(A) = R$. If $L(\text{condense}(A)) \cap L_{k\text{-TQBF}} \neq \emptyset$ then also $R \cap L_{k\text{-TQBF}} \neq \emptyset$.

Proof. The condensed automaton $\text{condense}(A)$ recognizes a subset of R . In particular, every recognized true quantified Boolean formula w , is also in R . \square

Lemma 21. Let $R \subseteq L_{k\text{-QBF}}$ be a regular language and A be a DFA with $L(A) = R$. If $R \cap L_{k\text{-TQBF}} \neq \emptyset$ then also $L(\text{condense}(A)) \cap L_{k\text{-TQBF}} \neq \emptyset$.

Proof (proof idea). Let $w \in R$ evaluate to true. Every literal of w is part of some Λ . Each one can be substituted by a representative in $\text{rep}(\Lambda)$, making a case distinction between existential and universal quantification, yielding a word in $L(\text{condense}(A))$ which is also true. \square

4.3 Deciding $\text{int}_{\text{Reg}}(L_{\Sigma_k})$ and $\text{int}_{\text{Reg}}(L_{\Pi_k})$

Lemma 22. Let A be a DFA with $L(A) \subseteq L_{k\text{-QBF}}$ and $w \in L(\text{condense}(A))$ where w is the labeling of an accepting path p in $\text{condense}(A)$ containing at least one loop. Let w' be w without the factors read in the loops of p . If w is a true quantified Boolean formula, then so is w' .

Proof. The formula w is in conjunctive form $w = c_1 \wedge \dots \wedge c_n$ with some clauses c_i which all evaluate to true. Loops only contain whole clauses and thus w' consists of a subset of the clauses of w . Thus, w' also evaluates to true. \square

This means that words recognized through simple accepting paths in the condensed automaton are the only ones that need consideration when looking for true quantified Boolean formulae. To determine whether L_{Σ_k} or L_{Π_k} contain a word of $L(\text{condense}(A))$ we need to consider the outermost quantified variable. Since clauses containing this variable might occur on a loop we have to consider paths of length $\leq 2|Q|$.

Lemma 23. Let R be a regular language and A be a DFA with $L(A) = R$. It is decidable whether $L(\text{condense}(A)) \cap L_{\Sigma_k} \neq \emptyset$ and $L(\text{condense}(A)) \cap L_{\Pi_k} \neq \emptyset$.

Proof. Let $R \subseteq L_{k\text{-QBF}}$ w.l.o.g. enumerate all words w_1, w_2, \dots, w_n which are recognized through (the finitely many) accepting paths in $\text{condense}(A)$ of length $\leq 2|Q|$. For $i = 1, \dots, n$ test if w_i is a true quantified Boolean formula. Following Lemma 22, if all w_i evaluate to false, no other $w \in L(\text{condense}(A))$ can evaluate to true. Thus, the intersection $L(\text{condense}(A)) \cap L_{\Sigma_k}$ is non-empty iff at least one w_i evaluates to true and the first quantifier in w_i is existential. Analogously, $L(\text{condense}(A)) \cap L_{\Pi_k} \neq \emptyset$ iff at least one w_i evaluates to true and the first quantifier in w_i is universal. \square

In total this yields Theorem 6.

Theorem 1. Let R be a regular language. For each $k \in \mathbb{N}$ it is decidable whether $L_{\Sigma_k} \cap R = \emptyset$ and $L_{\Pi_k} \cap R = \emptyset$, i.e. $\text{int}_{\text{Reg}}(L_{\Sigma_k})$ and $\text{int}_{\text{Reg}}(L_{\Pi_k})$ are decidable.

Proof. Assume w.l.o.g. that $R \subseteq L_{k\text{-QBF}}$. Let A be a DFA recognizing R . Lemma 23 states that $\text{condense}(A) \cap L_{\Sigma_k} \neq \emptyset$ and $\text{condense}(A) \cap L_{\Pi_k} \neq \emptyset$ is decidable. Following Lemmas 20 and 21 $L(\text{condense}(A)) \cap L_{k\text{-TQBF}} \neq \emptyset \iff R \cap L_{k\text{-TQBF}} \neq \emptyset$. Thus both, $L_{\Sigma_k} \cap R = \emptyset$ and $L_{\Pi_k} \cap R = \emptyset$ is decidable. \square

5 Proof of Theorem 2

In this section we prove Theorem 7, stating that $\text{int}_{\text{Reg}}(L_{\text{TQBF}})$ is decidable. That is, one can test whether a regular language encoding quantified Boolean formulae in 3-CNF with unbounded quantifier alternation contains at least one *true* quantified Boolean formula. Observe that the automaton $\text{condense}(A)$ constructed in the last subsection to test for (non)emptiness of $R \cap L_{\Sigma_k}$ is dependent in k . We now give a construction which allows us to use the $\text{condense}(A)$ -automaton despite the presence of unboundedly nested quantifiers.

Fact 24. Let \mathbf{x}_1 and \mathbf{x}_2 be vectors of Boolean variables and ϕ be a propositional Boolean formula with $|\mathbf{x}_1| + |\mathbf{x}_2|$ free variables. Then following implications holds:

1. $\exists \mathbf{x}_1 \forall \mathbf{x}_2 \phi(\mathbf{x}_1, \mathbf{x}_2) \Rightarrow \forall \mathbf{x}_2 \exists \mathbf{x}_1 \phi(\mathbf{x}_1, \mathbf{x}_2)$
2. $\forall \mathbf{x}_1 \phi(\mathbf{x}_1) \Rightarrow \exists \mathbf{x}_1 \phi(\mathbf{x}_1)$
3. Let \mathcal{Q} be fixed as \exists or \forall . Then $\mathcal{Q} \mathbf{x}_1 \mathcal{Q} \mathbf{x}_2 \phi(\mathbf{x}_1, \mathbf{x}_2) \Rightarrow \mathcal{Q} \mathbf{x}_2 \mathcal{Q} \mathbf{x}_1 \phi(\mathbf{x}_1, \mathbf{x}_2)$

Fact 24 intuitively speaking states that we can “pull out” universal quantifiers. Furthermore, we can substitute universal quantifiers by existential ones and permute consecutive vectors quantified by the same type of quantifier while preserving validity.

5.1 Defining the Reduction

For a given DFA A we construct a new automaton $\text{restrict}(A)$ which accepts a true quantified Boolean formula if and only if A does. Depending on the size of A there is constant d such that, formulas in $L(\text{restrict}(A))$ have at most $d + 2$ (and therefore finite) quantifier alternations. Words of $L(\text{restrict}(A))$ have the property that quantifier levels of literals above a threshold d are summed up in a sequence of universal quantifier levels, followed by a non-overlapping sequence of existentially quantified levels.

Definition 25. For a DFA $A = (Q, \Gamma, \delta, q_0, F)$ and $s \in \{+, -\}$ let

$$G_{q,q'}^s := \{n \in \mathbb{N} \mid \exists q_1, q_2 \in Q : \delta(q_1, s) = q \wedge \delta^*(q, b^n) = q' \wedge \delta(q', a) = q_2\}$$

be the set of quantification depths we can use for s -signed literals starting in q and passing q' on the way.

Lemma 26. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a DFA. For all $q, q' \in Q$ and $s \in \{+, -\}$ the following statement holds: If there exists $n \in G_{q,q'}^s$ with $n \geq |Q|$ and $n \equiv 1 \pmod{2}$, then $|G_{q,q'}^s \cap \{i \mid i \geq |Q| \text{ and } i \equiv 1 \pmod{2}\}| = \infty$.

If the same even quantification level n with $n \geq 2|Q|^{|Q|}$ can be read in different locations of an DFA A , a smaller, also even, quantification level n' with $n' \leq 2|Q|^{|Q|}$ can be read in the same locations.

Lemma 27. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a DFA and $Z \subseteq Q \times Q$. For all $(q, q') \in Z$ and $s \in \{+, -\}$ the following statement holds: If $\exists n \in \bigcap_{(q,q') \in Z} G_{q,q'}^s$ with $n \geq 2|Q|^{|Q|}$ and $n \equiv 0 \pmod{2}$, then $\exists n' \in \bigcap_{(q,q') \in Z} G_{q,q'}^s$ with $n' \leq 2|Q|^{|Q|}$ and $n' \equiv 0 \pmod{2}$.

Lemmata 26 and 27 can be proved by using pumping arguments.

Definition 28. For a DFA $A = (Q, \Gamma, \delta, q_0, F)$, let $G_{q,q'}^{\triangle,s} := G_{q,q'}^s \cap \{i \mid i \geq |Q|\}$ be the set of quantifier levels higher then $|Q|$. We will define the level of $G_{q,q'}^{\triangle,s}$ algorithmically. We set `counter` $\leftarrow 2|Q|^{|Q|} + 1$ in the beginning and compute for each $q, q' \in Q$ and $s \in \{+, -\}$ the level of a $G_{q,q'}^{\triangle,s}$ in arbitrary, but fixed order with Algorithm 2.

Definition 29. For a DFA $A = (Q, \Gamma, \delta, q_0, F)$ define $\text{restrict}(A)$ as the automaton $A' = (Q', \Gamma, \delta', q_0, F)$ with δ' being the following modification of δ . For each $p, p' \in Q$ remove each transition of the form $\delta(p, b) = p'$. Instead, we will introduce a path $\delta^*(q, b^n) = q'$ for each $n \in G_{q,q'}^s$ with $n < |Q|$ using $n - 1$ supplementary states we add to Q' . Moreover, for each $G_{q,q'}^{\triangle,s}$ we add additional paths $\delta^*(q, b^n) = q'$ for each $n \in \text{level}(G_{q,q'}^{\triangle,s})$.

In the restricting process we remove all b -transitions to get rid of b -loops. Loop-free b -transitions are reintroduced in such a way, that the emptiness of the intersection with L_{TQBF} is not changed.

```

if  $G_{q,q'}^{\triangle,s} \cap \{i \mid i \equiv 1 \pmod{2}\} \neq \emptyset$  then
|    $\text{level}(G_{q,q'}^{\triangle,s}) \leftarrow \min(G_{q,q'}^{\triangle,s} \cap \{i \mid i \text{ is odd and } i \geq \text{counter}\})$ 
|    $\text{counter} \leftarrow \text{level}(G_{q,q'}^{\triangle,s}) + 2$ 
else
|    $\text{level}(G_{q,q'}^{\triangle,s}) \leftarrow G_{q,q'}^{\triangle,s} \cap \{i \mid i \equiv 0 \pmod{2} \text{ and } |Q| \leq i \leq 2|Q|^{|Q|}\}$ 
end

```

Algorithm 2. Computing the level for $G_{q,q'}^{\triangle,s}$. If possible we pick an existential representative above the threshold $2|Q|^{|Q|}$ and separate it by quantification level. Otherwise, a representative for every universal level below the this threshold is introduced.

5.2 Correctness of the Reduction

Lemma 30. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a deterministic finite automaton. Then, $L(A) \cap L_{TQBF} \neq \emptyset$ if and only if $L(\text{restrict}(A)) \cap L_{TQBF} \neq \emptyset$.

Proof (proof idea). “ \Rightarrow ”: Let $w \in L(A) \cap L_{TQBF}$. We use the pumping property to substitute universal quantifiers by existential ones and shift the existential quantifiers to the end, while we conflate the unbounded universal quantifiers to a finite block. Renaming the variables yields a word in $L(\text{restrict}(A))$.

“ \Leftarrow ”: By construction $L(\text{restrict}(A))$ is a proper subset of $L(A)$. \square

5.3 Deciding $\text{int}_{\text{Reg}}(L_{TQBF})$

Theorem 2. Let R be a regular language. Then $R \cap L_{TQBF} = \emptyset$ is decidable.

Proof. Let A be the finite automaton recognizing R and let k be the length of the longest b -factor along an accepting path of $\text{restrict}(A)$. Following Theorem 6 it is decidable whether $L(\text{restrict}(A)) \cap L_{k-TQBF} = \emptyset$. Lemma 30 states that $L(\text{restrict}(A))$ contains an encoded true quantified Boolean formula if and only if $L(A)$ does. Hence, $L(A) \cap L_{TQBF} = \emptyset$ if and only if $L(\text{restrict}(A)) \cap L_{\Sigma_k} = \emptyset$ and $L(\text{restrict}(A)) \cap L_{\Pi_k} = \emptyset$. Since the intersection is non-empty if and only if $\text{restrict}(A)$ encodes a true quantified formula, the latter is also decidable. \square

6 Discussion

We have shown that each level Σ_k^P , Π_k^P in the polynomial hierarchy, as well as PSPACE contain complete languages L_i such that $\text{int}_{\text{Reg}}(L_i)$ is decidable. While it was known, that for families of formal languages, such as context-free and indexed languages int_{Reg} is decidable, complexity classes, generally, do not have this property. Thus, the decidability of int_{Reg} seemed to be an adequate criterion to characterize *formal* language. However, L_{Σ_k} and L_{TQBF} show the existence of languages (probably) outside of NP with the aforementioned property, voiding regular intersection emptiness as a formality criterion. Further research in

differentiating families of formal languages from complexity classes and further discrediting int_{Reg} as a criterion of formality will be presented in the upcoming article [4].

Since every complexity class has its own complete *machine language*, for which int_{Reg} is undecidable, an interesting question would be how to characterize the subset of those elements of a complexity class which do have a decidable int_{Reg} . If we denote for a complexity class \mathcal{X} by $\mathcal{X}_{\text{int}_{\text{Reg}}}$ the set of all $L \in \mathcal{X}$ with decidable $\text{int}_{\text{Reg}}(L)$, our results imply equivalences like $\text{P}_{\text{int}_{\text{Reg}}} = \text{NP}_{\text{int}_{\text{Reg}}} \Leftrightarrow \text{P} = \text{NP}$.

The problems we decide in this work can be expected to be of high complexity. Since $w \in L \Leftrightarrow L \cap \{w\} \neq \emptyset$ the complexity of the corresponding word problems is a lower bound.

An open question is how much influence the kind of encoding has on the decidability of int_{Reg} . In the encoding we used for quantified Boolean formulae the literals in each clause were given sequentially, similar to Stockmeyer's [8] encoding. While letterwise shuffling strings that encode Turing machine configurations have been used to show undecidability results [2], we have the conjecture that int_{Reg} still is decidable for 3-SAT, where the literals in each clause are given *letterwise shuffled*. This would only yield the result for $\text{NP} = \Sigma_1^P$, but leave the open question whether for the set of true quantified Boolean formulae encoded letterwise shuffled int_{Reg} is decidable.

It would be also interesting to consider regular intersection of complete problems for complexity classes above PSPACE, such as for example EXP or EXPSPACE. Also, one could consider the decidability of non-regular sets (e.g. visibly pushdown languages, or one counter languages) of quantified Boolean formulae regarding their containment of true formulae.

Acknowledgments. We thank Benjamin Gras for the fruitful discussions during the TüFTLeR seminar. Also, we give our thanks to Michaël Cadilhac, Silke Czarnetzki and Michael Ludwig for proof-reading.

References

1. Aho, A.V.: Indexed grammars—an extension of context-free grammars. *J. ACM (JACM)* **15**(4), 647–671 (1968)
2. Calbrix, H., Knapik, T.: A string-rewriting characterization of Muller and Schupp's context-free graphs. In: Arvind, V., Ramanujam, S. (eds.) FSTTCS 1998. LNCS, vol. 1530, pp. 331–342. Springer, Heidelberg (1998). https://doi.org/10.1007/978-3-540-49382-2_31
3. Damm, W.: The IO- and OI-hierarchies. *Theor. Comput. Sci.* **20**(2), 95–207 (1982)
4. Güler, D., Lange, K.J.: What is (Not) a Formal Language (in preparation)
5. Krebs, A., Lange, K.-J.: Dense completeness. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 178–189. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31653-1_17
6. Krebs, A., Lange, K.-J., Ludwig, M.: On distinguishing NC¹ and NL. In: Potapov, I. (ed.) DLT 2015. LNCS, vol. 9168, pp. 340–351. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21500-6_27

7. Lange, K.J.: Complexity and structure in formal language theory. *Fundam. Inform.* **25**(3, 4), 327–352 (1996)
8. Stockmeyer, L.J.: The polynomial-time hierarchy. *Theor. Comput. Sci.* **3**(1), 1–22 (1976)
9. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time (preliminary report). In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, pp. 1–9. ACM (1973)
10. Van Leeuwen, J.: The membership question for ETOL-languages is polynomially complete. *Inf. Process. Lett.* **3**(5), 138–143 (1975)
11. Wrathall, C.: Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.* **3**(1), 23–33 (1976)



Learners Based on Transducers

Sanjay Jain^{1(✉)}, Shao Ning Kuek², Eric Martin³, and Frank Stephan^{1,2}

¹ Department of Computer Science, National University of Singapore,
13 Computing Drive, COM1, Singapore 117417, Republic of Singapore
{sanjay,fstephan}@comp.nus.edu.sg

² Department of Mathematics, National University of Singapore,
10 Lower Kent Ridge Road, S17, Singapore 119076, Republic of Singapore
shaoning@u.nus.edu

³ School of Computer Science and Engineering, The University of New South Wales,
Sydney, NSW 2052, Australia
eric.martin@unsw.edu.au

Abstract. As data come out one by one from an infinite stream, automatic learners maintain some string as long term memory, and update it at every new datum (example) they process. Transduced learners are generalization of automatic learners. Both kind of learners are evaluated with respect to the space they consume for learning. For automatic learners, it is unknown whether at any point, the size of the long term memory can be bounded by the length of the longest datum that has been received so far. Here it is shown that, even when restricting learning to automatic families, there is a hierarchy of classes that can be learnt with memory $O(n^k)$, and all automatic families which are learnable in principle can be learnt by a transduced learner using exponential sized memory.

Keywords: Automata and logic · Inductive inference · Transducers

1 Introduction

Gold [11] introduced the model of learning in the limit from positive data. Subsequent research in inductive inference [1, 3, 6, 15, 19, 22, 23] studied also variations on this model. The basic features of Gold’s model are the following. Let $L \subseteq \Sigma^*$ be a language, where Σ is a finite alphabet. The learner gets as input a *text for L*, that is, a sequence of strings x_0, x_1, \dots that contains all members but no non-member of L . As output, the learner conjectures a sequence of indices e_0, e_1, \dots as its hypotheses on what the input language might be. The hypotheses are taken from some hypothesis space $\{H_e : e \in I\}$, where I is the set of possible indices and every possible learning task equals some H_e . If the sequence

S. Jain and F. Stephan are supported in part by the Singapore Ministry of Education Academic Research Fund grants R146-000-181-112 and MOE2016-T2-1-019 / R146-000-234-112. Furthermore, S. Jain is supported in part by NUS grant C252-000-087-001. F. Stephan did part of the work while on sabbatical leave to UNSW Sydney.

of hypotheses converges to an index e for the language L (that is, $H_e = L$), then the learner is said to have learnt the input language from the text. The learner learns a language L if it learns it from all texts for L . It learns a class \mathcal{L} of languages if it learns all languages in \mathcal{L} . To measure the complexity of a learner, it is convenient to consider the learner as operating in cycles: it starts with hypothesis e_0 and in the n -th cycle, it gets the datum x_n and conjectures the hypothesis e_{n+1} . Freivalds et al. [10] and Kinber and Stephan [18] imposed the condition that between two cycles, the learner remembers only part of its previous inputs and works via some (long term) memory, which can be restricted. Thus, the complexity of learners can be measured in terms of two parameters: (a) the computational complexity of mapping the old memory and input datum to the new memory and hypothesis and (b) the length of the memory as a function of the length of the longest example seen so far.

Fundamental choices for (a) are: recursive learners (the mapping can be computed by a Turing machine), transduced learners (the mapping can be computed by a finite transducer) and automatic learners (the mapping is an automatic function). Pitt [23] showed that many complexity-theoretic restrictions—like requiring that the update time in each cycle be carried out in time polynomial in the sum of the lengths of all inputs seen so far—do not give a real restriction for most learning criteria from classical inductive inference. Automatic learners are more severely restricted and offer an interesting object of study [5, 13]. In particular, it is natural to investigate the target classes that are represented in an automata-theoretic framework, namely the automatic families [14]. These offer a representation that an automatic learner can handle easily. It is also natural to impose that hypothesis spaces be themselves automatic families containing the class to be learnt. It turns out that certain such families are learnable by a recursive learner, but not by an automatic learner. The inability to memorise all past data is a major weakness of automatic learners and is exploited by many non-learnability proofs. Still, as shown by Jain et al. [13], w.r.t. the learnability of automatic families from fat text (with infinitely many occurrences of each datum), automatic and recursive learners have the same power; their memory can even be restricted to the length of the longest datum seen so far, the so-called *word length memory limitation*.

The current work studies transduced learners which are a generalisation of automatic learners. For transduced learners, the update mapping of the learner is computed by a non-deterministic transducer which on all accepting runs, produces the same outputs for the same inputs. Both inputs (old memory and current datum) are read independently, and both outputs (new memory and hypothesis) are written independently. This independence makes transduced learners more powerful than automatic ones.

For (b), Freivalds et al. [10] imposed that learners operate in cycles and only remember, from one cycle to the next, information recorded in a (long term) memory, with restrictions on its length. For automatic and transduced learners, the memory is a string over a fixed alphabet, that may depend on the learner, whose size is measured by the length of the string [10, 13, 18]. In the subfield of

automatic learning, this way of restricting the memory led to fruitful findings, still leaving one major question open: is it truly restrictive to bound the memory by the length of the longest datum seen so far? A positive answer will be provided for transduced learners. Moreover, there is a learning hierarchy based on the memory sizes as a function of the length of the longest datum seen so far. In particular, polynomials and exponential functions of various degrees and exponents, respectively, are the most prominent memory bounds.

2 Preliminaries

Let \mathbb{N} denote the set of natural numbers $\{0, 1, \dots\}$. Let Σ denote a finite alphabet (set of symbols), ε the empty string, and Σ^* the set of all strings (words) over Σ . A language is a subset of Σ^* for some finite alphabet Σ . Concatenation of strings u and v is denoted by $u \cdot v$, or just uv when the context makes it clear. A string u of length n can be considered as a function from $\{0, 1, \dots, n - 1\}$ to Σ , with $u(i)$ the $(i + 1)$ -th symbol in u . Length lexicographic order between strings is defined as follows: $u <_l v$ if either $|u| < |v|$ or $|u| = |v|$ and u is lexicographically before v w.r.t. some underlying ordering of Σ .

2.1 Automatic Relations and Functions

A relation $R = \{(u_1, \dots, u_n) : u_i \in \Sigma^*\}$ is said to be *automatic* iff it is recognised by a finite automaton with n inputs which reads all inputs at the same speed (one symbol per input and cycle with a special symbol $\#$ when the corresponding input is exhausted) [4, 6, 12, 16, 17, 24]. A function f with m inputs and n outputs is said to be automatic iff the corresponding relation, that is, $R = \{(u_1, \dots, u_m, v_1, \dots, v_n) : f(u_1, \dots, u_m) = (v_1, \dots, v_n)\}$, is automatic. A class of languages $\{L_e : e \in I\}$ defined using indexing I is said to be an *automatic family* if I is regular and $\{(e, x) : x \in L_e\}$ is automatic.

2.2 Transducers

A *transducer* processes its inputs at different speeds; its transition function δ does not necessarily process one symbol from each input component, but possibly none or many. Formally, a transducer is a tuple $(Q, n, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, n is input arity, Σ is a finite set of symbols, $q_0 \in Q$ is the starting state, $F \subseteq Q$ is the set of final states, and the transition function δ is a subset of $Q \times (\Sigma^*)^n \times Q$. A run of a transducer is of the form: $(p_0, s_{1,1}, \dots, s_{1,n}, p_1), (p_1, s_{2,1}, \dots, s_{2,n}, p_2), \dots, (p_{k-1}, s_{k,1}, \dots, s_{k,n}, p_k)$, where p_0 is the starting state and for all $i < k$, $(p_i, s_{i+1,1}, s_{i+1,2}, \dots, s_{i+1,n}, p_{i+1}) \in \delta$. Note that the lengths of the $s_{i,j}$ may differ. The run is *accepting* if $p_k \in F$, and the accepted input is (w_1, w_2, \dots, w_n) , where $w_i = s_{1,i} s_{2,i} \dots s_{k,i}$ for $i = 1, 2, \dots, n$. The relation recognised by a transducer is the set of inputs it accepts (in some accepting run); such relations are called *rational* or *transduced*. Note that the main difference between an automatic relation and a transduced relation is that

the transducer can read the inputs independently at different speeds and therefore the non-determinism of the transducer may prove useful. Nivat [20] provided a characterisation when a relation is transduced which is based on the notion of a homomorphism. Here a homomorphism is a mapping h which replaces each symbol a by a possibly empty word $h(a)$. Now a k -ary relation R is transduced iff there is a regular set A and there are k homomorphisms h_1, \dots, h_k such that, for each symbol, at most one of the homomorphism h_1, \dots, h_k maps this symbol to a non-empty word and the relation R is the set $\{(h_1(u), \dots, h_k(u)) : u \in A\}$.

A function f is said to be *transduced* or *rational* if the relation $\{(u_1, \dots, u_m, v_1, \dots, v_n) : f(u_1, \dots, u_m) = (v_1, \dots, v_n)\}$ is rational and every input tuple (u_1, \dots, u_m) has a unique output tuple (v_1, \dots, v_n) . A class of languages $\{L_e : e \in I\}$ indexed by I is said to be a *transduced family* if I is regular and the relation $\{(e, x) : x \in L_e\}$ is recognised by some transducer; the class is automatic iff the relation is recognised by a finite automaton. Transduced families have some of the decidability properties of automatic families, in particular those below.

Proposition 1. *If $\{L_e : e \in I\}$ is a transduced family and a transducer \mathbf{M} accepts $\{(e, x) : x \in L_e, e \in I\}$, then one can effectively (from \mathbf{M} , parameter y and finite set D), for each of the following sets, find a DFA recognising it:*

- (a) $A_y = \{p : y \in L_p\};$
- (b) $A'_y = \{p : y \notin L_p\};$
- (c) $B_y = \{p : \{z \in L_p : |z| > |y|\} \neq \emptyset\};$
- (d) $B'_y = \{p : \{z \in L_p : |z| > |y|\} = \emptyset\};$
- (e) $C_D = \{p : D \subseteq L_p\};$
- (f) $C'_D = \{p : D = L_p\};$
- (g) $F_e = \{x : x \in L_e\}.$

In particular, given d and e , it can be effectively determined whether $L_d \subseteq L_e$ and whether $L_d \subset L_e$.

2.3 Learning Theory

Gold [11] defined a *text* as a mapping T from \mathbb{N} to $\Sigma^* \cup \{\#\}$, whose contents, denoted $\text{content}(T)$, is the set $\{T(x) : x \in \mathbb{N}\} \setminus \{\#\}$; it is a text for a language L iff $\text{content}(T) = L$. The initial segment of text T of length n is denoted $T[n]$.

To learn a target class $\mathcal{L} = \{L_e : e \in I\}$, defined using indexing I , a learner uses a hypothesis space $\mathcal{H} = \{H_e : e \in J\}$, defined using indexing J , with $\mathcal{L} \subseteq \mathcal{H}$.

The following notions are adapted from Gold [11]. A learner uses some alphabet Γ for its memory. It starts with an initial memory and hypothesis. On each datum, it updates its memory and hypothesis. That is, a *learner* is a mapping \mathbf{M} from $(\Gamma^* \cup \{?\}) \times (\Sigma^* \cup \{\#\})$ to $(\Gamma^* \cup \{?\}) \times (J \cup \{?\})$, together with an initial memory memo_0 and hypothesis hyp_0 . Intuitively, $?$ denotes both null memory (different from ε) and null hypothesis (when the learner issues no hypothesis). One can extend the definition of a learner to arbitrary initial sequences of texts T , setting $\mathbf{M}(T[0]) = (\text{memo}_0, \text{hyp}_0)$, and then inductively

setting $\mathbf{M}(T[n+1]) = (\text{mem}_{n+1}, \text{hyp}_{n+1}) = \mathbf{M}(\text{mem}_n, T(n))$. Intuitively, mem_n and hyp_n are the memory and conjecture of the learner after having seen the data in $T[n]$, respectively. A learner \mathbf{M} converges on text T to a hypothesis e iff for all but finitely many n , $\text{hyp}_n = e$. Note that the memory is not required to converge. A learner \mathbf{M} *explanatorily learns* a language L if for all texts T for L , \mathbf{M} converges on T to a hypothesis e with $H_e = L$. A learner \mathbf{M} (*explanatorily*) learns a class \mathcal{L} of languages iff it learns each $L \in \mathcal{L}$ [8, 9, 11].

Blum and Blum [3] defined a finite sequence σ to be a *locking sequence* for a learner \mathbf{M} on language L if (a) $\text{content}(\sigma) \subseteq L$, (b) the hypothesis e of \mathbf{M} on σ satisfies $H_e = L$, and (c) for all τ with $\sigma \subseteq \tau$ and $\text{content}(\tau) \subseteq L$, the hypothesis of \mathbf{M} on τ is e . They showed that if \mathbf{M} learns L then such a σ exists.

A learner \mathbf{M} is said to be recursive if the corresponding function F , mapping (old memory, datum) to (new memory, hypothesis), is recursive. Jain et al. [13] defined \mathbf{M} to be an *automatic* learner if F is automatic. Finally, \mathbf{M} is said to be a *transduced* learner if F can be computed by a transducer.

In this work, learners are recursive, and can or not be transduced or automatic. The memory limitations of the learner discussed in this paper is based on the length of the memory of the learner in terms of the length of the longest datum seen so far. Thus, for example, a learner \mathbf{M} is word size memory bounded if for some constant c , for all finite sequences σ , if $M(\sigma) = (\text{mem}, \text{hyp})$, and $n = \max\{|x| : x \in \text{content}(\sigma)\}$, then $|\text{mem}| \leq n + c$. Similarly, the learner is $O(n^2)$ memory bounded if $|\text{mem}| \leq cn^2 + c$.

Example 2. For all $e \in \{0, 1\}^+$, let $L_e = \{0, 1\}^* \setminus (\{0, 1\}^* \cdot \{e\})$ and consider the class defined by the transduced family $\{L_e : e \in \{0, 1\}^+\}$. The transduced learner for this family has its current memory always the same as the current hypothesis e (initialised to 0). For an input word x , if x ends with e then e is updated to its length-lexicographic successor else e remains unchanged. Thanks to its non-deterministic nature, a transducer can check whether x ends with e and give the corresponding output. Note that both outputs, new memory and new hypothesis, of the update function of this learner are always the same.

During the learning process, as long as the current value of e is length-lexicographically strictly below the target, the learner will eventually see an input ending with e , as there are infinitely many of these inputs, and then update the hypothesis and memory to the next binary word in length-lexicographical order. Eventually e reaches the correct value and then no further datum can cause another update of the hypothesis. Hence, one can verify that the transduced learner indeed converges to the correct hypothesis. Proposition 3 provides a more complicated version of this class that has a transduced but not an automatic learner.

3 Automatic versus Transduced Learners and Memory-Size Hierarchies for Transduced Learners

Proposition 5 shows that some automatic classes can be learnt by transduced learners but not by automatic ones. Proposition 3 shows that furthermore, if

one considers transduced classes, then a transduced learner can succeed while keeping the word-size memory bound, whereas no automatic learner succeeds.

Proposition 3. *There is a transduced family which can be learnt by a transducer with word-size memory while it does not have an automatic learner at all.*

The following result provides lower bounds for the long term memory in terms of the longest example seen so far.

Proposition 4. *Suppose that S is a regular set and $f(n) = |\{x \in S : |x| \leq n\}|$. Let $\mathcal{L} = \{L : \exists n [L \subseteq \{x \in S : |x| \leq n\} \text{ and } f(n) - 1 \leq |L| \leq f(n)]\}$. Then any learner, whether automatic, transduced or recursive, needs in the worst case memory of length at least $f(n)/c$, for some constant c , after having seen some sequence containing only words of length up to n .*

Proof. Consider any learner \mathbf{M} for \mathcal{L} which uses alphabet Γ for its memory.

For any n , consider $L = \{x \in S : |x| \leq n\}$. Suppose there are two finite sequences σ and τ with $\text{content}(\sigma) \neq \text{content}(\tau)$, $\text{content}(\sigma) \cup \text{content}(\tau) \subseteq L$, and the memories of \mathbf{M} after having seen the inputs σ and τ are the same. Let z be in the symmetric difference of $\text{content}(\sigma)$ and $\text{content}(\tau)$, say in $\text{content}(\sigma) \setminus \text{content}(\tau)$. Let T be a text for $L \setminus \{z\}$. Now, \mathbf{M} either converges to the same hypothesis on σT and τT or fails to converge on both. As σT and τT are texts for L and $L \setminus \{z\}$, respectively, which are different languages in \mathcal{L} , \mathbf{M} fails to learn at least one of these languages.

It follows that \mathbf{M} has at least $2^{f(n)}$ different memory values on different finite sequences with elements from S of length at most n . Thus, at least one of these memory values must have length at least $f(n)/c$, where $c = \log_2(|\Gamma|)$. \square

If $f(n) = n^k$ for some constant k , the lower bound is $\Omega(n^k)$; if $f(n) = c^n$ for some constant c , the lower bound is $\Omega(c^n)$ on the maximum length of the memory on input sequences containing words of length up to n . It will be shown that for some regular languages S , similar upper bounds are obtained.

Proposition 5. *Let $S = \{0\}^* \cdot \{1\}^*$. Let $f(n) = |\{x \in S : |x| \leq n\}|$ (which is equal to $(n^2 + 3n + 2)/2$). Define \mathcal{L} as the set of all languages L for which $\exists n [L \subseteq \{x \in S : |x| \leq n\} \text{ and } f(n) - 1 \leq |L| \leq f(n)]$. Then \mathcal{L} can be learnt by a transduced learner with memory size $O(n^2)$ but \mathcal{L} cannot be learnt by any automatic learner, even without explicit memory bounds.*

Proof. Suppose for a contradiction that an automatic learner \mathbf{M} learns \mathcal{L} . Fixing $n \in \mathbb{N}$, consider a sequence σ containing exactly n elements of length at most n from S . As \mathbf{M} is automatic, its memory on σ can be of length at most cn for some constant c , and thus the number of possible memories of \mathbf{M} after seeing σ is bounded by d^n for some constant d . On the other hand, there are at least $\binom{n^2/2}{n}$ possible contents of such sequences, and for large enough n , this is larger than d^n . Thus, for large enough n , there exist two sequences σ and σ' , with different content, each containing exactly n elements of length at most n , such that the

memory of \mathbf{M} after seeing σ and σ' is the same. Let $x \in \text{content}(\sigma) \setminus \text{content}(\sigma')$, and let T be a text for $S \cap \{y \in S : |y| \leq n\} \setminus \{x\}$. Now, \mathbf{M} on texts σT and $\sigma' T$ either does not converge or converges to the same conjecture even though they are texts for different languages in \mathcal{L} . Thus, \mathbf{M} cannot learn \mathcal{L} .

Now it is shown that a transduced learner can learn \mathcal{L} . For representation of languages in \mathcal{L} , the indices are of the form $0^i 1^j 2^k$ and 3^{k+1} . If $w = 0^i 1^j 2^k$ then L_w contains all words in S of length up to $i + j + k$ except for $0^i 1^j$. If $w = 3^{k+1}$ then L_w contains all members of S of length up to k .

The learner uses as memory a string of the form $\{0, 1\}^*$, where certain positions are marked. Intuitively, for memory $w = w(0)w(1)\dots w(n-1)$, each position in the memory string represents a string of the form $0^i 1^j$, the marked positions representing the strings that have been seen in the input. A position p in the string represents the string formed by taking the number of 0s in $w(0)w(1)\dots w(p)$, followed by $p - r$ 1s for the largest $r \leq p$ such that $w(r) = 0$; if there is no 0 in $w(0)w(1)\dots w(p)$ then r is taken to be -1 . Thus, a position p represents the string formed by taking the sequence of 0s up to position p (inclusive) followed by the number of 1s between position p (inclusive) and the position of the last 0 up to position p (inclusive).

For example, if the strings 00, 001 and 011 have been observed, then the value of the memory data structure is 011'0'1' and the strings represented by the positions of the marked (primed) letters as ordered in the memory word are 011, 00 and 001. Note that the principle of taking the “maximal numbers of 0s” before the third mark implies that the word 0111 is not represented in the above memory as there is a 0 between the 1s taken over. The overall goal of updating the memory is to let the current input word $0^i 1^j$ be represented in the memory by a position and the symbol at this position be marked. The beginning of the memory can be marked in order to record that ε has been observed in the input.

The learner starts with memory ε without any marks. Now suppose that at any time, (1) the new input word is $0^i 1^j$, (2) i' is the number of 0s in the current memory word, and (3) j' is the number of 1s in the current memory word which have exactly i 0s in the memory before their position. Note that j' is 0 if either $i' < i$ or $i' \geq i$ and after the first i 0s, either the word ends or another 0 follows. The non-deterministic transducer does the following:

1. First, while there is a 0 to be read in both memory and input datum d : read old memory copying each symbol to new memory and whenever a 0 is read, read it also on d until at least one of the memory or d has only symbols from $\{1\}^*$ left; thus the memory is copied until $\min\{i, i'\}$ 0s (along with intermediate 1s in the memory) are copied from the old memory and d has the first $\min\{i, i'\}$ symbols read.
2. If $i' \geq i$ (this is determined by the transducer by guessing, and verifying when reading the rest of the words) then read j 1s from the current datum and j' 1s from the old memory and write $\max\{j, j'\}$ 1s to the new memory. Then copy the remaining part of the old memory to the new memory.

3. If $i' < i$ then first copy all remaining 1s from the old memory to the new memory and then copy the remaining symbols $0^{i-i'}1^j$ from the current datum to the new memory.
4. Besides this, all symbols copied from the old memory to the new memory keep their marks in case they have some already, and the symbol at the position representing 0^i1^j in the new memory also receives a mark.

What follows demonstrates how the hypothesis is written when writing to the new memory, as both outputs are independently written into different words. Still, the workings of the transducer is best understood when the transducer is thought of as non-deterministically extracting hypothesis from new memory.

Note that the memory keeps track of all words seen in the input using the marks. It can also be used to indicate missing words: if a position representing 0^i1^j is unmarked, then 0^i1^j has not been seen in the input; if the i -th 0 in the memory does not have a 1 preceding it, then $0^{i-1}1$ is not seen in the input. Call a word v stored in the memory maximal iff $v1$ is not represented in the memory. Note that if all positions in the memory are marked and the lengths of all maximal words in the memory except the maximal word v have the same parity as the length of $v1$, then $v1$ has not been seen in the input so far, even though $v1$'s length is at most that of the maximal word in the input language.

When writing the new hypothesis, the learner can verify and act according to the first of the following cases which applies. Suppose i' is the total number of 0s in the memory.

1. If the position representing ε is not marked, then the hypothesis is $2^{i'}$;
2. If the memory ends in 1, then the hypothesis is $0^{i'+1}$;
3. If some 0 in the memory is not preceded by a 1, then the hypothesis is $0^{i-1}1^{i'-i+1}$ for the least i such that the i -th 0 is not preceded by a 1;
4. If some 0 in the memory is not marked then the hypothesis is $0^i2^{i'-i}$ for the least i such that the position representing 0^i is not marked;
5. If some 1 is not marked then the hypothesis is $0^i1^j2^{j'-j}$ where j is such that 0^i1^j is represented by the position of the leftmost unmarked 1 and j' is the number of 1s between the i -th and $(i+1)$ -st 0s in memory;
6. If all positions are marked and the lengths of all maximal words represented, except for the maximal word v , share the same parity, then the hypothesis is $v1$;
7. If none of the above conditions applies then the hypothesis is $3^{i'+1}$, which is the set of all words in S of length up to i' .

Note that in order to check the sixth case, the transducer can always count the number of 0s up to the current position modulo 2, and then count the number of 1s following this 0 modulo 2. Also, the transduced learner can easily verify for any particular case that none of the earlier cases applies. Thus, the learner can generate the hypothesis based on the above.

The memory keeps track of all data seen in the above described data structure. Also, the hypothesis is computed in such a way that it is correct whenever all of the shortest $f(n)$ members of S except perhaps one have been seen, but no

longer data have been observed. As an example, the following table illustrates data, memory and hypothesis updates of the learner. The initial memory is ε and the old memory is always the new memory of the previous step.

Datum	New memory	w	Words in L_w
1	1'	ε	None
01	1'01'	00	$\varepsilon, 0, 1, 01, 11$
ε	'1'01'	00	$\varepsilon, 0, 1, 01, 11$
00	'1'01'0'	02	$\varepsilon, 1, 00, 01, 11$
0	'1'0'1'0'	11	$\varepsilon, 0, 1, 00, 01$
11	'1'1'0'1'0'	333	$\varepsilon, 0, 1, 00, 01, 11$

The above example illustrates parts of the proof. \square

The previous proof can be generalised to larger alphabet sizes; however, one has to fix the alphabet size and exponent of the polynomial. Thus both preceding results give the following corollary, in which the bound $\Theta(n^k)$ for the family given by \mathcal{L}_k indicates that one can learn with memory size $O(n^k)$, but every learner needs at least memory size $\Omega(n^k)$.

Corollary 6. *Suppose Σ has k symbols $0, 1, \dots, k-1$, and let S_k be $\{0\}^* \cdot \{1\}^* \cdots \{k-1\}^*$. Let \mathcal{L}_k be the class of all $L_{vw} = \{x \in S_k : |x| \leq |vw| \text{ and } x \neq v\}$ for all $w \in \{k\}^*$ and $v \in \{0\}^* \cdot \{1\}^* \cdots \{k-1\}^*$, and L_w be $\{x \in S_k : |x| < |w|\}$ for all $w \in \{k+1\}^+$. Note that the number of elements in S_k of length at most n is $f(n) = \binom{n+k}{k} = \sum_{m \leq n} \binom{m+k-1}{k-1}$. Now, \mathcal{L}_k can be learnt by a transduced learner with a memory length bound of $\Theta(n^k)$, where n is the length of the longest example seen so far.*

An additional corollary to Proposition 5 can be obtained with respect to target-sized learners. The result uses the following fact which Jain et al. [14] showed for all automatic families: there is a constant c such that for each language L_e , if words in the language L_e have at most length n then the shortest index d of L_e has at most length $n+c$. Stephan [25] defined a learner to have a *target-sized* memory bound if the length of the memory is never longer than the length of the shortest index of the language being learnt plus a constant. If one relaxes this bound by just requiring the existence of a function f such that the memory is never longer than $f(n)$ with n being the size of the shortest index of the target, then one can get the following corollary.

Corollary 7. *The class \mathcal{L}_k (from Corollary 6) of all subsets of $\{0\}^* \cdot \{1\}^* \cdots \{k-1\}^*$ of words up to length n except perhaps one, can be learnt by a transduced learner with target-sized memory of size $O(f(n))$, where $f(n) = \binom{n+k}{k}$, but not with any better memory constraint, except for a multiplicative constant.*

The class of all languages of binary words up to length n except perhaps for one can be learnt by a transduced learner with exponential target-sized memory.

The class of all $L_w = \{v \in \Sigma^* : \varepsilon <_U v \leq_U w\}$ with $w \neq \varepsilon$ and $L_\varepsilon = \Sigma^*$ cannot be learnt with any type of target-sized memory.

Proposition 8. If an automatic class can be explanatorily learnt from text then there is a transduced learner for the same class which learns it with $O(c^n)$ sized memory where c is some constant that depends only on the class and where n is the size of the longest datum seen so far.

4 A Space Bound for Learning All Learnable Transduced Classes

A learner \mathbf{M} is set-driven [21] if for all sequences σ and τ such that $content(\sigma)$ is equal to $content(\tau)$, \mathbf{M} 's memory and hypothesis are the same after seeing either σ or τ . It is first shown that every learnable transduced family can be learnt by a set-driven recursive learner, which can be obtained uniformly from a transducer learner for the family. This learner is defined for all transduced families. However, for unlearnable families, the learner will fail on some input texts. The learner employs the properties of transduced families listed in Proposition 1.

A tell-tale set for a language L with respect to a class \mathcal{L} of languages is a finite subset D of L such that for all $L' \in \mathcal{L}$, if $D \subseteq L' \subseteq L$ then $L' = L$. Angulin [1] has shown that for any learnable family of languages \mathcal{L} , every language in \mathcal{L} has a tell-tale with respect to \mathcal{L} .

Proposition 9. For every learnable transduced family $\mathcal{L} = \{L_e : e \in I\}$, some set-driven recursive learner learns that family. Furthermore, this learner can be effectively obtained from the transducer describing the transduced family.

Hence for some recursive function g , the memory of the recursive learner is bounded by $g(n)$ where n is the length of the longest datum seen so far.

Proof. The learner is given by the following algorithm. Let $D = \{a_1, a_2, \dots, a_m\}$ be the set of words observed in the input so far (where each a_i is distinct). Let $f(D)$ be the length-lexicographically least index e with $D \subseteq L_e$ (note that by Proposition 1, this can be found effectively). Let n be the maximum of

- the length of the description of the DFA accepting I ,
- the length of the description of the transducer accepting the set defined as $\{(e, x) : x \in L_e, e \in I\}$ for the transduced family $\{L_e : e \in I\}$ to be learnt,
- the alphabet size for the transduced family and
- the lengths $|a_1|, |a_2|, \dots, |a_m|$ of all words in D .

Now the learner \mathbf{M} chooses the first of the following options which applies:

1. If there is an $e \in I$ with $L_e = D$ then \mathbf{M} outputs the length-lexicographically least such e (by Proposition 1, this is decidable for transduced families);

2. If $f(D)$ is defined, because there is e with $D \subseteq L_e$, then \mathbf{M} selects the length-lexicographically least index e of length at most $|f(D)| + n$ such that $D \subseteq L_e$ and there is no index d of length at most $|f(D)| + n$ with $D \subseteq L_d \subset L_e$;
3. Otherwise, D is not consistent with any hypothesis in the family and \mathbf{M} outputs ? to signal that there is no valid conjecture.

By definition, \mathbf{M} is set-driven. Furthermore, if the input language L_e is finite then \mathbf{M} will converge to its length-lexicographically least index after having seen all elements. If L_e is infinite then there must exist a finite subset D of L (a tell-tale set) such that there is no language L_d in \mathcal{L} such that $D \subseteq L_d \subset L_e$. Thus, for large enough n , and thus after having received large enough datum, step 2 would output the least index for L_e . Thus, \mathbf{M} learns all languages in \mathcal{L} .

For the function g , note that there are only finitely many pairs of descriptions of transduced families $\{L_e : e \in I\}$ and data sets $D \subseteq \{0, 1, \dots, n-1\}^*$ such that their size is bounded by n . One can therefore take $g(n)$ to be the maximum of the space used by the algorithm when run with the given parameterisation describing the transduced family and the data set D as input. The number $g(n)$ can be algorithmically computed from n . \square

Proposition 10. *If a class has a set-driven recursive learner using space bound $g(n)$, then it also has a transduced learner using space bound $g(n) + c^n$ for its memory, for some constant c , with n denoting the size of the longest datum seen so far.*

5 Conclusion and Subsequent Work

It was demonstrated that whereas many questions on memory usage remain open for automatic learners, transduced learners that learn a transduced family can always bound the memory size as a function of the longest datum seen so far. When learning automatic families, concrete bounds have been found, and a hierarchy of polynomial and exponential bounds has emerged. It has been shown that every family can be learnt using some exponential bound.

Subsequent work addressed the question of the extent to which transduced learners can satisfy additional properties like consistency [2], conservativeness [1, 22] and iterativeness [26]. Many learners constructed here can be made iterative; furthermore, transduced learners can be made consistent and conservative, similarly to the polynomial-time setting [7]; however, these criteria cannot be combined with more restrictive memory-limitations. These results have been delayed to the full version of the paper due to space limitations.

References

1. Angluin, D.: Inductive inference of formal languages from positive data. *Inf. Control* **45**, 117–135 (1980)
2. Bārzdiņš, J.: Inductive inference of automata, functions and programs. In: Proceedings of the 20th International Congress of Mathematicians, Vancouver, pp. 455–460 (1974). (in Russian). English translation in American Mathematical Society Translations: Series 2, 109:107–112 (1977)
3. Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. *Inf. Control* **28**, 125–155 (1975)
4. Blumensath, A., Grädel, E.: Automatic structures. In: 15th Annual IEEE Symposium on Logic in Computer Science, LICS 2000. pp. 51–62 (2000)
5. Case, J., Jain, S., Le, T.D., Ong, Y.S., Semukhin, P., Stephan, F.: Automatic learning of subclasses of pattern languages. In: Dediū, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 192–203. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21254-3_14
6. Case, J., Jain, S., Seah, S., Stephan, F.: Automatic functions, linear time and learning. *Log. Methods Comput. Sci.* **9**(3:19), 1–26 (2013)
7. Case, J., Kötzing, T.: Difficulties in forcing fairness of polynomial time inductive inference. In: Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S. (eds.) ALT 2009. LNCS (LNAI), vol. 5809, pp. 263–277. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04414-4_23
8. Case, J., Lynes, C.: Machine inductive inference and language identification. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP 1982. LNCS, vol. 140, pp. 107–115. Springer, Heidelberg (1982). <https://doi.org/10.1007/BFb0012761>
9. Case, J., Smith, C.: Comparison of identification criteria for machine inductive inference. *Theor. Comput. Sci.* **25**, 193–220 (1983)
10. Freivalds, R., Kinber, E., Smith, C.H.: On the impact of forgetting on learning machines. *J. ACM* **42**, 1146–1168 (1995)
11. Gold, E.M.: Language identification in the limit. *Inf. Control* **10**, 447–474 (1967)
12. Hodgson, B.R.: Théories décidables par automate fini. Ph.D. thesis, Département de mathématiques et de statistique, Université de Montréal (1976)
13. Jain, S., Luo, Q., Stephan, F.: Learnability of automatic classes. *J. Comput. Syst. Sci.* **78**, 1910–1927 (2012)
14. Jain, S., Ong, Y.S., Pu, S., Stephan, F.: On automatic families. In: Proceedings of the Eleventh Asian Logic Conference, in Honor of Professor Chong Chitat on his Sixtieth Birthday, pp. 94–113. World Scientific (2012)
15. Jain, S., Osherson, D.N., Royer, J.S., Sharma, A.: Systems That Learn, 2nd edn. Bradford—MIT Press, Cambridge (1999)
16. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60178-3_93
17. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: richness and limitations. *Log. Methods Comput. Sci.* **3**(2:2), 1–18 (2017)
18. Kinber, E., Stephan, F.: Language learning from texts: mind changes, limited memory and monotonicity. *Inf. Comput.* **123**, 224–241 (1995)
19. Lange, S., Zeugmann, T., Zilles, S.: Learning indexed families of recursive languages from positive data: a survey. *Theor. Comput. Sci.* **397**, 194–232 (2008)
20. Nivat, M.: Transductions des langages de Chomsky. *Annales de l’Institut Fourier, Grenoble* **18**, 339–455 (1968)

21. Osherson, D., Stob, M., Weinstein, S.: Learning strategies. *Inf. Control* **53**, 32–51 (1982)
22. Osherson, D., Stob, M., Weinstein, S.: Systems That Learn. An introduction to learning theory for cognitive and computer scientists. Bradford—MIT Press, Cambridge (1986)
23. Pitt, L.: Inductive inference, DFAs, and computational complexity. In: Jantke, K.P. (ed.) AII 1989. LNCS, vol. 397, pp. 18–44. Springer, Heidelberg (1989). https://doi.org/10.1007/3-540-51734-0_50
24. Rubin, S.: Automata presenting structures: a survey of the finite string case. *Bull. Symb. Log.* **14**, 169–209 (2008)
25. Stephan, F.: Methods and theory of automata and languages. School of Computing, National University of Singapore (2016)
26. Wiehagen, R.: Limes-erkennung rekursiver funktionen durch spezielle strategien. *J. Inf. Process. Cybern. (EIK)* **12**(1–2), 93–99 (1976)



Model Learning as a Satisfiability Modulo Theories Problem

Rick Smetsers, Paul Fiterău-Broștean^(✉), and Frits Vaandrager

Institute for Computing and Information Sciences,
Radboud University, Nijmegen, The Netherlands
fiteraup@yahoo.com

Abstract. We explore an approach to model learning that is based on using *satisfiability modulo theories* (SMT) solvers. To that end, we explain how DFAs, Mealy machines and register automata, and observations of their behavior can be encoded as logic formulas. An SMT solver is then tasked with finding an assignment for such a formula, from which we can extract an automaton of minimal size. We provide an implementation of this approach which we use to conduct experiments on a series of benchmarks. These experiments address both the scalability of the approach and its performance relative to existing active learning tools.

1 Introduction

We are interested in algorithms that construct black-box state diagram models of software and hardware systems by observing their behavior and performing experiments. Developing such algorithms is a fundamental research problem that has been widely studied. Roughly speaking, two approaches have been pursued in the literature: *passive learning* techniques, where models are constructed from (sets of) runs of the system, and *active learning* techniques, that accomplish their task by actively doing experiments on the system.

Gold [11] showed that the passive learning problem of finding a minimal DFA that is compatible with a finite set of positive and negative examples, is NP-hard. In spite of these hardness results, many DFA identification algorithms have been developed over time, see [13] for an overview. Some of the most successful approaches translate the DFA identification problem to well-known computationally hard problems, such as SAT [12], vertex coloring [10], or SMT [18], and then use existing solvers for those problems.

Angluin [5] presented an efficient algorithm for active learning a regular language L , which assumes a *minimally adequate teacher* (MAT) that answers two types of queries about L . With a *membership query*, the algorithm asks whether or not a given word w is in L , and with an *equivalence query* it asks whether or

This work is supported by the Netherlands Organization for Scientific Research (NWO) projects 628.001.009 on Learning Extended State Machine for Malware Analysis (LEMMA), and 612.001.216 on Active Learning of Security Protocols (ALSeP).

This paper greatly extends earlier work [24].

not the language L_H of an hypothesized DFA H is equal to L . If L_H and L are different, a word in the symmetric difference of the two languages is returned. Angluin's algorithm has been successfully adapted for learning models of real-world software and hardware systems [19, 21, 25], as shown in Fig. 1. A membership query (MQ) is implemented by bringing the *system under learning* (SUL) in its initial state and the observing the outputs generated in response to a given input sequence, and an equivalence query (EQ) is approximated using a *conformance testing tool* (CT) [17] via a finite number of *test queries* (TQ). If these test queries do not reveal a difference in the behavior of an hypothesis H and the SUL, then we assume the hypothesis model is correct.

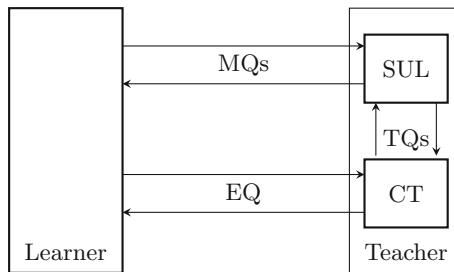


Fig. 1. Model learning within the MAT framework.

Walkinshaw et al. [26] observed that from each passive learning algorithm one can trivially construct an active learning algorithm that only poses equivalence queries. Starting from the empty set of examples, the passive algorithm constructs a first hypothesis H_1 that is forwarded to the conformance tester. The first counterexample w_1 of the conformance tester is then used to construct a second hypothesis H_2 . Next counterexamples w_1 and w_2 are used to construct hypothesis H_3 , and so on, until no more counterexamples are found.

In this article, we compare the performance of existing active learning algorithms with passive learning algorithms that are ‘activated’ via the trick of Walkinshaw et al. [26]. At first, this may sound like a crazy thing to do: why would one compare an efficient active learning algorithm, polynomial in the size of the unknown state machine, with an algorithm that makes a possibly super-polynomial number of calls [6] to a solver for an NP-hard problem? The main reason is that in practical applications i/o interactions often take a significant amount of time. In [22], for instance, a case study of an interventional X-ray system is described in which a single i/o interaction may take several seconds. Therefore, the main bottleneck in these applications is the total number of membership and test queries, rather than the time required to decide which queries to perform. Also, in practical applications the state machines are often small, with at most a few dozen states (see for instance [3, 4, 22]). Therefore, even though passive learning algorithms do not scale well, there is hope that they can still

handle these applications. Active learning algorithms rely on asking a large number of membership queries to construct hypotheses. Passive learning algorithms pose no membership queries, but instead need a larger number of equivalence queries, which are then approximated using test queries. A priori, it is not clear which approach performs best in terms of the total number of membership and test queries needed to learn a model.

Our experiments compare the original L^* [5] and the state-of-the-art TTT [15] active learning algorithm with an SMT-based passive learning algorithm on a number of practical benchmarks. We encode the question whether there exists a state machine with n states that is consistent with a set of observations into a logic formula, and then use the Z3 SMT solver [9] to decide whether this formula is satisfiable. By iteratively incrementing the number of states we can find a minimal state machine consistent with the observations. As equivalence oracle we use a state-of-the-art conformance testing algorithm based on adaptive distinguishing sequences [16, 23]. In line with our expectations, the passive learning approach is competitive with the active learning algorithms in terms of the number of membership and test queries needed for learning.

An advantage of SMT encodings, when compared for instance with encodings based on SAT or vertex coloring, is the expressivity of the underlying logic. In recent years, much progress has been made in extending active learning algorithms to richer classes of models, such as register automata [2, 8, 14] in which data may be tested and stored in registers. We show that the problem of finding a register automaton that is consistent with a set of observations can be expressed as an SMT problem, and compare the performance of the resulting learning algorithm with that of Tomte [2], a tool for active learning of register automata, on some simple benchmarks. New algorithms for active learning of FSMs, Mealy machines and various types of register automata are often extremely complex, and building tools implementations often takes years [2, 8, 15]. Adapting these tools to slightly different scenarios is typically a nightmare. One such scenario is when the system is missing *reset* functionality. This renders most active learning tools impractical, as these rely on the ability to reset the system. Developing SMT-based learning algorithms for register automata in settings with and without resets only took us a few weeks. This shows that the SMT-approach can be quite effective as a means for prototyping learning algorithms in various settings.

The rest of this paper is structured as follows. Section 2 describes how one can encode the problem of learning a minimal consistent automaton in SMT. The scalability and effectiveness of our approach, and its applicability in practice are assessed in Sect. 3. Conclusions are presented in Sect. 4.

2 Model Learning as an SMT Problem

This section describes how to express the existence of an automaton A of at most n states that is consistent with a set of observations S in a logic formula. *If and only if* there exists an assignment to the variables of this formula that makes it true, then exists an automaton A with at most n states that is consistent with S .

We use an SMT solver to find such an assignment. If the SMT solver concludes that the formula is satisfiable, then its solution provides us with A .

We distinguish three types of *constraints*:

- *axioms* must be satisfied for A to behave as intended by its definition.
- *observation constraints* must be satisfied for A to be consistent with S .
- *size constraints* must be satisfied for A to have n states or less.

Hence, the problem can be solved by iteratively incrementing n until the encoding of the axioms, observation constraints and size constraints is satisfiable.

In the following subsections, we present encodings for deterministic finite automata (Sect. 2.1), register automata (Sect. 2.2), and input-output register automata (Sect. 2.3). Not included here are encodings for Mealy and Moore machines. These are available in the extended version of this work¹.

2.1 An Encoding for Deterministic Finite Automata

A *deterministic finite automaton* (DFA) accepts and rejects *strings*, which are sequences of labels. We define a DFA as follows.

Definition 1. A DFA is a tuple (L, Q, q_0, δ, F) comprising a finite, non-empty set of labels L , a finite non-empty set of states Q , an initial state $q_0 \in Q$, a transition function $\delta : Q \times L \rightarrow Q$ and a set of accepting states $F \subseteq Q$.

Let x be a string. We use x_i to denote i th label of x . We use $x_{[i,j]}$ to denote the substring of x starting at position i and ending at position j (inclusive), i.e. $x = x_{[1,|x|]}$. A DFA accepts a string x if its computation ends in an accepting state, or more formally, if $\delta(q_0, x) \in F$ where δ is extended to strings.

Let S_+ be a set of strings that should be accepted, and let S_- be a disjoint set of strings that should be rejected. Let S be the set that contains all of these strings, along with their labels, i.e. $S = \{(x, \text{true}) : x \in S_+\} \cup \{(x, \text{false}) : x \in S_-\}$. A DFA is *consistent* with S if it accepts all strings in S_+ , and rejects all strings in S_- .

In our DFA encoding, Q is a finite subset of the (non-negative) natural numbers \mathbb{N} with $q_0 = 0$, while F is encoded as a function $\lambda : Q \rightarrow \mathbb{B}$, such that $q \in F \iff \lambda(q) = \text{true}$.

Similarly to Heule and Verwer [12] and Bruynooghe et al. [7], we arrange elements of S in an *observation tree* (OT) with the following definition:

Definition 2. An OT for $S = \{S_+, S_-\}$ is a tuple (L, Q, λ) , where L is a set of labels, $Q = \{x \in L^* : x \text{ is a prefix of a string in } S_+ \cup S_-\}$, $\lambda : S_+ \cup S_- \rightarrow \mathbb{B}$ is a output function for the strings, with $x \in S_+ \iff \lambda(x) = \text{true}$.

An OT allows us to provide efficient encodings for the labeled strings in S . Suppose an OT $T = (L, Q^T, \lambda^T)$ for a given set $S = \{S_+, S_-\}$. To find a DFA

¹ <https://gitlab.science.ru.nl/rick/z3gi/blob/lata/resources/paper.pdf>.

$A = (L, Q^A, q_0, \delta^A, F)$ consistent with S , we define a surjective (i.e. many-to-one) function $\text{map} : Q^T \rightarrow Q^A$ encoding correspondence between prefixes in the OT and states in A . We then introduce the following observation constraints:

$$\text{map}(\epsilon) = q_0 \quad (1)$$

$$\forall xl \in Q^T : x \in L^*, l \in L \quad \delta^A(\text{map}(x), l) = \text{map}(xl) \quad (2)$$

$$\forall x \in S_+ \cup S_- \quad \lambda^A(\text{map}(x)) = \lambda^T(x) \quad (3)$$

Equation 1 maps the empty string to the initial state of A . Equation 2 encodes the observed prefixes as transitions of A while Eq. 3 encodes the observed outputs, with λ^A encoding F .

Lastly, we limit A to at most n states by the following size constraint:

$$\forall q \in \{0, \dots, n-1\} \quad \forall l \in L \quad \bigvee_{q'=0}^{n-1} \delta(q, l) = q' \quad (4)$$

2.2 An Encoding for Register Automata

A *register automaton* (RA) can be seen as an automaton that is extended with a set of *registers* that can store data parameters. The values in these registers can then be used to express conditions over the transitions of the automaton, or *guards*. If the guard is satisfied the transition is fired, possibly storing the provided data parameter (this is called an *assignment*) and bringing the automaton from the current *location* to the next. As such, an RA can be used to accept or reject sequences of label-value pairs. Unlike finite automata, “states” in a register automaton are called locations because the *state* of the automaton also comprises the values of the registers. Therefore, an infinite number of possible states can be modeled using a small number of locations and registers.

The RAs we define have the following restrictions. Transitions in an RA do not imply (dis)equality of distinct registers (*right invariance*), values cannot be moved from one register to another (*non-swapping*) and registers always store unique values (*unique-valued*). The first two restrictions help simplify the encoding while the third is necessary to avoid the non-determinism caused by two used registers holding the same value. RAs with these restrictions can require more locations to be consistent with a given set of action strings, than those without. However, they are equally expressive. For a formal treatment of these restrictions and their implications, we refer to [2].

We define an RA as follows.

Definition 3. An RA is a tuple $(L, R, Q, q_0, \delta, \lambda, \tau, \pi)$. Therein, L , Q , q_0 and λ are a set of labels, a set of locations, the start location, and a location output function respectively. R is a finite set of registers. $\delta : Q \times L \times (R \cup \{r_\perp\}) \rightarrow Q$ is a register transition function. $\tau : Q \times R \rightarrow \mathbb{B}$ is a register use predicate, and $\pi : Q \times L \rightarrow (R \cup \{r_\perp\})$ is a register update function.

We call a label-value pair an *action* and denote it $l(v)$ for input label l and parameter v . We assume w.l.o.g. that parameter values are integers (\mathbb{Z}). A sequence of actions is called an *action string*, and is denoted by σ . A set of observations S for an RA comprises a set of action strings that should be accepted S_+ , and a set of action strings that should be rejected S_- . An RA is consistent with $S = \{S_+, S_-\}$ if it accepts all action strings in S_+ , and rejects all action strings in S_- .

Roughly speaking, an RA can be described as a DFA (Definition 1) enriched with a finite set of registers R and two additional functions. The first function, τ , specifies which registers are in use in a location. In a location q there can be two types of transitions for a label l and parameter value v . If v is equal to some used register r , then the transition $\delta(q, l, r)$ is taken. Else (v is different to all used registers), the *fresh* transition $\delta(q, l, r_\perp)$ is taken.

The second function, π , specifies if and where to store a value v when a fresh transition ($\delta(q, l, r_\perp)$) is taken. If $\pi(q, l) = r_\perp$ then the value v on transition $\delta(q, l, r_\perp)$ is not stored. Else (if $\pi(q, l) = r$ for some register $r \in R$), the value v on transition $\delta(q, l, r_\perp)$ is stored in register r .

For the RA to behave as intended we introduce the following axioms. First, we require that no registers are used in the initial location:

$$\forall r \in R \quad \tau(q_0, r) = \text{false} \quad (5)$$

Second, if a register is used after a transition, it was used before or updated:

$$\begin{aligned} \forall q \in Q \quad \forall l \in L \quad \forall r \in R \quad \forall r' \in (R \cup \{r_\perp\}) \\ \tau(\delta(q, l, r'), r) = \text{true} \implies (\tau(q, r) = \text{true} \vee (r' = r_\perp \wedge \pi(q, l) = r)) \end{aligned} \quad (6)$$

Third, if a register is updated, then it is used afterwards:

$$\forall q \in Q \quad \forall l \in L \quad \forall r \in R \quad \pi(q, l) = r \implies \tau(\delta(q, l, r_\perp), r) = \text{true} \quad (7)$$

Our goal is to learn an RA that is consistent with a set of action strings $S = \{S_+, S_-\}$. For this, we need to define a function that keeps track of the valuation of registers during runs over these action strings. Let $A = (L, R^A, Q^A, q_0, \delta^A, \lambda^A, \tau^A, \pi^A)$ be an RA, and let $T = (L \times \mathbb{Z}, Q^T, \lambda^T)$ be an OT for S . In addition to the *map* function ($\text{map} : Q^T \rightarrow Q^A$), we define a *valuation function* $\text{val} : Q^T \times R^A \rightarrow \mathbb{Z}$ that maps a state of T and a register of A to the value the register contains in that state.

Before constructing constraints for action strings, we first *canonize* them by making them *neat* [2, Definition 7]. An action string is *neat* if each parameter value is equal to either a previous value, or to the largest preceding value plus one. To exemplify, let a be an input label, and let $a(3)a(1)a(3)a(45)$ be an action string, then $a(0)a(1)a(0)a(2)$ is its corresponding neat action string. Aarts et al. [1] show that an RA can be learned by just studying its neat action strings.

Observation constraints for an RA are constructed as follows. First, we map the empty string to the initial location of A (Eq. 1). Second, we assert that a register is updated if and only if its valuation changes:

$$\forall \sigma l(v) \in Q^T \quad \forall r \in R^A \quad val(\sigma l(v), r) \neq val(\sigma, r) \Leftrightarrow \pi^A(map(\sigma), l) = r \quad (8)$$

We proceed by formulating how a register's valuation changes:

$$\forall \sigma l(v) \in Q^T \quad \forall r \in R^A$$

$$val(\sigma l(v), r) = \begin{cases} v & \text{if } \delta^A(map(\sigma), l, r_{\perp}) = map(\sigma l(v)) \\ & \quad \wedge \pi(map(\sigma), l) = r \\ val(\sigma, r) & \text{otherwise} \end{cases} \quad (9)$$

We then encode the observed transitions:

$$\forall \sigma l(v) \in Q^T$$

$$map(\sigma l(v)) = \begin{cases} \delta^A(map(\sigma), l, r) & \text{if } \exists!r \in R : \tau^A(map(\sigma), r) = \text{true} \\ & \quad \wedge val(\sigma, r) = v \\ \delta^A(map(\sigma), l, r_{\perp}) & \text{otherwise} \end{cases} \quad (10)$$

Finally, we encode the observed outputs, which can be done in the same way as for DFAs (see Eq. 3).

The task for the SMT solver is to find a solution that is consistent with these constraints. Obviously, we are interested in an RA with the minimal number of locations and registers. The number of locations can be limited in the same way as we limited states in DFAs (see Eq. 4). The number of registers is defined by the variables r that we quantify over in the presented equations. Therefore, they can be limited as such. In our case, the number of registers is never higher than the number of locations (because we can only update a single register from each location). Hence, the learning problem can be solved by iteratively incrementing the number of locations n , and for each n incrementing the number of registers from 1 to n , until a satisfiable encoding is found.

2.3 An Extension for Input-Output Register Automata

An *input-output register automaton* (IORA) is a register automaton transducer that generates an output action (i.e. label and value) after each input action. As in the RA-case, we restrict both input and output labels to a single parameter. Input and output values may update registers. Input values may be tested for (dis)equality with values in registers. Output values can be equal to the values stored, or may be fresh. As such, an input-output register automaton can be used for modeling software that produces parameterized outputs.

For a formal description of IORAs we refer to [1]. We define an IORA in Definition 4. Again, in the interest of our encoding, our definition is very different from that in [1]. Despite this, the semantics are similar.

Definition 4. *An IORA is a tuple $(I, O, R, Q, q_0, \delta, \lambda, \tau, \pi, \omega)$. Therein, I and O are finite, disjoint sets of input and output labels. R , Q , q_0 , τ and π are the same*

as for an RA (Definition 3). $\delta : (Q \cup \{q_\perp\}) \times (I \cup O) \times (R \cup \{r_\perp\}) \rightarrow (Q \cup \{q_\perp\})$ is a register transition function with a sink location. $\lambda : (Q \cup \{q_\perp\}) \rightarrow \mathbb{B}$ is a location output function with a sink location while $\omega : Q \rightarrow \mathbb{B}$ is a location type function which returns **true** if a location is an input location, and **false** if it is an output location.

A set of observations S for an IORA consists of *action traces*, which are pairs (σ^I, σ^O) where $\sigma^I \in (I \times \mathbb{Z})^*$ is an *input action string*, and $\sigma^O \in (O \times \mathbb{Z})^*$ is an *output action string* with $|\sigma^I| = |\sigma^O|$. An IORA is consistent with a set S if for each pair $(\sigma^I, \sigma^O) \in S$ it generates σ^O when provided with σ^I .

Despite that semantically an IORA is a transducer, we define it as an RA (Definition 3) which distinguishes between input and output labels, and which defines an additional function ω for the location type. From an *input location* transitions are allowed only for input actions. After an input action the IORA reaches an *output location*, in which a *single* transition is allowed. This transition determines the output action generated in response, and the input location the IORA will transition to. Transitions that are not allowed lead to a designated *sink location*, which is denoted q_\perp .

Using this definition we incorporate the axioms defined for our RA encoding (Eqs. 5–7) also in our IORA encoding. To these, we add the following axioms for an IORA to behave as intended.

First, observe that we do not use λ as an output function for an IORA. Instead, we use it to denote which locations are allowed. Hence, we require that the sink location q_\perp is the only rejecting location:

$$\forall q \in (Q \cup \{q_\perp\}) \quad \lambda(q) = \begin{cases} \text{false} & \text{if } q = q_\perp \\ \text{true} & \text{otherwise} \end{cases} \quad (11)$$

Second, we require that transitions do not lead to the sink location:

$$\forall q \in Q \quad \forall o \in O \quad \forall r \in (R \cup \{r_\perp\}) \quad \omega(q) = \text{true} \implies \delta(q, o, r) = q_\perp \quad (12)$$

$$\forall q \in Q \quad \forall i \in I \quad \forall r \in (R \cup \{r_\perp\}) \quad \omega(q) = \text{false} \implies \delta(q, i, r) = q_\perp \quad (13)$$

$$\forall l \in I \cup O \quad \forall r \in (R \cup \{r_\perp\}) \quad \delta(q_\perp, l, r) = q_\perp \quad (14)$$

Finally, we require that input locations are *input enabled* (Eq. 15), and that there is only one transition possible in an output location (Eq. 16):

$$\forall q \in Q \quad \forall i \in I \quad \forall r \in (R \cup \{r_\perp\}) \quad \omega(q) = \text{true} \implies \delta(q, i, r) \neq q_\perp \quad (15)$$

$$\forall q \in Q \quad \exists! o \in O \quad \exists! r \in (R \cup \{r_\perp\}) \quad \omega(q) = \text{false} \implies \delta(q, o, r) \neq q_\perp \quad (16)$$

Our goal is to learn an IORA $A = (I, O, R^A, Q^A, q_0, \delta^A, \lambda^A, \tau^A, \pi^A, \omega^A)$ that is consistent with a set of action traces S . Because of the nature of our encoding, we consider each action trace $\sigma = (\sigma^I, \sigma^O)$ in S as an interleaving of the input action string σ^I and the output action string σ^O , i.e. $\sigma = \sigma_1^I \sigma_1^O \dots \sigma_{|\sigma^I|}^I \sigma_{|\sigma^I|}^O$. Let $T = ((I \cup O) \times \mathbb{Z}, Q^T, \lambda^T)$ be an OT for such strings.

The constraints for an IORA can now be constructed in the same way as for an RA (Eqs. 1 and 8–10). Observe that we do not use λ to encode observed outputs (this is already done by encoding the transitions of the OT). Instead, λ is used to denote which locations are allowed. All the locations in Q are allowed (because we have observed them) and q_{\perp} is the only location that is not allowed ($\lambda(q_{\perp}) = \text{false}$ by Eq. 11). As such, we add the following constraint:

$$\forall \sigma \in Q^T \quad \text{map}(\sigma) \neq q_{\perp} \quad (17)$$

We can now determine if there is an IORA with at most n locations and m registers in the same way as for RAs.

3 Implementation and Evaluation

We implemented our encodings using Z3Py, the Python front-end of Z3 [9]². Our tool can generate an automaton model from a given set of observations (passive learning), or a reference to the system and a tester implementation (active learning), also when this system cannot be reset. We have also implemented a tester for the classes of automata supported. The tester generates test queries (or *tests*) each test consisting of an access sequence to an arbitrary state in the current hypothesis, and a sequence generated by a random walk from that state. In experiments, we configure the tester to build shorter tests. Longer tests worsen the scalability of our tool, and are unneeded for learning small models. Validity of the learned models was ensured by running a large number of tests on the last hypothesis and checking the number of states. We conducted a series of experiments to assess the scalability and effectiveness of our approach.

Our first experiment assesses the scalability of our encodings by adapting the scalable *Login*, *FIFO set* and *Stack* benchmarks of [2] to DFAs, Mealy machines, RAs and IORAs. To generate tests, we used the testing algorithm described earlier. The maximum length of the random sequence is $3 + \text{size}$, where *size* is the number of users or elements in the system. The *solver timeout* – the amount of time the solver was provided to compute a solution or indicate its absence – was set to 10 seconds for the DFA and Mealy systems, and to 10 min for the RA and IORA systems whose constraints could take considerably longer to process. Each learning run ends when either a solution is found (indicating success), or when a maximum bound for n is reached (indicating failure). We hence continue incrementing n even after encountering solver timeouts. By doing so, we lose the minimality guarantee yet may learn larger systems. For each system we performed 5 learning runs and collated the resulting statistics.

The results are shown in Table 1. Columns describe the system, the number of successful learning runs, the number of states/locations (which may vary due to loss of minimality) and registers (where applicable), average and standard deviation for the number of tests and inputs used in learning except for validating the last hypothesis, and for the amount of time learning took. The table only

² See <https://gitlab.science.ru.nl/rick/z3gi/tree/lata>.

Table 1. Scalable experiments. The model name encodes the formalism, type and size.

Model	Succ	States loc	Regs	Tests		Inputs		Time (sec)	
				Avg	Std	Avg	Std	Avg	Std
DFA_FIFOSet(8)	5	10.0		228.0	81.11	2156.0	832.02	76.28	42.49
DFA_FIFOSet(9)	2	11.5		413.5	161.93	4194.0	2104.35	199.99	26.1
DFA_Login(3)	5	11.0		446.0	100.18	3092.0	781.73	131.84	39.26
Mealy_FIFOSet(11)	5	12.0		280.0	110.65	3694.0	1722.57	79.75	23.69
Mealy_FIFOSet(12)	4	15.5		370.0	200.12	5021.0	3312.85	227.15	290.99
Mealy_FIFOSet(13)	2	14.5		526.5	318.91	8021.5	5608.06	190.36	51.96
Mealy_Login(3)	5	10.0		104.0	10.03	726.0	69.93	52.4	4.83
Mealy_Login(4)	1	16.0		241.0	0.0	2094.0	0.0	370.19	0.0
RA_Stack(1)	5	3.0	1	32.0	21.18	109.0	90.48	3.18	0.86
RA_Stack(2)	5	5.0	2	202.0	71.88	1018.0	394.58	124.72	53.41
RA_FIFOSet(1)	5	3.0	1	49.0	12.92	180.0	52.56	4.94	6.07
RA_FIFOSet(2)	5	6.0	2	365.0	88.41	2025.0	578.33	333.09	334.12
RA_Login(1)	5	4.0	1	306.0	163.18	1336.0	765.23	54.96	9.99
RA_Login(2)	3	8.0	2	1606.0	345.22	9579.0	2163.67	6258.11	1179.27
IORA_Stack(1)	5	7.0	1	7.0	1.58	24.0	6.63	8.77	1.92
IORA_FIFOSet(1)	5	7.0	1	8.0	3.27	31.0	9.36	6.45	0.84
IORA_Login(1)	5	9.0	1	33.0	6.65	152.0	29.89	1509.18	477.04

includes entries for the largest systems we could learn. A table with entries for all the systems learned is featured in the extended version of this work (See footnote 1).

As part of our second experiment we applied our tool on models learned in three case-studies [3, 4, 22]. These models are Mealy machines detailing aspects of the behavior of bankcard protocols, biometric passports and power control services (PCS). We used the tester described in [23]. This produces tests similar to our own, but extended by distinguishing sequences. These tests are parameterized by the length of the random sequence and a factor k . We set both to 1. The solver timeout is set to 1 min.

We use this experiment to also draw comparison between our approach and learners in the classical framework. To that end, to learn the case study models we also use the TTT [15] and classical L* [5] algorithms. Both of these algorithms are provided by the LearnLib learning tool (v0.12.1) [21]. To draw further comparison we additionally include scalable systems in this experiment. This allows us to compare the SMT-based approach with Tomte (v0.41) [2], a state-of-the-art learning tool for register automata. We note that the test configurations are similar for all learners. Due to the high standard deviation, we ran 20 experiments for each benchmark.

Table 2 shows the results of this experiment. We include the alphabet size and number of states/locations in the model, as well as the average time it took for the SMT approach to learn (this statistic is not useful for the other learners, which took at most a couple of seconds). We remark that the SMT-based approach is able to successfully learn all models every time, though it takes a long time for the larger ones. We also note that increasing the length of the random sequence used in tests even by 1 meant the SMT-based approach occasionally failed to infer the larger VISA model.

Analyzing Table 2, we remark that the SMT-based approach largely outperforms L* and Tomte, while staying competitive with TTT. The approach performs better on scalable systems than on the case study models. This can be attributed to scalable systems requiring more counterexamples. To process these counterexamples, active learning algorithms require additional queries. Performance is further hindered if counterexamples are not optimal (e.g. they are unnecessarily long). Such problems do not affect the SMT-based approach.

Table 2. Comparison with other learners

Model	States loc	Alph size	SMT			TTT		L*		Tomte	
			Tests	Inputs	Time	Tests	Inputs	Tests	Inputs	Tests	Inputs
Biometric passport	6	9	220	1057	26	220	941	333	1143		
MAESTRO	6	14	835	4375	359	860	4437	1190	4718		
MasterCard	6	14	839	4379	353	996	5260	1190	4718		
PIN	6	14	757	3945	338	911	4769	1190	4718		
SecureCode	4	14	313	1485	90	194	682	798	2758		
VISA	9	14	796	4770	2115	750	4094	2040	9015		
PCS_1	8	9	629	3530	189	417	2179	657	2682		
PCS_2	3	9	71	279	9	75	196	252	657		
PCS_3	7	9	508	2651	154	476	2472	576	2196		
PCS_4	7	9	559	3024	154	451	2297	576	2196		
PCS_5	9	9	1120	6260	778	417	1753	1308	5340		
PCS_6	9	9	1158	6442	704	457	1977	1308	5340		
Mealy_FIFOSet(2)	3	2	6	27	0	12	38	14	38		
Mealy_FIFOSet(7)	8	2	52	481	7	71	588	235	2494		
Mealy_FIFOSet(10)	11	2	179	2152	63	163	1822	486	6743		
Mealy_Login(2)	6	3	37	214	7	57	242	57	219		
Mealy_Login(3)	10	3	89	644	64	120	704	240	1720		
IORA_FIFOSet(1)	7	2	9	31	7					21	36.5
IORA_Stack(1)	7	2	8.5	33	8					19	34
IORA_Login(1)	9	3	33	152	849					157	580

Our final experiment assesses our extension for learning systems without resets using benchmarks from recent related work [20]. For the sake of space we

summarize results and refer the reader to the extended version (See footnote 1) for details. Despite the simplicity of the test setup, the SMT-based approach performed only slightly worse than the sophisticated SAT-based approach of [20].

4 Conclusions

We have experimented with an approach for model learning which uses SMT solvers. The approach is highly versatile, as shown in its adaptations for learning FSMs and register automata, and for learning without resets. We provide an open source tool implementing these adaptations. Experiments indicate that our approach is competitive with the state-of-the-art. While the approach does not scale well, we have shown that it can be used for learning small models in practice. In the future we wish to improve the scalability of the approach via more efficient encodings. We hope this paper gives rise to a broader direction of future work, since the presented approach has several advantages over traditional model learning algorithms. Notably, it appears to be quite effective for rapid prototyping of learning algorithms for new formalisms and settings.

References

1. Aarts, F., et al.: Generating models of infinite-state communication protocols using regular inference with abstraction. *FMSD* **46**(1), 1–41 (2015)
2. Aarts, F., Fifteau-Brosteau, P., Kuppens, H., Vaandrager, F.: Learning register automata with fresh value generation. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) *ICTAC 2015*. LNCS, vol. 9399, pp. 165–183. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25150-9_11. <http://www.sws.cs.ru.nl/publications/papers/fvaan/TomteFresh/>
3. Aarts, F., de Ruiter, J., Poll, E.: Formal models of bank cards for free. In: *ICST Workshops*, pp. 461–468. IEEE (2013)
4. Aarts, F., Schmaltz, J., Vaandrager, F.: Inference and abstraction of the biometric passport. In: Margaria, T., Steffen, B. (eds.) *ISoLA 2010*. LNCS, vol. 6415, pp. 673–686. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16558-0_54
5. Angluin, D.: Learning regular sets from queries and counterexamples. *I&C* **75**(2), 87–106 (1987)
6. Angluin, D.: Negative results for equivalence queries. *Mach. Learn.* **5**, 121–150 (1990)
7. Bruynooghe, M., et al.: Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with IDP3. *TPLP* **15**(6), 783–817 (2015)
8. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. *FAOC* **28**(2), 233–263 (2016)
9. De Moura, L., Bjørner, N.: Satisfiability modulo theories: introduction and applications. *CACM* **54**(9), 69–77 (2011)
10. Florêncio, C.C., Verwer, S.: Regular inference as vertex coloring. *TCS* **558**, 18–34 (2014)
11. Gold, E.: Language identification in the limit. *I&C* **10**(5), 447–474 (1967)

12. Heule, M., Verwer, S.: Software model synthesis using satisfiability solvers. *Empir. Softw. Eng.* **18**(4), 825–856 (2013)
13. De la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, Cambridge (2010)
14. Howar, F., Steffen, B., Jonsson, B., Cassel, S.: Inferring canonical register automata. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 251–266. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_17
15. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: a redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 307–322. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_26
16. Lee, D., Yannakakis, M.: Testing finite-state machines: state identification and verification. *IEEE Trans. Comput.* **43**(3), 306–320 (1994)
17. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines—a survey. *Proc. IEEE* **84**(8), 1090–1123 (1996)
18. Neider, D.: Computing minimal separating DFAs and regular invariants using SAT and SMT solvers. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, pp. 354–369. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33386-6_28
19. Peled, D., Vardi, M., Yannakakis, M.: Black box checking. In: FORTE, pp. 225–240. Kluwer (1999)
20. Petrenko, A., Avellaneda, F., Groz, R., Oriat, C.: From passive to active FSM inference via checking sequence construction. In: Yevtushenko, N., Cavalli, A.R., Yenigün, H. (eds.) ICTSS 2017. LNCS, vol. 10533, pp. 126–141. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67549-7_8
21. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. *STTT* **11**(5), 393–407 (2009)
22. Schuts, M., Hooman, J., Vaandrager, F.: Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 311–325. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33693-0_20
23. Smeenk, W., Moerman, J., Vaandrager, F., Jansen, D.N.: Applying automata learning to embedded control software. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 67–83. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25423-4_5
24. Smetsers, R.: Grammatical Inference as a Satisfiability Modulo Theories Problem. arXiv preprint [arXiv:1705.10639](https://arxiv.org/abs/1705.10639) (2017)
25. Vaandrager, F.: Model learning. *CACM* **60**(2), 86–95 (2017)
26. Walkinshaw, N., Derrick, J., Guo, Q.: Iterative refinement of reverse-engineered models by model-based testing. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 305–320. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05089-3_20



Analytic Combinatorics of Lattice Paths with Forbidden Patterns: Enumerative Aspects

Andrei Asinowski¹(✉) , Axel Bacher²(✉) , Cyril Banderier²(✉) ,
and Bernhard Gittenberger¹(✉)

¹ Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien,
Wiedner Hauptstraße 8-10/104, 1040 Vienna, Austria

{andrei.asinowski,gittenberger}@dmg.tuwien.ac.at

² Laboratoire d’Informatique de Paris Nord, Université Paris 13,
99, rue Jean-Baptiste Clément, 93430 Villetaneuse, France
{axel.bacher,cyril.banderier}@lipn.univ-paris13.fr

Abstract. This article presents a powerful method for the enumeration of pattern-avoiding words generated by an automaton or a context-free grammar. It relies on methods of analytic combinatorics, and on a matricial generalization of the kernel method. Due to classical bijections, this also gives the generating functions of many other structures avoiding a pattern (e.g., trees, integer compositions, some permutations, directed lattice paths, and more generally words generated by a push-down automaton). We focus on the important class of languages encoding lattice paths, sometimes called generalized Dyck paths. We extend and refine the study by Banderier and Flajolet in 2002 on lattice paths, and we unify several dozens of articles which investigated patterns like peaks, valleys, humps, etc., in Dyck and Motzkin words. Indeed, we obtain formulas for the generating functions of walks/bridges/meanders/excursions avoiding any fixed word (a *pattern*). We show that the autocorrelation polynomial of this forbidden pattern (as introduced by Guibas and Odlyzko in 1981, in the context of regular expressions) still plays a crucial role for our algebraic functions. We identify a subclass of patterns for which the formulas have a neat form. En passant, our results give the enumeration of some classes of self-avoiding walks, and prove several conjectures from the On-Line Encyclopedia of Integer Sequences. Our approach also opens the door to establish the universal asymptotics and limit laws for the occurrence of patterns in more general algebraic languages.

Keywords: Lattice paths · Pattern avoidance · Finite automata
Autocorrelation · Generating function · Kernel method
Asymptotic analysis

Work funded by the Austrian Science Fund (FWF) project SFB F50 “Algorithmic and Enumerative Combinatorics”.

1 Introduction

Combinatorial structures having a rational or an algebraic generating function play a key role in many fields: computer science (analysis of algorithms involving trees, lists, words), computational geometry (integer points in polytopes, maps, graph decomposition), bioinformatics (RNA structure, pattern matching), number theory (integer compositions, automatic sequences and modular properties, integer solutions of varieties), probability theory (Markov chains, directed random walks), see e.g. [3, 12, 21, 35]. They are often the trace of a structure which has a recursive specification in terms of a system of tree-like structures, or of some functional equation solvable by variants of the kernel method [13].

Since the seminal article by Chomsky and Schützenberger on the link between context-free grammars and algebraic functions [15], which also holds for push-down automata [33], many articles encoded and enumerated combinatorial structures via a formal language approach. See e.g. [19, 25, 30] for such an approach on the so-called *generalized Dyck languages*. These languages are in fact equivalent to directed lattice paths, and in this article, we try to understand how some of these fundamental objects can be enumerated when they have the additional constraint to avoid a given pattern. For sure, such a class of objects can be described as the intersection of a context-free language and a rational language; therefore, classical closure properties imply that they are directly generated by another (but huge and clumsy) context-free language. Unfortunately, despite the fact the algebraic system associated with the corresponding context-free grammar is *in theory* solvable by a resultant computation or by Gröbner bases, this leads *in practice* to equations which are so big that no current computer could handle them in memory, even for generalized Dyck languages with 20 different letters.

In this article, we introduce a generic and efficient way to tackle the question of enumerating words avoiding a given pattern (for languages generated by push-down automata) which bypass these intractable equations. For directed lattice paths, our method allows to handle an arbitrary number of letters (i.e., allowed jumps), up to alphabets of thousands of letters, computationally in a few minutes. It relies on an analytic combinatorics approach, and also on the kernel method, which we used in our investigation of enumerative and asymptotic properties of lattice paths [4–7]. This allows to unify the considerations of many articles which investigated natural patterns like peaks, valleys, humps, etc., in Dyck and Motzkin words, corresponding patterns in trees, compositions..., see e.g. [9, 10, 14, 16–18, 20, 26, 29, 31] and all the examples mentioned in our Sect. 7.

2 Definitions and Notations

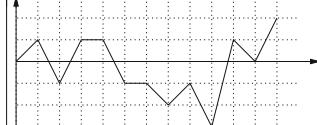
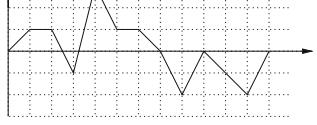
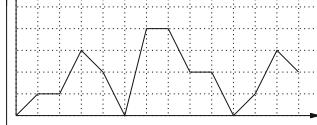
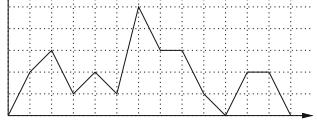
Let \mathcal{S} , the *set of steps* (or *jumps*), be some finite subset of \mathbb{Z} , that contains at least one negative and at least one positive number. A *lattice path with steps from \mathcal{S}* is a finite word $w = [v_1, v_2, \dots, v_n]$ in which all letters belong to \mathcal{S} , visualized as a directed polygonal line in the plane, which starts in the origin and is formed by successive appending of vectors $(1, v_1), (1, v_2), \dots, (1, v_n)$. The letters that form the path $w = [v_1, v_2, \dots, v_n]$ are referred to as its *steps*. The

length of w , to be denoted by $|w|$, is the number of steps in w . The final altitude of w , to be denoted by $h(w)$, is the sum of all steps in w , that is $v_1 + v_2 + \dots + v_n$; visually, it is the y -coordinate of the point where w terminates.

Under this setting, it is usual to consider two restrictions: being the whole path (weakly) above the x -axis, and having final altitude 0 (equivalently, terminating at the x -axis). Consequently, one considers four classes of lattice paths:

1. A *walk* is any path as described above.
2. A *bridge* is a path that terminates at the x -axis.
3. A *meander* is a path that stays (weakly) above the x -axis.
4. An *excursion* is a path that stays (weakly) above the x -axis and also terminates at the x -axis. In other words, an excursion fulfills both restrictions (See Table 1).

Table 1. Summary of our results. For the four types of paths and for any set of jumps encoded by $P(u)$, we give the corresponding generating function of such lattice paths avoiding a pattern p (of length ℓ and final altitude b). The formulas involve the autocorrelation polynomial $R(t, u)$ of p , and the small roots u_i of the kernel $K(t, u) := (1 - tP(u))R(t, u) + t^\ell u^b$.

	ending anywhere	ending at 0
on \mathbb{Z}	 <p>walks</p> $W(t, u) = \frac{R(t, u)}{K(t, u)}$	 <p>bridges</p> $B(t) = \sum_{i=1}^e \frac{u'_i}{u_i} \frac{t}{1 + \frac{R'(t, u_i)}{R(t, u_i)^2} t^{\ell+1} u_i^b - \frac{(\ell-1)t^\ell u_i^b}{R(t, u_i)}}$
on \mathbb{N}	 <p>meanders</p> $M(t, u) = \frac{R(t, u)}{K(t, u)} \prod_{i=1}^c (u - u_i(t))$	 <p>excursions</p> $E(t) = \frac{(-1)^{c+1}}{t} \prod_{i=1}^c u_i(t)$

For each of these classes (when no pattern is forbidden), Banderier and Flajolet [4] gave general expressions for the corresponding generating functions and the asymptotics of their coefficients. In the generating functions, the variable t corresponds to the length of a path, and the variable u to its final altitude. $P(u)$ is the *characteristic polynomial* of the set of steps \mathcal{S} , defined by $P(u) = \sum_{s \in \mathcal{S}} u^s$. The smallest (negative) number in \mathcal{S} is denoted by $-c$, and the largest (positive) number in \mathcal{S} is denoted by d : that is¹, if one orders the terms of $P(u)$ by the powers of u , one has $P(u) = u^{-c} + \dots + u^d$.

¹ Some weights (or probabilities, or multiplicities) could be associated with each jump, but we omit them in this article to keep readability. All the proofs would be similar.

3 Lattice Paths with Forbidden Patterns, and Autocorrelation Polynomial

We consider lattice paths with step set \mathcal{S} that avoid a certain *pattern*, that is, an a priori fixed path $p = [a_1, a_2, \dots, a_\ell]$. To be precise, we define an *occurrence* of p in a lattice path w as a substring of w which coincides with p . If there is no occurrence of p in w , we say that w *avoids* p . For example, the path $[1, 2, 3, 1, 2]$ has two occurrences of $[1, 2]$, but it avoids $[2, 1]$.

Before we state our results, we introduce some notations.

A *presuffix* of p is a non-empty string that occurs in p both as a prefix and as a suffix. In particular, the whole word p is a (trivial) presuffix of itself. If p has one or several non-trivial presuffixes, we say that p exhibits an *autocorrelation* phenomenon. For example, for the pattern $p = [1, 1, 2, 1, 2]$ we have no autocorrelation. In contrast, the pattern $p = [1, 1, 2, 3, 1, 1, 2, 3, 1, 1]$ has three non-trivial presuffixes: $[1]$, $[1, 1]$, and $[1, 1, 2, 3, 1, 1]$, and thus in this case we have autocorrelation.

While analysing the Boyer–Moore string searching algorithm and properties of periodic words, Guibas and Odlyzko introduced in 1981 [22] what turns out to be one of the key characters of our article, the autocorrelation polynomial² of the pattern p : For any given word p , let \mathcal{Q} be the set of its presuffixes; the *autocorrelation polynomial* of p is

$$R(t, u) = \sum_{q \in \mathcal{Q}} t^{|\bar{q}|} u^{h(\bar{q})}, \quad (1)$$

where \bar{q} denotes the complement of q in p .

For example, consider the pattern $p = [1, 1, 2, 3, 1, 1, 2, 3, 1, 1]$. Its four presuffixes produce four terms of $R(t, u)$ as follows:

q	$ \bar{q} $	$h(\bar{q})$
$[1]$	9	15
$[1, 1]$	8	14
$[1, 1, 2, 3, 1, 1]$	4	7
$[1, 1, 2, 3, 1, 1, 2, 3, 1, 1]$	0	0

Therefore, for this p we have $R(t, u) = 1 + t^4 u^7 + t^8 u^{14} + t^9 u^{15}$.

Notice that if for some p no autocorrelation occurs, then we have $\mathcal{Q} = \{p\}$ and therefore $R(t, u) = 1$.

Finally, we define the *kernel* as the following Laurent polynomial:

$$K(t, u) := (1 - tP(u))R(t, u) + t^{|p|} u^{h(p)}. \quad (2)$$

Also in our case it can be shown that each root $u = u(t)$ of $K(t, u) = 0$ is either small or large, and that the number of small roots is $e := \max\{c, -h(p)\}$: we shall denote them by u_1, \dots, u_e .

² A similar notion also appears in the work of Schützenberger on synchronizing words [34].

Now we can state our main results. Recall that t is the variable for the length of a path, and u is the variable for its final altitude.

Theorem 1. *Let \mathcal{S} be a set of steps, and let p be a pattern with steps from \mathcal{S} . Denote $\ell = |p|$, $b = h(p)$.*

1. *The bivariate generating function for walks avoiding the pattern p , is*

$$W(t, u) = \frac{R(t, u)}{K(t, u)}. \quad (3)$$

If one does not keep track of the final altitude, this yields

$$W(t) = W(t, 1) = \frac{1}{1 - tP(1) + t^{\ell}/R(t, 1)}. \quad (4)$$

2. *The generating function for bridges avoiding the pattern p is*

$$B(t) = t \sum_{i=1}^e \frac{u'_i(t)}{u_i(t)} \frac{R(t, u_i(t))}{R(t, u_i(t)) + \frac{(\partial_t R)(t, u_i(t))}{R(t, u_i(t))} t^{\ell+1} u_i(t)^b - (\ell-1) t^{\ell} u_i(t)^b}, \quad (5)$$

where u_1, \dots, u_e are the small roots of the kernel $K(t, u)$, as defined in (2).

Definition. For meanders and excursions, the formulas have a noteworthy shape when the pattern p is a *pseudomeander*, i.e. a lattice path which does not cross the x -axis, except, possibly, at the last step. Notice that if the pattern p is a pseudomeander, then $h(p) \geq -c$, and therefore, the number of small roots of $K(t, u)$ is c .

Theorem 2. *Let \mathcal{S} be a set of steps, and let p be a pseudomeander. Denote $\ell = |p|$, $b = h(p)$.*

1. *The bivariate generating function $M(t, u)$ for meanders avoiding the pattern p is*

$$M(t, u) = \frac{R(t, u)}{u^c K(t, u)} \prod_{i=1}^c (u - u_i(t)), \quad (6)$$

where $u_1(t), \dots, u_c(t)$ are the small roots of $K(t, u) = 0$.

If one does not keep track of the final altitude, this yields

$$M(t) = M(t, 1) = \frac{R(t, 1)}{K(t, 1)} \prod_{i=1}^c (1 - u_i(t)). \quad (7)$$

2. *The generating function for excursions avoiding the pattern p is*

$$E(t) = M(t, 0) = \frac{(-1)^{c+1}}{t} \prod_{i=1}^c u_i(t) \quad (8)$$

if $b > -c$; if $b = c$, then one has $t - t^{\ell}$ rather than t in the denominator.

N.B.: When p is not a pseudomeander, there are still some algebraic expressions, but not as concise.

Remark. Notice that for these four classes of lattice paths, if one forbids a pattern of length 1 or if one uses symbolic weights for each jump, this recovers the formulas from Banderier and Flajolet [4].

4 Automaton and Transfer Matrix A

The following automata, sharing the spirit of the Knuth–Morris–Pratt algorithm, will allow us to tackle pattern avoidance. Let $p = [a_1, \dots, a_\ell]$ be the “forbidden” pattern. As we construct a lattice path $w = [v_1, v_2, \dots]$ step by step, it might start “accumulating” p , that is, contain some prefix of p . We introduce ℓ many states that indicate how much of p has the path w accumulated at each step. These states, $X_0, X_1, \dots, X_{\ell-1}$, will be labelled by proper prefixes of p : $X_i = [a_1, a_2, \dots, a_i]$ (in particular, the first state is labelled by the empty word: $X_0 = \epsilon = []$). We say that $w_{1..m} = [v_1, v_2, \dots, v_m]$, a prefix of w , is in the state X_i (or, alternatively: w , after its m th step, is in the state X_i), if the label of X_i is the longest proper prefix of p that coincides with a suffix of $w_{1..m}$: $v_{m-i+1} = a_1, v_{m-i+2} = a_2, \dots, v_m = a_i$.

If w is in the state X_i after certain step v_m , then its state after the next step v_{m+1} is uniquely determined by i and v_{m+1} (unless $i = \ell-1$ and $v_{m+1} = a_\ell$ which is impossible since this would yield an occurrence of the forbidden pattern). This gives a finite automaton completely determined by $P(u)$ and p . Its states are labelled by $X_0, \dots, X_{\ell-1}$, and for $i, j \in \{0, \dots, \ell-1\}$ we have an arrow labelled λ from X_i to X_j if j is the maximum number such that X_j is a suffix of $X_i\lambda$. Its transition matrix will be denoted by A : it is an $\ell \times \ell$ matrix, and its (i, j) entry is the sum of all terms u^λ such that there is an arrow labelled λ from X_{i-1} to X_{j-1} . See Fig. 1 for an example.

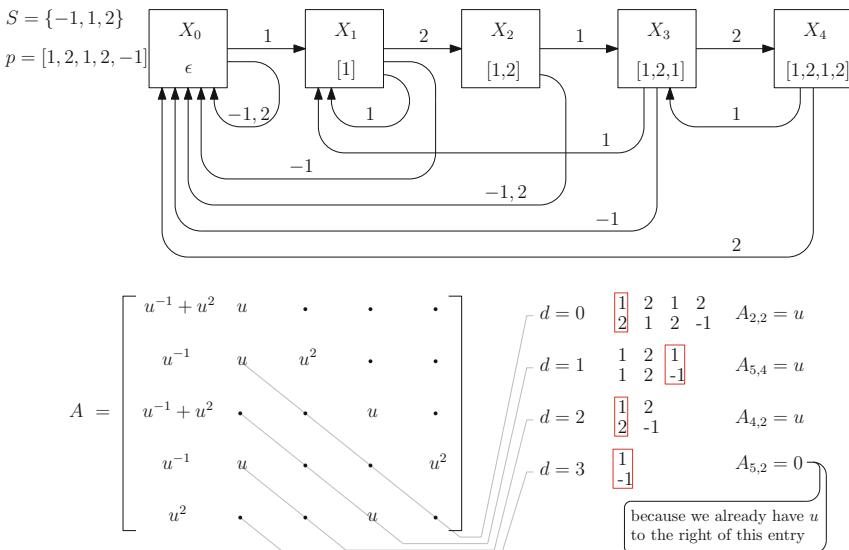


Fig. 1. The automaton and the transfer matrix A for $S = \{-1, 1, 2\}$ and the pattern $p = [1, 2, 1, 2, -1]$. (The 0 entries of A are replaced by dots.)

The matrix A has several general properties. In particular, for all i, j such that $j > i + 1$, we have $A_{i,j} = 0$; for each i , $1 \leq i \leq \ell - 1$, we have $A_{i,i+1} = a_i$; for each j , $2 \leq j \leq \ell$, every entry in the j th column is either 0 or a_{j-1} ; for each i , $1 \leq i \leq \ell - 1$, the sum of the entries in the i th row is $P(u)$; the sum of the entries in the ℓ -th row is $P(u) - a_\ell$.

The “essential” entries of A are those with $2 \leq j \leq i$. They can be determined by the following procedure, which is also illustrated in Fig. 1. For $d = 0, 1, \dots, \ell - 2$ we compare $[a_1, a_2, \dots, a_{\ell-d-1}]$ with $[a_{d+2}, a_{d+3}, \dots, a_\ell]$. If they coincide ($a_\beta = a_{d+1+\beta}$ for all $\beta = 1, 2, \dots, \ell-d-1$), then all the entries $A_{i,j}$ with $i-j = d$, $j \geq 2$, are 0. Otherwise, if β is the smallest number such that $a_\beta \neq a_{d+1+\beta}$, then $A_{d+\beta+1,\beta+1} = u^\beta$, unless a smaller d yielded u^β in the same row, to the right of this position (if this happens, $A_{d+\beta+1,\beta+1} = 0$).

5 The Structure of the Kernel: $\det(I - tA)$

In what follows, an important role will be played by the matrix $I - tA$, specifically by its determinant and by the sum of elements in the first row of its adjoint. In particular the role of $\det(I - tA)$ in our study is the analog of the role played by $1 - tP(u)$ in the study of Banderier and Flajolet [4], but our equation is more involved.

Theorem 3. *Let \mathcal{S} be a set of steps, and let p be a pattern with steps from \mathcal{S} . Then for A , the transfer matrix of the automaton, we have*

$$\det(I - tA) = K(t, u) = (1 - tP(u))R(t, u) + t^{|p|}u^{h(p)}, \quad (9)$$

$$(1 \ 0 \ \cdots \ 0) \text{adj}(I - tA) (1 \ 1 \ \cdots \ 1)^\top = R(t, u), \quad (10)$$

where $K(t, u)$ and $R(t, u)$ are the kernel and the autocorrelation polynomial, as defined in Eqs. (1) and (2). In particular, in the case without autocorrelation we have $\det(I - tA) = 1 - tP(u) + t^{|p|}u^{h(p)}$, and the sum of the entries in the first row of $\text{adj}(I - tA)$ is 1.

Proof (sketch). Consider first that every step s of S has a symbolic weight t_s . Let W be the generating function of walks avoiding p with these weights, R the autocorrelation polynomial of p , $P = \sum_s t_s$ the polynomial encoding the different steps s , and t_p the weight of p . Using the construction of [21, p. 60], we find that: $W = \frac{R(t, u)}{(1 - P(u))R(t, u) + t_p}$. By Cramer’s rule, W is also equal to the left-hand side of (10) divided by the left-hand side of (9). Since $(1 - P(u))R(t, u) + t_p$ is an irreducible polynomial with degree ℓ , the two rational representations of W are identical. We conclude by specializing t_s to $tu^{h(s)}$ for all s . \square

6 Proofs of the Generating Functions for Walks, Bridges, Meanders, and Excursions

Proof of Theorem 1 for walks. Let $W(t, u)$ be the bivariate generating function for the number of walks with steps from \mathcal{S} avoiding a fixed pattern p . For $\alpha = 1, 2, \dots, \ell$, we denote by $W_\alpha = W_\alpha(t, u)$ the corresponding bivariate generating function restricted to those walks that terminate in state α . Then we have the following vectorial functional equation:

$$(W_1 \ W_2 \ \cdots \ W_\ell) = (1 \ 0 \ \cdots \ 0) + t (W_1 \ W_2 \ \cdots \ W_\ell) A,$$

$$(W_1 \ W_2 \ \cdots \ W_\ell) (I - tA) = (1 \ 0 \ \cdots \ 0),$$

$$(W_1 \ W_2 \ \cdots \ W_\ell) = (1 \ 0 \ \cdots \ 0) \frac{\text{adj}(I - tA)}{|I - tA|}.$$

Therefore, the generating function for $W(t, u)$, which is the sum of the generating functions $W_\alpha(t, u)$ over all states, is equal to

$$\begin{aligned} W(t, u) &= (W_1 \ W_2 \ \cdots \ W_\ell) (1 \ 1 \ \cdots \ 1)^\top \\ &= \frac{(1 \ 0 \ \cdots \ 0) \text{adj}(I - tA) (1 \ 1 \ \cdots \ 1)^\top}{|I - tA|} = \frac{R(t, u)}{(1 - tP(u))R(t, u) + t^{|p|}u^{h(p)}}, \end{aligned}$$

where the last equality follows from Theorem 3. \square

Proof of Theorem 1 for bridges. In order to find the univariate generating function $B(t)$ for bridges, we need to extract the coefficient of $[u^0]$ from $W(t, u)$. To this end, we assume that t is a sufficiently small fixed number, extract the coefficient of a (univariate) function by means of Cauchy's integral formula, and apply the residue theorem:

$$B(t) = [u^0]W(t, u) = \frac{1}{2\pi i} \int_{|u|=\varepsilon} \frac{W(t, u)}{u} du = \sum_i^e \text{Res}_{u=u_i(t)} \frac{W(t, u)}{u},$$

where u_1, \dots, u_e are the small roots of $K(t, u)$. By the formula for residues of rational functions, we have

$$\begin{aligned} \text{Res}_{u=u_i(t)} \frac{W(t, u)}{u} &= \text{Res}_{u=u_i(t)} \frac{R(t, u)}{u((1 - tP(u))R(t, u) + t^\ell u^b)} \\ &= \left. \frac{R(t, u)}{\frac{d}{du}(u((1 - tP(u))R(t, u) + t^\ell u^b))} \right|_{u=u_i(t)}. \end{aligned}$$

The denominator of this expression is

$$-tuP'(u)R(t, u) + u(1 - tP(u))R_u(t, u) + bt^\ell u^b \Big|_{u=u_i(t)}. \quad (11)$$

Next, we differentiate $K(t, u_i) = 0$ with respect to t and obtain an expression for $P'(u_i(t))$. When we substitute it to (11), we obtain (5). \square

Proof of Theorem 2 for meanders and excursions. Similarly to our notation above, we denote by $M(t, u)$ the bivariate generating function for meanders, and by $M_\alpha(t, u)$ the bivariate generating function for meanders that terminate in state α . Then we have the following vectorial functional equation:

$$(M_1 \ M_2 \ \cdots \ M_\ell) = (1 \ 0 \ \cdots \ 0) + t \ (M_1 \ M_2 \ \cdots \ M_\ell) \ A - t [u^{<0}] \\ ((M_1 \ M_2 \ \cdots \ M_\ell) \ A),$$

where $[u^{<0}]$ denotes all the terms in which the power of u is negative. The first component of $[u^{<0}]((M_1 \ M_2 \ \cdots \ M_\ell) \ A)$ is a sum of several fractions of the form $M_i^j(t, u)/u^\gamma$, where $M_i^j(t, u)$ is the generating function for p -avoiding meanders that terminate in state i at altitude j , and $1 \leq \gamma \leq c$. We denote this expression by $F(t, u)/u^c$, where $F(t, u)$ is polynomial in t and u . All other components of $[u^{<0}]((M_1 \ M_2 \ \cdots \ M_\ell) \ A)$ are 0 because if the path arrives at state X_i with $i > 1$ this means that it accumulated a non-empty prefix of p . And since p is a pseudomeander, w will always remain (weakly) above the x -axis while it accumulates its non-empty prefix. Thus we obtain

$$(M_1 \ M_2 \ \cdots \ M_\ell) \ (I - tA) = \left(1 - \frac{t}{u^c} F(t, u)\right) (1 \ 0 \ \cdots \ 0), \quad (12)$$

$$(M_1 \ M_2 \ \cdots \ M_\ell) = \left(1 - \frac{t}{u^c} F(t, u)\right) (1 \ 0 \ \cdots \ 0) \frac{\text{adj}(I - tA)}{|I - tA|}. \quad (13)$$

Finally, we multiply it the last identity by $(1 \ 1 \ \cdots \ 1)$ from the right and obtain, through the use of Theorem 3,

$$M(t, u) = \left(1 - \frac{t}{u^c} F(t, u)\right) \frac{R(t, u)}{K(t, u)}. \quad (14)$$

In order to determine $1 - tF(t, u)/u^c$, we proceed as follows. First, the kernel $K(t, u)$ has precisely c small roots. This follows from the fact that the lowest degree of u in $K(t, u)$ is $-c$: it is contributed by u^{-c} in $P(u)$, while the lowest degree of u in $t^\ell u^b$ is at least $-c$; and this these terms cannot cancel out (the only exception is the trivial case of a one-letter pattern); then it can be shown (for example, by the Newton polygon method) that $K(t, u)$ has c many (distinct) roots whose Puiseux series starts with $\text{const} \cdot u^{1/c}$. Further, it can be show in a similar way that all other roots of $K(t, u)$ are large. As usual, we denote the small roots by u_1, \dots, u_c .

For each $1 \leq i \leq c$, we substitute $u = u_i$ into (12). Then the matrix $I - tA$ is singular and therefore it has a (right) eigenvector \mathbf{v} that belongs to the eigenvalue 0. Moreover, the first component of \mathbf{v} is not 0: otherwise the columns of $I - tA$, with the first column excluded, are linearly dependent, which contradicts the structure of the matrix. Then comparing the first components yields the identity $1 - \frac{t}{u_i^c} F(t, u) = 0$, which means that each u_i , $i = 1, \dots, c$, satisfies $u^c - tF(t, u) = 0$, which is a polynomial equation of degree c . This implies $u^c - tF(t, u) = (u - u_1)(u - u_2) \dots (u - u_c)$. We substitute this into (14) and finally obtain the claimed result (6).

Setting $u = 0$ in Formula (6) gives the formula for excursions. \square

7 Examples

We end our article with a small set of examples, see Table 2. We can also link some self-avoiding walks and pattern avoiding lattice paths. In fact, the enumeration and the asymptotics of self-avoiding walks in \mathbb{Z}^2 is one of the famous open problems of combinatorics and probability theory. As it is classical for intractable problems, many natural subclasses have been introduced, and solved. Our lattice paths avoiding some pattern allow to enumerate many of these subclasses of self-avoiding walks, like partially directed self-avoiding walks with an added constraint of living in a half-plane or a strip [2]. Partially directed walks have three kinds of steps, say n , e and s , and the self-avoiding condition means that factors ns and sn are disallowed. Consider the following three models:

- in the first model, the half-plane is the one over the line $x = 0$; the heights of the steps are $h(n) = 1$, $h(e) = 0$ and $h(s) = -1$;
- in the second model, the half-plane is the one over the line $x = y$; the heights are $h(n) = 1$, $h(e) = -1$ and $h(s) = -1$;
- in the third model, the half-plane is the one over the line $x = -y$; the heights are $h(n) = 1$, $h(e) = 1$ and $h(s) = -1$.

These models are illustrated in Fig. 2 on next page. Each of them leads to an algebraic generating function, compatible with our formulas.

Table 2. Our results provide a unified approach for the computation of generating functions for different models. In particular, this solves several conjectures in the On-Line Encyclopedia of Integer Sequences, it also allows to produce many formulas: this recovers several earlier works, often related to Dyck/Motzkin paths.

Steps, pattern, model	Generating function	OEIS reference ^a
$S = \{-1, 0, 1\}$ $p = [1, 0, \dots, 0, -1]$ bridges	$\frac{1}{\sqrt{1-2t-3t^2+2t^\ell-2t^{\ell+1}+t^{2\ell}}}$	$\ell = 2$: OEIS A051286 (Proving a claim therein on Whitney numbers, see also [11])
$S = \{-1, 0, 1\}$ $p = [1, 0, \dots, 0, -1]$ meanders	$\frac{1-3t+t^\ell-\sqrt{1-2t-3t^2+2t^\ell-2t^{\ell+1}+t^{2\ell}}}{2t(1-3t+t^\ell)}$	$\ell = 2$: OEIS A091964 (RNA folding, see [23])
$S = \{-1, 0, 1\}$ $p = [1, 0, \dots, 0, -1]$ excursions	$\frac{1-t+t^\ell-\sqrt{1-2t-3t^2+2t^\ell-2t^{\ell+1}+t^{2\ell}}}{2t^2}$	$\ell = 2$: OEIS A004148 [24] $\ell = 3$: OEIS A114584 [27]
$S = \{1, -1\}$ $p = [1, -1, 1, -1, \dots, 1]$ excursions	$\frac{1-t^{\ell+1}-\sqrt{1-4t^2+2t^{\ell+1}+4t^{\ell+3}-3t^{2\ell+2}}}{2t^2(1-t^{\ell-1})}$	$\ell = 3$: \approx OEIS A001006 (Motzkin numbers [32, 36])
$S = \{1, -1\}$ $p = [1, -1, 1, -1, \dots, -1]$ excursions	$\frac{1-t^{\ell+2}-\sqrt{1-4t^2+6t^{\ell+2}-4t^{2\ell+2}+t^{2\ell+4}}}{2t^2(1-t^\ell)}$	$\ell = 4$: OEIS A078481 (Irreducible stack sortable permutations, [8, 28])

^aSuch references are links to the webpage dedicated to the corresponding sequence in the On-Line Encyclopedia of Integer Sequences, <http://oeis.org>.

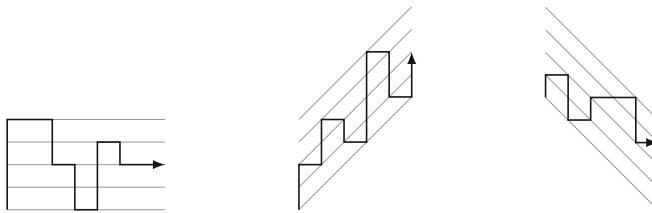


Fig. 2. Some models of self-avoiding walks are encoded by partially directed lattice paths avoiding a pattern (see [2]).

8 Conclusion and Extensions

In this article, we showed how a vectorial generalization of the kernel method allows to enumerate lattice paths avoiding a given pattern. Our approach is flexible, and our future researches include the case of several patterns at once and the general study (enumeration, limit laws) of counting the *number of occurrences* of a given pattern in algebraic structures. The asymptotics are interestingly much more involved than in [4], we analyse these aspects in our companion article [1].

References

1. Asinowski, A., Bacher, A., Banderier, C., Gittenberger, B.: Analytic combinatorics of lattice paths with forbidden patterns: asymptotic aspects. In preparation (2017)
2. Bacher, A., Bousquet-Mélou, M.: Weakly directed self-avoiding walks. *J. Comb. Theory Ser. A* **118**(8), 2365–2391 (2011)
3. Banderier, C., Drmota, M.: Formulae and asymptotics for coefficients of algebraic functions. *Comb. Probab. Comput.* **24**(1), 1–53 (2015)
4. Banderier, C., Flajolet, P.: Basic analytic combinatorics of directed lattice paths. *Theoret. Comput. Sci.* **281**(1–2), 37–80 (2002)
5. Banderier, C., Gittenberger, B.: Analytic combinatorics of lattice paths: enumeration and asymptotics for the area. *Discrete Math. Theor. Comput. Sci. Proc. AG*: 345–355 (2006)
6. Banderier, C., Nicodème, P.: Bounded discrete walks. *Discrete Math. Theor. Comput. Sci. AM*: 35–48, 2010
7. Banderier, C., Wallner, M.: The kernel method for lattice paths below a rational slope. In: *Lattice Paths Combinatorics And Applications, Developments in Mathematics Series*, pp. 1–36. Springer (2018)
8. Baril, J.-L.: Avoiding patterns in irreducible permutations. *Discret. Math. Theor. Comput. Sci.* **17**(3), 13–30 (2016)
9. Baril, J.-L., Petrossian, A.: Equivalence classes of Motzkin paths modulo a pattern of length at most two. *J. Integer Seq.* **18**(11), 1–17 (2015). Article no. 15.7.1
10. Bernini, A., Ferrari, L., Pinzani, R., West, J.: Pattern-avoiding Dyck paths. *Discrete Math. Theor. Comput. Sci. Proc.* 683–694 (2013)
11. Bóna, M., Knopfmacher, A.: On the probability that certain compositions have the same number of parts. *Ann. Comb.* **14**(3), 291–306 (2010)
12. Bousquet-Mélou, M.: Rational and algebraic series in combinatorial enumeration. In: *International Congress of Mathematicians, vol. III*, pp. 789–826. EMS (2006)
13. Bousquet-Mélou, M., Jehanne, A.: Polynomial equations with one catalytic variable, algebraic series and map enumeration. *J. Comb. Theory Ser. B* **96**(5), 623–672 (2006)

14. Brennan, C., Mayhunhu, S.: Visits to level r by Dyck paths. *Fund. Inform.* **117**(1–4), 127–145 (2012)
15. Chomsky, N., Schützenberger, M.-P.: The algebraic theory of context-free languages. In: Computer Programming and Formal Systems, pp. 118–161, North-Holland, Amsterdam (1963)
16. Dershowitz, N., Zaks, S.: More patterns in trees: up and down, young and old, odd and even. *SIAM J. Discret. Math.* **23**(1), 447–465 (2008/2009)
17. Deutsch, E., Shapiro, L.W.: A bijection between ordered trees and 2-Motzkin paths and its many consequences. *Discret. Math.* **256**(3), 655–670 (2002)
18. Ding, Y., Du, R.R.X.: Counting humps in Motzkin paths. *Discret. Appl. Math.* **160**(1–2), 187–191 (2012)
19. Duchon, P.: On the enumeration and generation of generalized Dyck words. *Discret. Math.* **225**(1–3), 121–135 (2000)
20. Eu, S.-P., Liu, S.-C., Yeh, Y.-N.: Dyck paths with peaks avoiding or restricted to a given set. *Stud. Appl. Math.* **111**(4), 453–465 (2003)
21. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press, Cambridge (2009)
22. Guibas, L.J., Odlyzko, A.M.: String overlaps, pattern matching, and nontransitive games. *J. Combin. Theory Ser. A* **30**(2), 183–208 (1981)
23. Hofacker, I.L., Reidys, C.M., Stadler, P.F.: Symmetric circular matchings and RNA folding. *Discret. Math.* **312**(1), 100–112 (2012)
24. Jin, E.Y., Reidys, C.M.: Asymptotic enumeration of RNA structures with pseudo-knots. *Bull. Math. Biol.* **70**(4), 951–970 (2008)
25. Labelle, J., Yeh, Y.N.: Generalized Dyck paths. *Discret. Math.* **82**(1), 1–6 (1990)
26. Manes, K., Sapounakis, A., Tasoulas, I., Tsikouras, P.: Strings of length 3 in Grand-Dyck paths and the Chung-Feller property. *Electron. J. Combin.* **19**(2), 10 (2012). Paper 2
27. Manes, K., Sapounakis, A., Tasoulas, I., Tsikouras, P.: Equivalence classes of ballot paths modulo strings of length 2 and 3. *Discret. Math.* **339**(10), 2557–2572 (2016)
28. Mansour, T.: Statistics on Dyck paths. *J. Integer Seq.* **9**, 1–17 (2006). Article no. 06.1.5
29. Mansour, T., Shattuck, M.: Counting humps and peaks in generalized Motzkin paths. *Discret. Appl. Math.* **161**(13–14), 2213–2216 (2013)
30. Merlini, D., Rogers, D.G., Sprugnoli, R., Verri, M.C.: Underdiagonal lattice paths with unrestricted steps. *Discret. Appl. Math.* **91**(1–3), 197–213 (1999)
31. Park, Y., Park, S.K.: Enumeration of generalized lattice paths by string types, peaks, and ascents. *Discret. Math.* **339**(11), 2652–2659 (2016)
32. Righi, C.: Number of “*udu*’s of a Dyck path and ad-nilpotent ideals of parabolic subalgebras of $sl_{l+1}(\mathbb{C})$. *Sém. Lothar. Combin.* **59**, 17 (2007/2010). Article no. B59c
33. Schützenberger, M.-P.: On context-free languages and push-down automata. *Inf. Control* **6**, 246–264 (1963)
34. Schützenberger, M.-P.: On the synchronizing properties of certain prefix codes. *Inf. Control* **7**, 23–36 (1964)
35. Stanley, R.P.: Enumerative Combinatorics: Volume 1. Cambridge Studies in Advanced Mathematics, vol. 49, 2nd edn. Cambridge University Press, Cambridge (2011)
36. Sun, Y.: The statistic “number of *udu*’s” in Dyck paths. *Discret. Math.* **287**(1–3), 177–186 (2004)



Bubble-Flip—A New Generation Algorithm for Prefix Normal Words

Ferdinando Cicalese , Zsuzsanna Lipták , and Massimiliano Rossi

Dipartimento di Informatica, University of Verona,
Strada le Grazie, 15, 37134 Verona, Italy

{ferdinando.cicalese,zsuzsanna.liptak,massimiliano.rossi.01}@univr.it

Abstract. We present a new recursive generation algorithm for prefix normal words. These are binary words with the property that no factor has more 1s than the prefix of the same length. The new algorithm uses two operations on binary words, which exploit certain properties of prefix normal words in a smart way. We introduce infinite prefix normal words and show that one of the operations used by the algorithm, if applied repeatedly to extend the word, produces an ultimately periodic infinite word, which is prefix normal and whose period's length and density we can predict from the original word.

Keywords: Algorithms on automata and words
Combinatorics on words · Combinatorial generation
Prefix normal words · Infinite words · Binary languages

1 Introduction

Prefix normal words are binary words with the property that no factor has more 1s than the prefix of the same length. For example, 11001010 is prefix normal, but 11001101 is not, since the factor 1101 has too many 1s. These words were introduced in [10], originally motivated by the problem of Jumbled Pattern Matching [1, 2, 4, 8, 9, 11, 12, 15, 17].

Prefix normal words have however proved to have diverse other connections [5–7]. Among these, it has been shown that prefix normal words form a *bubble language* [18–20], a family of binary languages which include Lyndon words, k -ary Dyck words, necklaces, and other important classes of binary words. These languages have efficient generation algorithms¹, and can be listed as Gray codes, i.e. listings in which successive words differ by a constant number of operations. More recently, connections of the language of prefix normal words to the Binary Reflected Gray Code have been discovered [21], and prefix normal words have proved to be applicable to certain graph problems [3].

¹ Here, the term *efficient* is used in the sense that the cost per output word should be small—in the best case, this cost is constant amortized time (CAT).

In this paper, we present a new recursive generation algorithm for prefix normal words of fixed length. In combinatorial generation, the aim is to find a way of efficiently listing (but not necessarily outputting) each one of a given class of combinatorial objects. Often it is necessary to examine each one of such objects, even though their number may be very large, typically exponential. The latest volume 4 A of Donald Knuth's *The Art of Computer Programming* devotes over 200 pages to combinatorial generation of basic combinatorial patterns [14], such as permutations and bitstrings, and much more is planned on the topic [13].

The previous generation algorithm for prefix normal words of length n runs in amortized linear time per word [6], while it was conjectured there that its running time is actually amortized $O(\log n)$ per word, a conjecture which is still open. Our new algorithm recursively generates all prefix normal words from a seed word, applying two operations, which we call *bubble* and *flip*. Our new algorithm's running time is $O(n)$ per word, and it allows new insights into properties of prefix normal words. It can be applied (a) to produce all prefix normal words of fixed length, or (b) to produce all prefix normal words of fixed length sharing the same *critical prefix*. (The critical prefix of a binary word is the first run of 1s followed by the first run of 0s.) This could help proving a conjecture formulated in [6], namely that the expected critical prefix length of an n -length prefix normal word is $O(\log n)$. Moreover, it could prove useful in counting prefix normal words of fixed length: it is easy to see that this number grows exponentially, however, neither a closed form nor a generating function are known [7].

In the second part of the paper, we prove some surprising results about extending prefix normal words. Note that if w is prefix normal, then so is $w0$, but not necessarily $w1$. We introduce infinite prefix normal words and show that repeatedly applying the flip-operation used by the new algorithm produces an ultimately periodic infinite word. Moreover, we are able to predict both the length and the density of the period, and give an upper bound on when the period will appear. Due to space limitations, some proofs have been omitted.

2 Basics

A (finite) binary word (or string) w is a finite sequence of elements from $\{0, 1\}$. We denote the i 'th character of w by w_i , and its length by $|w|$. Note that we index words from 1. The empty word, denoted ε , is the unique word with length 0. The set of binary words of length n is denoted $\{0, 1\}^n$ and the set of all finite words by $\{0, 1\}^*$.² For two words u, v , we write $w = uv$ for their concatenation. For an integer $k \geq 1$ and $u \in \{0, 1\}^n$, u^k denotes the $k \cdot n$ -length word $uuu \cdots u$ (k -fold concatenation of u). If $w = uxv$, with $u, x, v \in \{0, 1\}^*$ (possibly empty), then u is called a prefix, x a factor (or substring), and v a suffix of w . We denote by $w_i \cdots w_j$, for $i \leq j$, the factor of w spanning the positions i through j . For a word u , we write $|u|_1$ for the number of 1s in u . We denote by \leq_{lex} the lexicographic order between words.

² In Sect. 4, we deal with infinite binary words, and give appropriate definitions there.

We denote by $\text{pref}_i(w)$ the prefix of w of length i , and by $P_w(i) = |\text{pref}_i(w)|_1$, the number of 1s in the prefix of length i . (In succinct indexing, this function is often called $\text{rank}_1(w, i)$.) If clear from the context, we write $P(i)$ for $P_w(i)$.

Definition 1 (Prefix normal words, prefix normal condition). A word $w \in \{0, 1\}^*$ is called prefix normal if, for all factors u of w , $|u|_1 \leq P_w(|u|)$. We denote the set of all finite prefix normal words by \mathcal{L} , and the set of prefix normal words of length n by \mathcal{L}_n . Given a binary word w , we say that a factor u of w satisfies the prefix normal condition if $|u|_1 \leq P_w(|u|)$.

Example 1. The word 1101000100110100 is not prefix normal because the factor 1001101 violates the prefix normal condition.

Fact 2 ([7]: Basic facts about prefix normal words) Let $w \in \{0, 1\}^n$.

- (i) If $w \in \mathcal{L}$, then either $w = 0^n$ or $w_1 = 1$.
- (ii) $w \in \mathcal{L}$ if and only if $\text{pref}_i(w) \in \mathcal{L}$ for $i = 1, \dots, n$.
- (iii) If $w \in \mathcal{L}$ then $w0^i \in \mathcal{L}$ for all $i = 1, 2, \dots$.
- (iv) Let $w \in \mathcal{L}$. Then $w1 \in \mathcal{L}$ if and only if for all $1 \leq i < n$, we have $P_w(i+1) > |w_{n-i+1} \dots w_n|_1$.

Definition 3 (Critical prefix). Given a non-empty binary word w , it can be uniquely written in the form $w = 1^s 0^t \gamma$, where $s, t \geq 0$, $s = 0$ implies $t > 0$, and $\gamma \in 1\{0, 1\}^* \cup \{\varepsilon\}$. We refer to $1^s 0^t$ as the critical prefix of w .

We will define several operations on binary words in this paper. For an operation $\text{op} : \{0, 1\}^* \rightarrow \{0, 1\}^*$, we denote by $\text{op}^{(i)}$ the i 'th iteration of op . We denote by $\text{op}^*(w) = \{\text{op}^{(i)}(w) \mid i \geq 1\}$, the set of words obtainable from w by a finite number of applications of op .

3 The Bubble-Flip Algorithm

In this section we present our new generation algorithm for all prefix normal words of a given length. We show that the words are generated in lexicographic order. We also show how our procedure can be easily adapted to generate all prefix normal words of a given length with the same critical prefix.

3.1 The Algorithm

Let $w \in \{0, 1\}^n$. We let $\text{RightmostOne}(w)$ be the largest index r such that $w_r = 1$, if it exists, and ∞ otherwise. We will use the following operations on prefix normal words:

Definition 4 (Operation flip). Given $w \in \{0, 1\}^n$, and $1 \leq j \leq n$, we define $\text{flip}(w, j)$ to be the binary word obtained by changing the j -th character in w , i.e., $\text{flip}(w, j) = w_1 w_2 \dots w_{j-1} \bar{w}_j w_{j+1} \dots w_n$, where \bar{x} is $1 - x$.

Definition 5 (Operation bubble). Given $w \in \{0, 1\}^n \setminus \{0^n\}$ and $r = \text{RightmostOne}(w) < n$, we define $\text{bubble}(w) = w_1 w_2 \cdots w_{r-1} 0 1 0^{n-r-1}$, i.e., the word obtained from w by shifting the rightmost 1 one position to the right.

We start by giving a simple characterization of those flip-operations which preserve prefix normality.

Lemma 6. Let $w \in \mathcal{L}_n$ such that $r = \text{RightmostOne}(w) < n$ and let j be an index with $r < j \leq n$. Then $\text{flip}(w, j)$ is not prefix normal if and only if there exists $1 \leq k < r$ such that for $\beta = w_{r-k+1} \dots w_r$, it holds $|\beta|_1 = P_w(k)$ and $|w_{k+1} \dots w_{k+j-r}|_1 = 0$.

Proof. First note that $\text{flip}(w, j) \in \mathcal{L}$ if and only if $v = \text{pref}_j(\text{flip}(w, j)) \in \mathcal{L}$, by Fact 2 (ii) and (iii). Moreover, $v \in \mathcal{L}$ if and only if for every suffix u of $w_1 \dots w_{j-1}$, it holds that $|u|_1 < P_w(|u| + 1)$ (which we refer to as $(*)$ -condition), again by Fact 2 (iv). We now only have to show that having a suffix u which violates the $(*)$ -condition is equivalent to the right side of the lemma's statement.

Now let u be a suffix of $w_1 \dots w_{j-1}$. If $|u| < r - j - 1$, then $|u|_1 = 0$, and since w is prefix normal and contains at least one 1, u cannot violate the $(*)$ -condition. Thus assume that u spans over position r , the position of the last 1 in w . We can write u as $u = \beta 0^{j-r-1}$, and clearly we have $|\beta|_1 = |u|_1$. Denote $|\beta| = k$. Since w is prefix normal, we have $P_w(k) \geq |\beta|_1 = |u|_1$. Thus, if $|u|_1 = P_w(|u| + 1)$, this, together with the monotonicity of P_w , implies that $P_w(k) = |\beta|_1$. Moreover, then we also have $P_w(|u| + 1) = P_w(k)$, which in turn implies that the prefix of length k is followed by $|u| + 1 - k = j - r$ many 0s.

Conversely, if exists $\beta = w_{r-k+1} \dots w_r$ with $|\beta|_1 = P_w(k)$, and $w_{k+1} \dots w_{k+j-r}$ consists only of 0s, then write $u = \beta 0^{r-j-1}$. We then have $|u|_1 = |\beta|_1 = P_w(k) = P_w(k + j - r) = P(|u| + 1)$, in violation of the $(*)$ -condition. \square

Definition 7 (Phi). Let $w \in \mathcal{L}_n \setminus \{0^n\}$. Let $r = \text{RightmostOne}(w)$. Define $\varphi(w)$ as the minimum j such that $r < j \leq n$ and $\text{flip}(w, j)$ is prefix normal, and $\varphi(w) = n + 1$ if no such j exists.

Example 2. For the word $w = 1101001001011000$, we have $\varphi(w) = 16$, since the words $\text{flip}(w, 14)$ and $\text{flip}(w, 15)$ both violate the prefix normal condition, for the prefixes of length 3 and 6, respectively.

Lemma 8. Let $w \in \mathcal{L}_n \setminus \{0^n\}$ and $r = \text{RightmostOne}(w)$. Let m be the maximum length of a run of zeros following a prefix of $w_1 \dots w_{r-1}$ which has the same number of 1s as the suffix of $w_1 \dots w_r$ of the same length. Formally, $m = \max\{j - k \mid 1 \leq k < j < r, |w_1 \dots w_k|_1 = |w_{r-k+1} \dots w_r|_1 \text{ and } |w_{k+1} \dots w_j|_1 = 0\}$ (where we set the maximum of the empty set to 0). Then, $\varphi(w) = \min(r + m + 1, n + 1)$.

Proof. We first show that $\varphi(w) \leq r + m + 1$. Skipping trivial cases, we can assume that $m < n - r$, for otherwise the desired inequality holds by definition.

Algorithm 1. COMPUTE φ

Given a prefix normal word w , computes the leftmost index j , after the rightmost 1 of w , such that $\text{flip}(w, j)$ is prefix normal

```

1  $r \leftarrow \text{RightmostOne}(w)$ ,  $f \leftarrow 0$ ,  $g \leftarrow 0$ ,  $i \leftarrow 1$ ,  $\max \leftarrow 0$ 
2 while  $i < r$  do
3    $f \leftarrow f + w_i$ ,  $g \leftarrow g + w_{r-i+1}$ 
4   if  $f = g$  then
5      $l \leftarrow 0$ ,  $i \leftarrow i + 1$ 
6     while  $i < r$  and  $w_i = 0$  do
7        $l \leftarrow l + 1$ ,  $i \leftarrow i + 1$ 
8     if  $l > \max$  then
9        $\max \leftarrow l$ 
10    else
11       $i \leftarrow i + 1$ 
12 return  $\min\{r + \max + 1, n + 1\}$ 

```

Let $m' = m + 1$. Then, there are no $j, k \in \{1, \dots, r - 1\}$ such that $j - k = m'$, $|w_1 \cdots w_k|_1 = |w_{r-k+1} \cdots w_r|_1$ and $|w_{k+1} \cdots w_j|_1 = 0$. Thus, by Lemma 6, we have that $\text{flip}(w, r + m') \in \mathcal{L}$, hence $\varphi(w) \leq r + m' = r + m + 1$.

Let now j, k be indices attaining the maximum in the definition of m , i.e., $1 < k < j < r$, $j - k = m$, $|w_1 \cdots w_k|_1 = |w_{r-k+1} \cdots w_r|_1$ and $|w_{k+1} \cdots w_j|_1 = 0$. Let $0 < m' \leq m$ then for $j' = k + m'$ we have $|w_1 \cdots w_k|_1 = |w_{r-k+1} \cdots w_r|_1$ and $|w_{k+1} \cdots w_{j'}|_1 = 0$. Then, by Lemma 6, $\text{flip}(w, r + m') \notin \mathcal{L}$. Hence $\varphi(w) > r + m'$, for $m' \leq m$, and in particular $\varphi(w) \geq r + m + 1$, which completes the proof. \square

Algorithm 1 implements the idea of Lemma 8 to compute φ . For a given prefix normal word w , it finds the position r of the rightmost one in w . Then, for each length i such that the number of 1s in $\text{pref}_i(w)$ (counted by f) is the same as the number of 1s in $w_{r-i+1} \cdots w_r$ (counted by g), the algorithm counts the number of 0s in w following $\text{pref}_i(w)$ and sets m to the maximum of the length of such runs of 0's. By Lemma 8 and the definition of φ it follows that $\min\{r + m + 1, n + 1\}$ is equal to φ , as correctly returned by Algorithm 1. It is not hard to see that the algorithm has linear running time since the two while-loops are only executed as long as $i < r$, and the variable i increases at each iteration of either loop. Therefore, the total number of iterations of the two loops together is upper bounded by $r \leq n$. Thus, we have proved the following lemma:

Lemma 9. *For $w \in \mathcal{L}_n \setminus \{0^n\}$, Algorithm 1 computes $\varphi(w)$ in $O(n)$ time.*

The next lemma gives the basis of our algorithm: applying either of the two operations $\text{flip}(w, \varphi(w))$ or $\text{bubble}(w)$ to a prefix normal word w results in another prefix normal word.

Lemma 10. *Let $w \in \mathcal{L}_n \setminus \{0^n\}$. Then the following holds:*

- (a) *for every ℓ , such that $\varphi(w) \leq \ell \leq n$, $\text{flip}(w, \ell)$ is prefix normal, and*
- (b) *if $|w|_1 \geq 2$ then $\text{bubble}(w)$ is prefix normal.*

Proof. Let $r = \text{RightmostOne}(w)$. In order to show (a) we can proceed as in the proof of the upper bound in Lemma 8. Fix $\varphi(w) \leq \ell \leq n$, and let $m' = \ell - r$. Then, by Lemma 8, there exist no $1 < j < k < r$ such that $k - j = m'$ and $|w_1 \cdots w_k|_1 = |w_{r-k+1} \cdots w_r|_1$ and $|w_{k+1} \cdots w_j|_1 = 0$. This, by Lemma 6, implies that $\text{flip}(w, \ell) \in \mathcal{L}$.

For (b), let $r' = \max\{i < r \mid w_i = 1\}$, i.e., r' is the position of the penultimate 1 of w . Let $w' = w_1 \cdots w_{r'} 0^{n-r'}$. By Fact 2 we have $w' \in \mathcal{L}$. Moreover, $r \geq \varphi(w')$, since $\text{flip}(w', r) = w \in \mathcal{L}$. Thus, by (a), $\text{bubble}(w) = \text{flip}(w', r+1) \in \mathcal{L}$. \square

Definition 11 (\mathcal{PN}). Given $w \in \mathcal{L}_n \setminus \{0^n\}$ with $r = \text{RightmostOne}(w)$, we define $\mathcal{PN}(w)$ as the set of all prefix normal words v such that $v = w_1 \cdots w_{r-1} \gamma$ for some γ with $|\gamma|_1 > 0$. Formally,

$$\mathcal{PN}(w) = \{v \in \mathcal{L} \mid v = w_1 \cdots w_{r-1} \gamma, |\gamma| = n - r + 1, |\gamma|_1 > 0\}.$$

Moreover, we will use the convention that $\mathcal{PN}(\text{flip}(w, \varphi(w))) = \emptyset$ if $\varphi(w) > n$, and $\mathcal{PN}(\text{bubble}(w)) = \emptyset$ if $\text{RightmostOne}(w) = n$, since then $\text{flip}(w, \varphi(w))$ resp. $\text{bubble}(w)$ are undefined.

Lemma 12. Given $w \in \mathcal{L}_n \setminus \{0^n\}$, we have

$$\mathcal{PN}(w) = \{w\} \dot{\cup} \mathcal{PN}(\text{flip}(w, \varphi(w))) \dot{\cup} \mathcal{PN}(\text{bubble}(w)).$$

We are now ready to describe an algorithm that computes all words in the set $\mathcal{PN}(w)$ for a prefix normal word w . The pseudocode is given in Algorithm 2. The procedure generates recursively the set $\mathcal{PN}(w)$ as the union of $\mathcal{PN}(\text{flip}(w, \varphi(w)))$ and $\mathcal{PN}(\text{bubble}(w))$. The call to subroutine *Visit()* is a placeholder indicating that the algorithm has generated a new word in $\mathcal{PN}(w)$, which could be printed or examined or processed as required. By Lemma 12 we know that *Visit()* is executed for each word in $\mathcal{PN}(w)$ exactly once.

Algorithm 2. GENERATE $\mathcal{PN}(w)$

Given w prefix normal word such that $|w|_1 > 1$, generate the set $\mathcal{PN}(w)$

```

1 if RightmostOne( $w$ )  $\neq n$  then
2    $w' = \text{bubble}(w)$ 
3   GENERATE  $\mathcal{PN}(w')$ 
4 Visit()
5  $j = \varphi(w)$ 
6 if  $j \leq n$  then
7    $w'' = \text{flip}(w, j)$ 
8   GENERATE  $\mathcal{PN}(w'')$ 

```

Lemma 13. For $w \in \mathcal{L}_n \setminus \{0^n\}$, Algorithm 2 generates all prefix normal words $v \in \mathcal{PN}(w)$ in lexicographic order, spending $O(n)$ time on each word generation.

Proof. Algorithm 2 recursively generates first all words in $\mathcal{PN}(\text{bubble}(w))$, then the word w , and finally the words in $\mathcal{PN}(\text{flip}(w, \varphi(w)))$. As we saw above, these sets form a partition of $\mathcal{PN}(w)$, hence every word $v \in \mathcal{PN}(w)$ is generated exactly once. Moreover, by definition of \mathcal{PN} , for every $u \in \mathcal{PN}(\text{bubble}(w))$ it holds that $u = w_1 \cdots w_{r-1} 0\gamma$ with $|\gamma| = n - r$ and $|\gamma|_1 > 0$, thus it follows that $u <_{\text{lex}} w$. In addition, for every $v \in \mathcal{PN}(\text{flip}(w, \varphi(w)))$ it holds that $v = w_1 \cdots w_{r-1} 1\delta\gamma$ where $|\delta| = k = \varphi(w) - r - 1$, $|\delta|_1 = 0$, $|\gamma| = n - r - k$ and $|\gamma|_1 > 0$, thus $w <_{\text{lex}} v$. Since these relations hold at every level of the recursion, it follows that the words are generated by Algorithm 2 in lexicographic order (Fig. 1). The linear time bound comes from the computation of φ in line 5 (see Lemma 9). \square

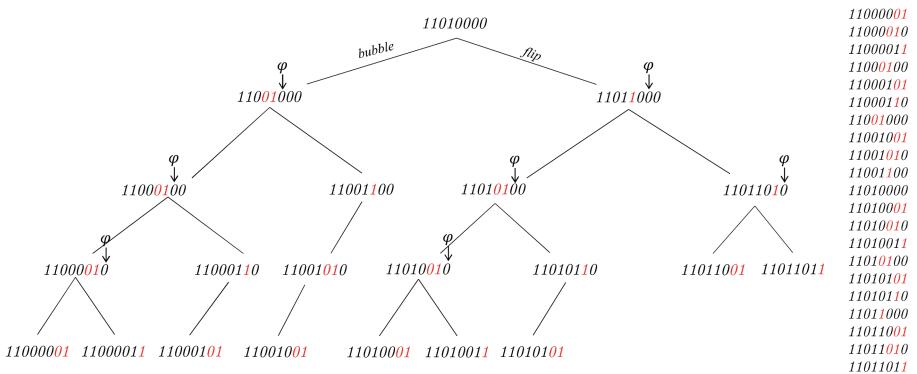


Fig. 1. The words in $\mathcal{PN}(11010000)$ represented as a tree. If a node of the tree is word w , its left child is $\text{bubble}(w)$ and its right child is $\text{flip}(w, \varphi(w))$. In the tree, the position of $\varphi(w)$ is indicated and the corresponding flip is highlighted in red (resp. grey) in the right child. Algorithm 2 generates these words by performing an in-order traversal of the tree. The corresponding list of words is on the right. (Color figure online)

Recall that by Fact 2(i) every prefix normal word of length n , other than 0^n , has 1 as its first character. Clearly, there is only one prefix normal word of length n with a single 1, namely 10^{n-1} . Moreover, by Fact 2(i) and the definition of \mathcal{PN} , the set of all prefix normal words of length n with at least two 1s coincides with $\mathcal{PN}(110^{n-2})$. These observations lead to Algorithm 3, which generates all prefix normal words w of length n . The procedure generates the words 0^n and 10^{n-1} and then $\mathcal{PN}(110^{n-2})$. Thus we have proved the following theorem:

Theorem 14. *The BUBBLE-FLIP algorithm generates all prefix normal words of length n in $O(n)$ time per word.*

Proof. From the previous discussion and Lemmas 12 and 13.

Algorithm 3. BUBBLE-FLIP

For a given n , generates all prefix normal words of length n

- 1 $w = 0^n$
 - 2 $\text{Visit}()$
 - 3 $w = 10^{n-1}$
 - 4 $\text{Visit}()$
 - 5 $w = 110^{n-2}$
 - 6 GENERATE $\mathcal{PN}(w)$
-

3.2 Prefix Normal Words with Given Critical Prefix

It was conjectured in [6] that the average length of the critical prefix taken over all prefix normal words is $O(\log n)$. Using the BUBBLE-FLIP algorithm, we can generate all prefix normal words with a given critical prefix u , which could be used to count all prefix normal words with critical prefix u .

Recall Definition 3. In the following lemma, we present a characterization of prefix normal words of length n with the same critical prefix $1^s 0^t$ in terms of our generation algorithm. For $s > 1, t \geq 0$, let us denote by $\text{CritSet}(s, t, n)$ the set of all prefix normal words of length n and critical prefix $1^s 0^t$.

Lemma 15. *Fix $s \geq 1$ and $t \geq 0$, and let $u = 1^s 0^t$. Then,*

$$\text{CritSet}(s, t, n) = \begin{cases} \{u\} & \text{if } s + t = n, \\ \{v\} \cup \mathcal{PN}(\text{flip}(v, \varphi(v))), & \text{if } s + t < n, \end{cases}$$

where $v = u10^{n-(s+t+1)}$.

4 On Finite and Infinite Prefix Normal Words

In this section, we study infinite prefix normal words. We focus on infinite extensions of finite prefix normal words which satisfy the prefix normal condition at every finite point and which are in a certain sense densest among all possible infinite extensions of the starting word. We show that words in this class are ultimately periodic, and we are able to determine both the size and the density of the period and to upper bound the starting point of the periodic behaviour.

4.1 Definitions

An infinite binary word is a function $v : \mathbb{N} \rightarrow \{0, 1\}$ (where \mathbb{N} denotes the set of natural numbers not including 0). The set of all infinite binary words is denoted $\{0, 1\}^\omega$. As with finite words, we refer to the i 'th character of v by v_i , to the factor spanning positions i through j by $v_i \cdots v_j$, and to the prefix of length i by $\text{pref}_i(v)$. As before, $P(i) = P_v(i)$ denotes the number of 1s in the prefix of length i . Given a finite word u , u^ω denotes the infinite word $uuu\cdots$. An infinite

word v is called *ultimately periodic* if there exist two integers $p, i_0 \geq 1$ such that $v_{i+p} = v_i$ for all $i \geq i_0$, or equivalently, if it can be written as $v = zu^\omega$ for some finite words z, u . The word v is called *periodic* if it is ultimately periodic with $i_0 = 1$, or equivalently, if there exists a finite word u such that $v = u^\omega$.

Definition 16 (Minimum density, minimum density prefix). Let $w \in \{0, 1\}^* \cup \{0, 1\}^\omega$. Denote by $D(i) = D_w(i) = P_w(i)/i$, the density of the prefix of length i . Define the minimum density of w as $\rho(w) = \inf\{D(i) \mid 1 \leq i\}$. If this infimum is attained somewhere, then we also define

$$\iota(w) = \min\{j \mid \forall i : D(j) \leq D(i)\}, \quad \text{and} \quad \kappa(w) = P_w(\iota(w)).$$

We refer to $\text{pref}_{\iota(w)}(w)$ as the minimum-density prefix, the shortest prefix with density $\rho(w)$. Note that $\iota(w)$ is always defined for finite words, while for infinite words, a prefix which attains the infimum may or may not exist.

Example 3. For $w = 110100101001$ and $u = 110100101010$ we have $\rho(w) = 5/11$, $\iota(w) = 11$, $\kappa(w) = 5$, and $\rho(u) = 1/2$, $\iota(u) = 6$, $\kappa(u) = 3$. For the infinite words $v = (10)^\omega$ and $v' = 1(10)^\omega$, we have $\rho(v) = \rho(v') = 1/2$, and $\iota(v) = 2$, $\kappa(v) = 1$, while $\iota(v')$ is undefined, since no prefix attains density $1/2$.

For a prefix normal word u , every factor of the infinite word $u0^\omega$ respects the prefix normal condition. Thus, we can define infinite prefix normal words.

Definition 17 (Infinite prefix normal words). An infinite binary word v is called prefix normal if, for every factor u of v , $|u|_1 \leq P_v(|u|)$.

Clearly, as for finite words, it holds that an infinite word is prefix normal if and only if all its prefixes are prefix normal. Therefore, the existence of infinite prefix normal words can also be derived from König's Lemma (see [16]), which states that the existence of an infinite prefix-closed set of finite words implies the existence of an infinite word which has all its prefixes in the set.

We now define an operation on finite prefix normal words which is similar to the flip operation from Sect. 3: it takes a prefix normal word w ending in a 1 and extends it by a run of 0s followed by a new 1, in such a way that this 1 is placed in the first possible position without violating prefix normality.

Definition 18 (Operation flipext). Let $w \in \mathcal{L} \cap \{0, 1\}^*1$. Define $\text{flipext}(w)$ as the finite word $w0^k1$, where $k = \min\{j \mid w0^j1 \in \mathcal{L}\}$. We further define the infinite word $v = \text{flipext}^\omega(w) = \lim_{i \rightarrow \infty} \text{flipext}^{(i)}(w)$.

For a prefix normal word w , the word $w0^{|w|}1$ is always prefix normal, so the operation flipext is well-defined. Let $w \in \mathcal{L}$ and $r = \text{RightmostOne}(w) < |w|$. Then $\text{flipext}(\text{pref}_r(w))$ is a prefix of $\text{flip}(w, \varphi(w))$ if and only if $\varphi(w) \leq |w|$, in particular, $\text{flip}(w, \varphi(w)) = \text{flipext}(\text{pref}_r(w)) \cdot 0^{|w|-\varphi(w)}$.

Definition 19 (Iota-factorization). Let w be a finite binary word, or an infinite binary word such that $\iota = \iota(w)$ exists. The iota-factorization of w is the factorization of w into ι -length factors, i.e. the representation of w in the form

$$w = u_1 u_2 \cdots u_r v,$$

where $r = \lfloor |w|/\iota \rfloor$, $|u_i| = \iota$ for $i = 1, \dots, r$, and $|v| < \iota$, for w finite, and

$$w = u_1 u_2 \cdots, \text{ where } |u_i| = \iota \text{ for all } i, \quad \text{for } w \text{ infinite.}$$

4.2 Flip Extensions and Ultimate Periodicity

Lemma 20. *Let w be a finite or infinite prefix normal word, such that $\iota = \iota(w)$ exists. Let $w = u_1 u_2 \cdots$ be the iota-factorization of w . Then for all i , $|u_i|_1 = \kappa(w)$.*

Proof. Since w is prefix normal, $|u_i| \leq \kappa = \kappa(w)$. On the other hand, assume there is an i_0 for which $|u_{i_0}|_1 < \kappa$. Then the prefix $u_1 u_2 \dots u_{i_0}$ has fewer than $i_0 \kappa$ many 1s, and thus density less than $i_0 \kappa / i_0 \iota = \kappa / \iota = D(\iota)$, in contradiction to the definition of ι . \square

The next lemma states that the iota-factorization of a word w constitutes a non-increasing sequence w.r.t. lexicographic order, as long as w fulfills a weaker condition than prefix normality, namely that factors of length $\iota(w)$ obey the prefix normal condition. That this does not imply prefix normality can be seen on the example $(1110010)^\omega$, which is not prefix normal.

Lemma 21. *Let w be a finite or infinite binary word, such that $\iota = \iota(w)$ exists. Let $w = u_1 u_2 \cdots$ be the iota-factorization of w . If for every i , $|u_i|_1 = \kappa = \kappa(w)$, and every factor u of length ι fulfills the prefix normal condition, then for all i , $u_i \geq_{\text{lex}} u_{i+1}$.*

Proof. Let us write $u_i = u_{i,1} \cdots u_{i,\iota}$. Let $a_{ij} = |u_{i,1} \cdots u_{i,j}|_1$ denote the number of 1s in the j -length prefix of u_i , and $b_{ij} = |u_{i,j+1} \cdots u_{i,\iota}|_1$ the number of 1s in the suffix of length $\iota - j$. By Lemma 20, we have that $a_{ij} + b_{ij} = \kappa$. On the other hand, $b_{ij} + a_{i+1,j} \leq \kappa$, since all ι -length factors satisfy the prefix normal condition. Thus, for all i : $a_{ij} \geq a_{i+1,j}$.

If $u_i \neq u_{i+1}$, let $h = \min\{j \mid j = 1, \dots, \iota : a_{ij} > a_{i+1,j}\}$. Thus, for every $j < h$, we have $u_{i,j} = u_{i+1,j}$ and $u_{i,h} = 1, u_{i+1,h} = 0$, implying $u_i \geq_{\text{lex}} u_{i+1}$. \square

Corollary 22. *Let w be a finite or infinite prefix normal word, such that $\iota = \iota(w)$ exists. Then for all i , $u_i \geq_{\text{lex}} u_{i+1}$, where u_i is the i 'th factor in the iota-factorization of w .*

We now prove that operation flipext leaves the minimum density invariant.

Lemma 23. *Let $w \in \mathcal{L}$ such that $w_n = 1$, and let $v \in \text{flipext}^*(w) \cup \{\text{flipext}^\omega(w)\}$. Then $\rho(v) = \rho(w)$, and as a consequence, $\iota(v) = \iota(w)$ and $\kappa(v) = \kappa(w)$.*

Theorem 24. *Let $w \in \mathcal{L}$ and $v = \text{flipext}^\omega(w)$. Then v is ultimately periodic. In particular, v can be written as $v = ux^\omega$, where $|x| = \iota(w)$ and $|x|_1 = \kappa(w)$.*

Proof. By Lemma 23, $\iota(v) = \iota(w)$, and by Lemma 20, in the iota-factorization of w , all factors u_i have $\kappa(w)$ 1s. Moreover, by Corollary 22, the factors u_i constitute a lexicographically non-increasing sequence. Since all u_i have length $\iota(w)$, and there are finitely many binary words of length $\iota(w)$, the claim follows. \square

Next we show that for a word $v \in \text{flipext}^*(w)$, in order to check the prefix normality of an extension of v , it is enough to verify that the suffixes up to length $|w|$ satisfy the prefix normal condition.

Lemma 25. *Let w be prefix normal and $v' \in \text{flipext}^*(w)$. Then for all $k \geq 0$ and $v = v'0^k1 \in \mathcal{L}$ if and only if for all $1 \leq j \leq |w|$, the suffixes of v of length j satisfy the prefix normal condition.*

By Theorem 24, we know that $v = \text{flipext}^\omega(w)$ has the form $v = ux^\omega$ for some x , whose length and density we can infer from w . The next theorem gives an upper bound on the waiting time for x , both in terms of the length of the non-periodic prefix u , and in the number of times a factor can occur before we can be sure that we have essentially found the periodic factor x (up to rotation).

Theorem 26. *Let $w \in \mathcal{L}$ and $v = \text{flipext}^\omega(w)$. Let us write $v = ux^\omega$, with $|x| = \iota(w)$ and x not a suffix of u . Then*

1. $|u| \leq (\binom{\iota}{\kappa} - 1)m\iota$, and
2. if for some $y \in \{0, 1\}^\iota$, it holds that y^{m+1} occurs with starting position $j > |w|$, then y is a rotation of x ,

where $\iota = \iota(w)$, $\kappa = \kappa(w)$, and $m = \lceil \frac{|w|}{\iota} \rceil$.

Proof. 1. Assuming 2., then every ι -length factor y which is not the final period can occur at most m times consecutively. By Corollary 22, consecutive non-equal factors in the iota-factorization of v are lexicographically decreasing, so no factor y can reoccur again once it has been replaced by another factor. By Theorem 24, the density of each factor is κ . There are at most $\binom{\iota}{\kappa}$ such y which are lexicographically smaller than $\text{pref}_\iota(w)$, and each of these has length ι .

2. By Lemma 25, in order to produce the next character of v , the operation flipext needs to access only the last $|w|$ many characters of the current word. After $m+1$ repetitions of u , it holds that the $|w|$ -length factor ending at position i is equal to the $|w|$ -length factor at position $i - \iota$, which proves the claim. \square

5 Ongoing Work

Our algorithm visits the words using an in-order traversal of the computation tree. If the words are visited in post-order, then the algorithm produces a Gray code: two consecutive words differ in at most one flip and one bubble operation.

We have developed several heuristics to speed up our algorithm. One of these is an application of Lemma 25, while another allows to propagate the value of the φ -function down the computation tree. We are currently evaluating the impact

of these heuristics on our algorithm's performance. Further, we want to apply the algorithm to enumerate (count) certain subsets of prefix normal words, with the aim of finding a closed form for the number of prefix normal words of length n , as well as for those with a given critical prefix.

Acknowledgements. We wish to thank three anonymous referees, who read our paper very carefully and whose detailed comments contributed to improving its exposition.

References

1. Amir, A., Apostolico, A., Hirst, T., Landau, G.M., Lewenstein, N., Rozenberg, L.: Algorithms for jumbled indexing, jumbled border and jumbled square on run-length encoded strings. *Theor. Comput. Sci.* **656**, 146–159 (2016)
2. Amir, A., Chan, T.M., Lewenstein, M., Lewenstein, N.: On hardness of jumbled indexing. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 114–125. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_10
3. Blondin Massé, A., de Carufel, J., Goupil, A., Lapointe, M., Nadeau, É., Van-domme, É.: Leaf realization problem, caterpillar graphs and prefix normal words. CoRR abs/1712.01942v1, previously part of CoRR abs/1709.09808 (2017)
4. Burcsi, P., Cicalese, F., Fici, G., Lipták, Zs.: Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.* **23**, 357–374 (2012)
5. Burcsi, P., Lipták, Zs., Fici, G., Ruskey, F., Sawada, J.: Normal, abby normal, prefix normal. In: Ferro, A., Luccio, F., Widmayer, P. (eds.) FUN 2014. LNCS, vol. 8496, pp. 74–88. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07890-8_7
6. Burcsi, P., Fici, G., Lipták, Zs., Ruskey, F., Sawada, J.: On combinatorial generation of prefix normal words. In: Kulikov, A.S., Kuznetsov, S.O., Pevzner, P. (eds.) CPM 2014. LNCS, vol. 8486, pp. 60–69. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07566-2_7
7. Burcsi, P., Fici, G., Lipták, Zs., Ruskey, F., Sawada, J.: On prefix normal words and prefix normal forms. *Theor. Comput. Sci.* **659**, 1–13 (2017)
8. Chan, T.M., Lewenstein, M.: Clustered integer 3SUM via additive combinatorics. In: Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC 2015), pp. 31–40 (2015)
9. Cicalese, F., Laber, E.S., Weimann, O., Yuster, R.: Approximating the maximum consecutive subsums of a sequence. *Theor. Comput. Sci.* **525**, 130–137 (2014)
10. Fici, G., Lipták, Zs.: On prefix normal words. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 228–238. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22321-1_20
11. Gagie, T., Hermelin, D., Landau, G.M., Weimann, O.: Binary jumbled pattern matching on trees and tree-like structures. *Algorithmica* **73**(3), 571–588 (2015)
12. Giaquinta, E., Grabowski, S.: New algorithms for binary jumbled pattern matching. *Inf. Process. Lett.* **113**(14–16), 538–542 (2013)
13. Knuth, D.: The Art of Computer Programming (TAOCP). <http://www-cs-faculty.stanford.edu/~knuth/taocp.html>. Accessed 15 Dec 2017
14. Knuth, D.E.: The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions. Addison-Wesley Professional, Boston (2005)

15. Kociumaka, T., Radoszewski, J., Rytter, W.: Efficient indexes for jumbled pattern matching with constant-sized alphabet. *Algorithmica* **77**(4), 1194–1215 (2017)
16. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
17. Moosa, T.M., Rahman, M.S.: Sub-quadratic time and linear space data structures for permutation matching in binary strings. *J. Discret. Alg.* **10**, 5–9 (2012)
18. Ruskey, F., Sawada, J., Williams, A.: Binary bubble languages and cool-lex order. *J. Comb. Theory Ser. A* **119**(1), 155–169 (2012)
19. Ruskey, F., Sawada, J., Williams, A.: De Bruijn sequences for fixed-weight binary strings. *SIAM J. Discret. Math.* **26**(2), 605–617 (2012)
20. Sawada, J., Williams, A.: Efficient oracles for generating binary bubble languages. *Electr. J. Comb.* **19**(1), P42 (2012)
21. Sawada, J., Williams, A., Wong, D.: Inside the binary reflected Gray code: Flip-Swap languages in 2-Gray code order. Unpublished manuscript (2017)



Permutations Sorted by a Finite and an Infinite Stack in Series

Murray Elder^(✉) and Yoong Kuan Goh

University of Technology Sydney, Ultimo, NSW 2007, Australia
murray.elder@uts.edu.au, gohyoongkuan@gmail.com

Abstract. We prove that the set of permutations sorted by a stack of depth $t \geq 3$ and an infinite stack in series has infinite basis, by constructing an infinite antichain. This answers an open question on identifying the point at which, in a sorting process with two stacks in series, the basis changes from finite to infinite.

Keywords: Patterns · String processing algorithms
Pattern avoiding permutations · Sorting with stacks

1 Introduction

A permutation is an arrangement of an ordered set of elements. Two permutations with same relative ordering are said to be *order isomorphic*, for example, 132 and 275 are order isomorphic as they have relative ordering ijk where $i < k < j$. A subpermutation of a permutation $p_1 \dots p_n$ is a word $p_{i_1} \dots p_{i_s}$ with $1 \leq i_1 < \dots < i_s \leq n$. A permutation p contains q if it has a subpermutation that is order isomorphic to q . For example, 512634 contains 231 since the subpermutation 563 is order isomorphic to 231. A permutation that does not contain q is said to *avoid* q . Let S_n denote the set of permutations of $\{1, \dots, n\}$ and let $S^\infty = \bigcup_{n \in \mathbb{N}_+} S_n$. The set of all permutations in S^∞ which avoid every permutation in $\mathcal{B} \subseteq S^\infty$ is denoted $Av(\mathcal{B})$. A set of permutations is a *pattern avoidance class* if it equals $Av(\mathcal{B})$ for some $\mathcal{B} \subseteq S^\infty$. A set $\mathcal{B} = \{q_1, q_2, \dots\} \subseteq S^\infty$ is an *antichain* if no q_i contains q_j for any $i \neq j$. An antichain \mathcal{B} is a *basis* for a pattern avoidance class \mathcal{C} if $\mathcal{C} = Av(\mathcal{B})$.

Sorting mechanisms are natural sources of pattern avoidance classes, since (in general) if a permutation cannot be sorted then neither can any permutation containing it. Knuth characterised the set of permutations that can be sorted by a single pass through an infinite stack as the set of permutations that avoid 231 [11]. Since then many variants of the problem have been studied, for example [1–9, 13–18]. The set of permutations sortable by a stack of depth 2 and an infinite stack in series has a basis of 20 permutations [7], while for two infinite stacks in series there is no finite basis [12]. For systems of a finite stack of depth 3 or more and infinite stack in series, it was not known whether the basis was finite or infinite.

Here we show that for depth 3 or more the basis is infinite. We identify an infinite antichain belonging to the basis of the set of permutations sortable by a stack of depth 3 and an infinite stack in series. A simple lemma then implies the result for depth 4 or more. A computer search by the authors [10] yielded 8194 basis permutations of lengths up to 13 (see Table 1; basis permutations are listed at <https://github.com/gohyoongkuan/stackSorting-3>). The antichain used to prove our theorem was found by examining this data and looking for patterns that could be arbitrarily extended.

Table 1. Number of basis elements for $S(3, \infty)$ of length up to 13

Permutation length	Number of sortable permutations	Number of basis elements
5	120	0
6	711	9
7	4700	83
8	33039	169
9	239800	345
10	1769019	638
11	13160748	1069
12	98371244	1980
13	737463276	3901

2 Preliminaries

The notation \mathbb{N} denotes the non-negative integers $\{0, 1, 2, \dots\}$ and \mathbb{N}_+ the positive integers $\{1, 2, \dots\}$.

Let M_t denote the machine consisting of a stack, R , of depth $t \in \mathbb{N}_+$ and an infinite stack, L , in series as in Fig. 1. A *sorting process* is the process of moving entries of a permutation from right to left from the input to stack R , then to stack L , then to the output, in some order. Each item must pass through both stacks, and at all times stack R may contain no more than t items (so if at some point stack R holds t items, the next input item cannot enter until an item is moved from R to L).

A permutation $\alpha = a_1 a_2 \dots a_n$ is in $S(t, \infty)$ if it can be sorted to $123\dots n$ using M_t . For example, $243651 \in S(t, \infty)$ for $t \geq 3$ since it can be sorted using the following process: place 2, 4 into stack R , move 4, 3, 2 across to stack L , place 6, 5, 1 into stack R , then output 1, 2, 3, 4, 5, 6. Note $243651 \notin S(2, \infty)$ by [7].

The following lemmas will be used to prove our main result.

Lemma 1. *Let $\alpha = a_1 a_2 \dots a_n \in S(t, \infty)$ for $t \in \mathbb{N}_+$. If $i < j$ and $a_i < a_j$ then in any sorting process that sorts α , if both a_i and a_j appear together in stack L then a_i must be above a_j .*

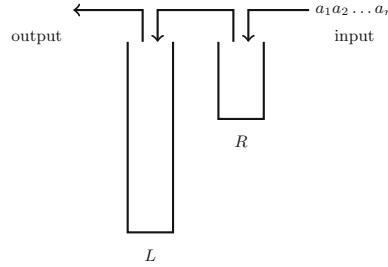


Fig. 1. A stack R of depth t and an infinite stack L in series

Proof. If a_j is above a_i in stack L then the permutation will fail to be sorted. \square

Lemma 2. Let $\alpha = a_1 a_2 \dots a_n \in S(t, \infty)$ for $t \geq 3$ and suppose $1 \leq i < j < k \leq n$ with $a_i a_j a_k$ order-isomorphic to 132. Then in any sorting process that sorts α , a_i, a_j, a_k do not appear together in stack R .

Proof. If a_i, a_j, a_k appear together in stack R , we must move a_k then a_j onto stack L before we can move a_i , but this means a_j, a_k violate Lemma 1. \square

Lemma 3. Let $\alpha = a_1 a_2 \dots a_n \in S(t, \infty)$ for $t \geq 3$ and $1 \leq i_1 < i_2 < \dots < i_6 \leq n$ with $a_{i_1} a_{i_2} \dots a_{i_6}$ order isomorphic to 243651. Then in any sorting process that sorts α , at some step of the process a_{i_4} and a_{i_5} appear together in stack R .

Proof. For simplicity let us write $a_{i_1} = 2, a_{i_2} = 4, a_{i_3} = 3, a_{i_4} = 6, a_{i_5} = 5, a_{i_6} = 1$. Before 6 is input, 2, 3, 4 are in the two stacks in one of the following configurations:

1. 2, 4, 3 are all in stack R . In this case we violate Lemma 2.
2. two items are in stack R and one is in stack L . In this case by Lemma 1 we cannot move 6 to stack L , so 6 must placed and kept in stack R . If $t = 3$ stack R is now full, so 5 cannot move into the system, and if $t \geq 4$, when 5 is input we violate Lemma 2.
3. one item, say a , is in stack R and two items are in stack L . In this case we cannot move 6, 5 into stack L by Lemma 1 so they remain in stack R on top of a , violating Lemma 2.
4. stack R is empty. In this case, 2, 3, 4 must be placed in stack L in order, else we violate Lemma 1. We cannot place 6, 5 into stack L until it is empty, so they must both stay in stack R until 4 is output.

In particular, the last case is the only possibility and in this case a_{i_4}, a_{i_5} appear in stack R together. \square

Lemma 4. Let $\alpha = a_1 a_2 \dots a_n \in S(t, \infty)$ for $t \geq 3$ and suppose $1 \leq i_1 < i_2 < \dots < i_5 \leq n$ with $a_{i_1} a_{i_2} \dots a_{i_5}$ order-isomorphic to 32514. Then, in any sorting process that sorts α , if a_{i_1}, a_{i_2} appear together in stack R , then at some step in the process a_{i_3}, a_{i_4} appear together in stack L .

Proof. For simplicity let us write $a_{i_1} = 3, a_{i_2} = 2, a_{i_3} = 5, a_{i_4} = 1, a_{i_5} = 4$. Figure 2 indicates the possible ways to sort these entries, and in the case that 2, 3 appear together in stack R we see that 4, 5 must appear in stack L together at some later point. \square

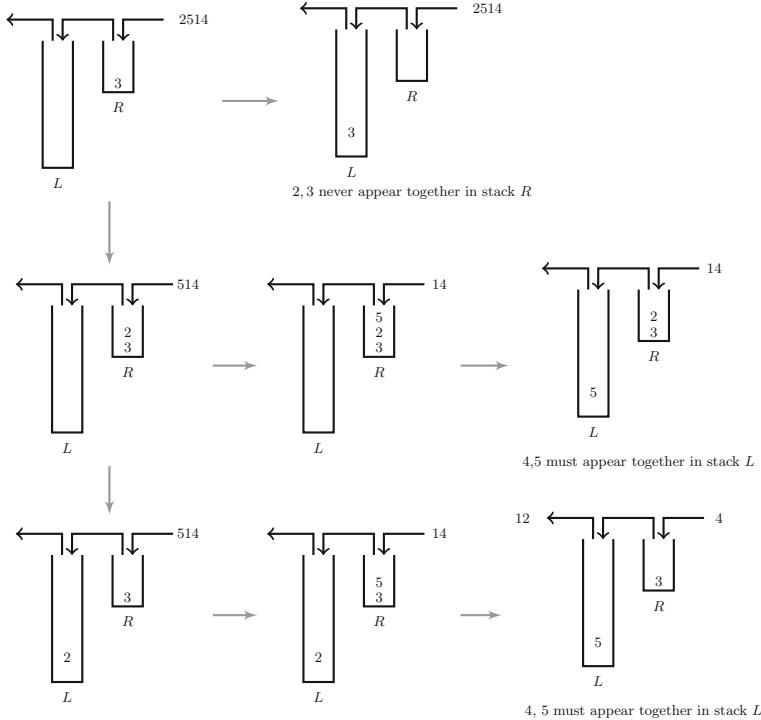


Fig. 2. Sorting 32514

Lemma 5. Let $\alpha = a_1 a_2 \dots a_n \in S(t, \infty)$ for $t \geq 3$ and suppose $1 \leq i_1 < i_2 < \dots < i_5 \leq n$ with $a_{i_1} a_{i_2} \dots a_{i_5}$ order-isomorphic to 32541. Then, in any sorting process that sorts α , if a_{i_1}, a_{i_2} appear together in stack L , then at the step that a_{i_1} is output,

1. a_{i_3}, a_{i_4} are both in stack R , and
2. if a_k is in stack L then $k < i_2$.

Proof. For simplicity let us write $a_{i_1} = 3, a_{i_2} = 2, a_{i_3} = 5, a_{i_4} = 4, a_{i_5} = 1$, and $\alpha = u_0 3 u_1 2 u_2 5 u_3 4 u_4 1 u_5$. Figure 3 indicates the possible ways to sort these entries. In the case that 2, 3 appear in stack R together, Lemma 1 ensures 2, 3 do not appear together in stack L . In the other case, before 3 is moved into stack L , any tokens in stack L come from $u_0 u_1$. Thus when 3 is output the only tokens in stack L will be a_k with $k < i_2$. Lemma 1 ensures that 4, 5 are not placed on top of 3 in stack L , so that the step that 3 is output they sit together in stack R . \square

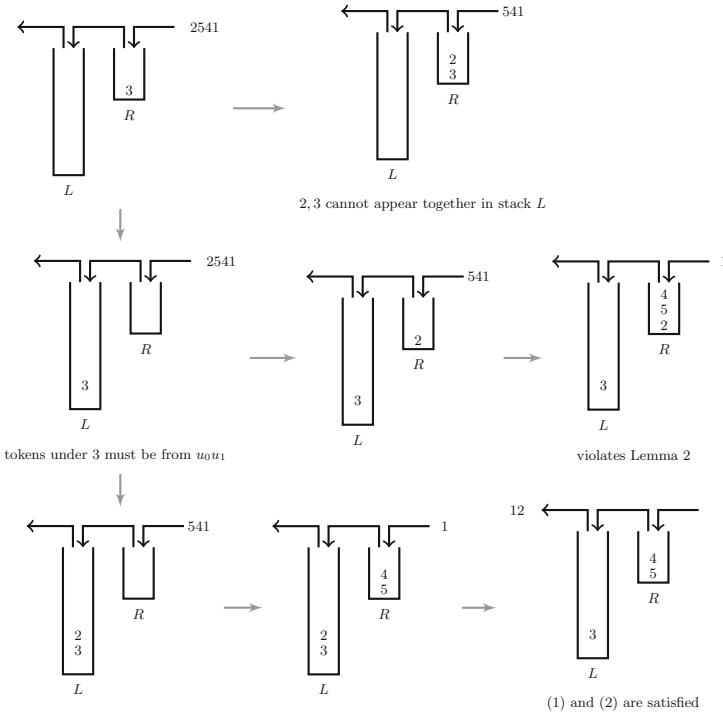


Fig. 3. Sorting 32541

3 An Infinite Antichain

We use the following notation. If $\alpha = a_1 \dots a_n$ is a permutation of $12 \dots n$ and $m \in \mathbb{Z}$ then let α_m be the permutation obtained by adding m to each entry of α . For example $(1\ 2\ 3)_4 = 5\ 6\ 7$ and $13_6 = 19$.

We construct a family of permutations $\mathcal{G} = \{G_i \mid i \in \mathbb{N}\}$ as follows. Define

$$\begin{aligned} P &= 2\ 4\ 3\ 7\ 6\ 1 \\ x_j &= (10\ 5\ 9)_{6j} \\ y_j &= (13\ 12\ 8)_{6j} \\ S_i &= (14\ 15\ 11)_{6i} \\ G_i &= P\ x_0\ y_0\ x_1\ y_1\ \dots\ x_i\ y_i\ S_i \end{aligned}$$

The first three terms are

$$G_0 = 2\ 4\ 3\ 7\ 6\ 1\ (10\ 5\ 9)\ (13\ 12\ 8)\ 14\ 15\ 11,$$

$$G_1 = 2\ 4\ 3\ 7\ 6\ 1\ (10\ 5\ 9)\ (13\ 12\ 8)\ (16\ 11\ 15)\ (19\ 18\ 14)\ 20\ 21\ 17,$$

$$G_2 = P\ (10\ 5\ 9)\ (13\ 12\ 8)\ (16\ 11\ 15)\ (19\ 18\ 14)\ (22\ 17\ 21)(25\ 24\ 20)\ 26\ 27\ 23.$$

A diagram of G_2 is shown in Fig. 4 which illustrates the general pattern.

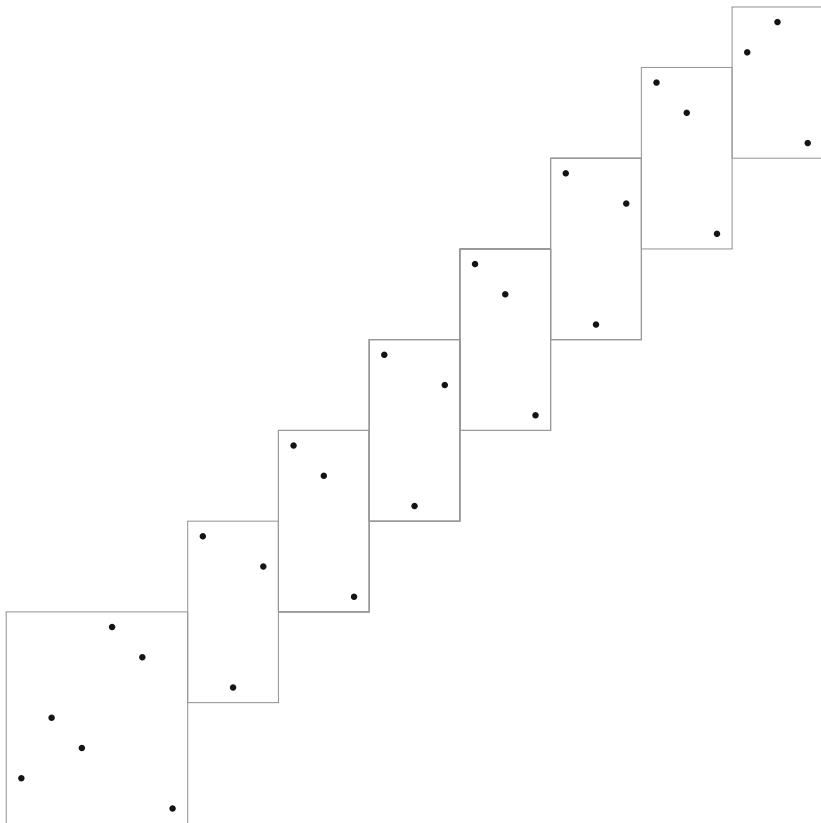


Fig. 4. Diagram of the permutation $G_2 = 2 4 3 7 6 1 x_0 y_0 x_1 y_1 x_2 y_2 26 27 23$

We will prove that each G_i is an element of the basis of $S(3, \infty)$ for all $i \in \mathbb{N}$. Note that if we define x_{-1}, y_{-1} to be empty, $G_{-1} = 243761895$ is also an element of the basis. We noticed this and G_0 had a particular pattern which we could extend using $x_j y_j$. However, we exclude G_{-1} from our antichain to make the proofs simpler.

Proposition 6. *The permutation $G_i \notin S(3, \infty)$ for all $i \in \mathbb{N}$.*

Proof. Suppose for contradiction that G_i can be sorted by some sorting process. Since P is order isomorphic to 243651, by Lemma 3 in any sorting process 7, 6 appear together in stack R . Next, 7 6 10 5 9 is order isomorphic to 32514 so by Lemma 4 since 7, 6 appear together in stack R we must have that 10, 9 appear together in stack L at some point in the process.

Now consider $x_j y_j = (10 5 9 13 12 8)_{6j}$, and assume that $10_{6j}, 9_{6j}$ both appear in stack L together. Since $(10 9 13 12 8)_{6j}$ is order isomorphic to 32541 by Lemma 5 $13_{6j}, 12_{6j}$ must be placed together in stack R and stay there until 10_{6j} is output.

Next consider $y_j x_{j+1} = (13 \ 12 \ 8 \ 16 \ 11 \ 15)_{6j}$, and assume that $13_{6j}, 12_{6j}$ both appear in stack R together. Then since $(13 \ 12 \ 16 \ 11 \ 15)_{6j}$ is order isomorphic to 32514 by Lemma 4 we have $16_{6j}, 15_{6j}$ appear together in stack L . Note that $16_{6j}, 15_{6j} = 10_{6(j+1)}, 9_{6(j+1)}$, so putting the above observations together we see that for all $0 \leq j \leq i$ we have $10_{6j}, 9_{6j}$ both appear in stack L together and $13_{6j}, 12_{6j}$ appear together in stack R and stay there until 10_{6j} is output.

Now we consider the suffix

$$x_i y_i S_i = (10 \ 5 \ 9 \ 13 \ 12 \ 8 \ 14 \ 15 \ 11)_{6i}$$

where $10_{6i}, 9_{6i}$ are together in stack L . Lemma 5 tells us not only that $13_{6i}, 12_{6i}$ appear together in stack R and stay there until 10_{6i} is output, but that anything sitting underneath 10_{6i} in stack L comes *before* 9_{6i} in G_i , so in particular $14_{6i}, 15_{6i}$ are not underneath 10_{6i} . All possible processes to sort $x_i y_i S$ are shown in Fig. 5. All possible sorting moves fail, which means G_i cannot be sorted. \square

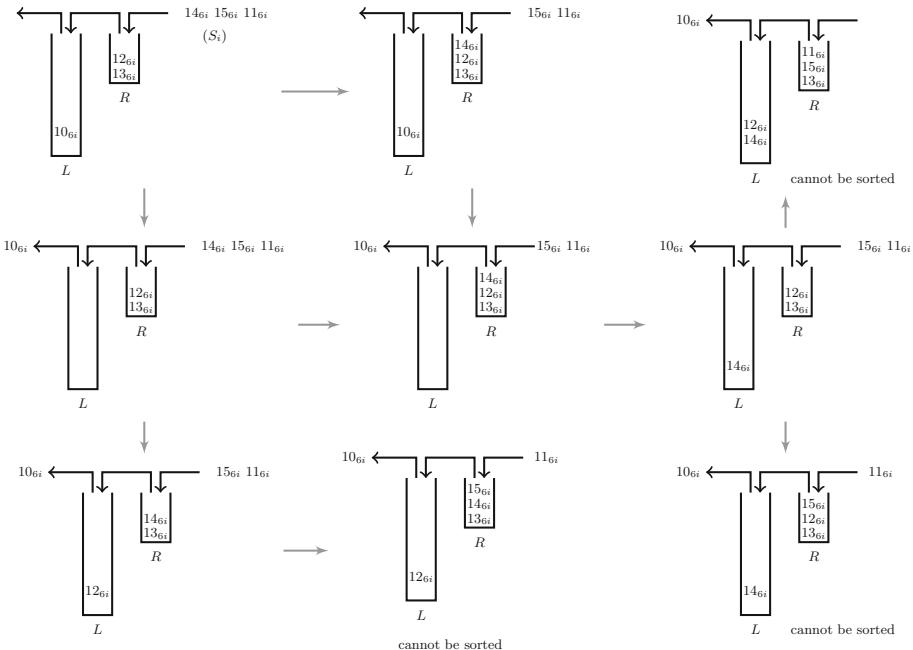


Fig. 5. All possible ways to sort $x_i y_i S$

The idea of the preceding proof can be summarised informally as follows. The prefix P forces 7, 6 to be together in stack R , then Lemmas 4 and 5 alternately imply that the $10_{6j}, 9_{6j}$ terms of x_j must be in stack L and the $13_{6j}, 12_{6j}$ terms of y_j must be in stack R . When we reach the suffix S_i the fact that certain entries

are forced to be in a particular stack means we are unable to sort the final terms. We now show that if a single entry is removed from G_i , we can choose to place the $10_{6j}, 9_{6j}$ terms in stack R and $13_{6j}, 12_{6j}$ terms in stack L , which allows the suffix to be sorted.

Lemma 7. *Let $0 \leq j \leq i$. If stack R contains one or both of $10_{6j}, 9_{6j}$ in ascending order, and $y_j \dots y_i S_i$ is to be input as in Fig. 6, then there is a sorting procedure to output all remaining entries in order.*

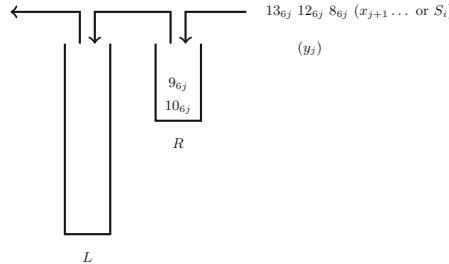


Fig. 6. A sortable configuration

Proof. For $j < i$ move $13_{6j}, 12_{6j}$ into stack L , output $8_{6j}, 9_{6j}, 10_{6j}$, move $16_{6j} = 10_{6(j+1)}$ into stack R , output $11_{6j} = 5_{6(j+1)}$, output $13_{6j}, 12_{6j}$ from stack L and input $15_{6j} = 9_{6(j+1)}$ so that the configuration has the same form as Fig. 6 with j incremented by 1.

For $j = i$ the remaining input is $(13 12 8 14 15 11)_{6j}$. Put $13_{6i}, 12_{6i}$ in stack L in order, output $8_{6i}, 9_{6i}, 10_{6i}$, put $14_{6i}, 15_{6i}$ in stack R and output $11_{6i}, 12_{6i}, 13_{6i}$, move 15_{6i} into stack L and output 14_{6i} then 15_{6i} .

If one of $9_{6j}, 10_{6j}$ is missing, use the same procedure ignoring the missing entry. \square

Lemma 8. *Let $0 \leq j \leq i$. If stack L contains one or both of $12_{6j}, 13_{6j}$ in ascending order, and $x_{j+1} \dots S_i$ (or just S_i if $j = i$) is to be input as in Fig. 7, then there is a sorting procedure to output all remaining entries in order.*

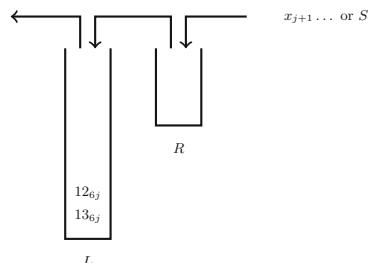


Fig. 7. Another sortable configuration

Proof. If $j < i$ move $10_{6(j+1)}$ into stack R , output $5_{6(j+1)}, 12_{6j}, 13_{6j}$, move $9_{6(j+1)}$ to stack R to reach the configuration in Fig. 6, which we can sort by Lemma 7. If $j = i$ then the remaining input is just $S_i = (14 \ 15 \ 11)_{6i}$: move $14_{6i}, 15_{6i}$ to stack R , then output all entries.

If one of $12_{6j}, 13_{6j}$ is missing, use the same procedure ignoring the missing entry. \square

Proposition 9. Let G'_i be a permutation obtained by removing a single entry from G_i . Then $G'_i \in S(3, \infty)$.

Proof. We give a deterministic procedure to sort G'_i . There are three cases depending on from where the entry is removed.

Term removed from P . Let P' be the factor P with one entry removed. We claim that there is a sorting sequence for $P'x_0$ which outputs the smallest six items in order and leaves $10, 9$ in stack R . To show this we simply consider all cases.

1. If 1 is removed, 2, 4, 3 can be output in order, then 7, 6 placed in stack L , 10 in stack R , then 5, 6, 7 output, and 9 placed on top of 10 in stack R .
2. If 2, 3, or 4 are removed, write $P' = ab761$ with $a, b \in \{2, 3, 4\}$. Place a, b in stack R , move 7, 6 into stack L , output 1, then output a, b in the correct order, then move 10 into stack R , output 5, 6, 7 and move 9 into stack R .
3. If 6 or 7 is removed, write $P' = 243a1$ with $a \in \{7, 6\}$. Place 4, 3, 2 in stack L in order, move a into stack R , output 1 then 2, 3, 4, then move a into stack L , move 10 into stack R , output 5, a and move 9 into stack R .

Thus after inputting $P'x_0$ we have the configuration shown in Fig. 6 with $j = 0$, which we can sort by Lemma 7.

Term removed from $x_s, 0 \leq s \leq i$.

Input P leaving 6, 7 in stack R , which brings us to the configuration in Fig. 8 with $j = 0$. Now assume we have input $P \dots x_{j-1}y_{j-1}$ with $j \leq s$ (note the convention that x_{-1}, y_{-1} are empty) and the configuration is as in Fig. 8.

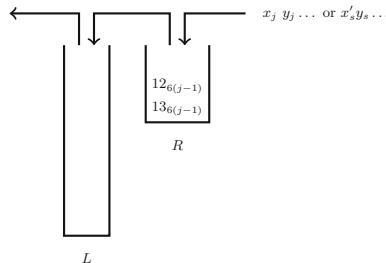


Fig. 8. Configuration after $P \dots x_{j-1}y_{j-1}$ is input

If $j < s$ we can input $x_j y_j$ into the stacks to arrive at the same configuration with j incremented by 1, as follows: move 10_{6j} to stack L , output $5_{6j}, 6_{6j} =$

$12_{6(j-1)}, 7_{6j} = 13_{6(j-1)}$, move 9_{6j} to stack L , move $13_{6j}, 12_{6j}$ to stack R , output $8_{6j}, 9_{6j}, 10_{6j}$.

If $j = s$, we proceed as follows:

1. If 5_{6s} removed, output $6_{6s} = 12_{6(s-1)}, 7_{6s} = 12_{6(s-1)}$, move $9_{6s}, 10_{6s}$ to stack R , to reach the configuration in Fig. 6 with $j = s$. From here the remaining entries can be sorted by Lemma 7.
2. If 10_{6s} is removed, output $5_{6s}, 6_{6s}, 7_{6s}$ and place 9_{6s} in stack R , to reach the configuration in Fig. 6 with $j = s$ and 10_{6s} missing. From here the remaining entries can be sorted Lemma 7.
3. If 9_{6s} is removed, move 6_{6s} to stack L , move 10_{6s} on top of 7_{6s} in stack R , output $5_{6s}, 6_{6s}$, move $13_{6s}, 12_{6s}$ into L , then output $8_{6s}, 10_{6s}$. This gives the configuration in Fig. 7 with $j = s$. From here the remaining entries can be sorted by Lemma 8.

Term removed from $y_s, 0 \leq s \leq i$ or S_i . Input Px_0 to reach the configuration in Fig. 9 with $j = 0$: move 2, 3, 4 into stack L , 7, 6 to R , output 1, 2, 3, 4, move 10 into L , output 5, 6, 7 then move 9 into L .

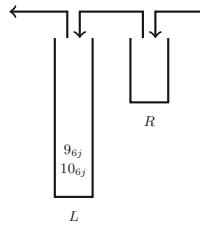


Fig. 9. Configuration after $Px_0y_0 \dots x_j$ is input

Now suppose we have input $Px_0y_0 \dots x_j$ to reach the configuration in Fig. 9. If no entry is removed from y_j and $j < i$ then we can input y_jx_{j+1} to return to the configuration in Fig. 9 with j incremented by 1 as follows: move $13_{6j}, 12_{6j}$ to stack R , output $8_{6j}, 9_{6j}, 10_{6j}$, move $10_{6(j+1)}$ to L , output $5_{6(j+1)} = 11_{6j}, 12_{6j}, 13_{6j}$, then move $9_{6(j+1)}$ to stack L .

If $j = s$ (y_s is removed):

1. If 8_{6s} is removed, output $9_{6s}, 10_{6s}$, move $13_{6s}, 12_{6s}$ to stack L to reach the configuration in Fig. 7, from which the remaining entries can be sorted by Lemma 8.
2. If $b \in \{13_{6s}, 12_{6s}\}$ is removed, place b in stack R , output $8_{6s}, 9_{6s}, 10_{6s}$, move b to stack L to reach the configuration in Fig. 7 with one of $12_{6s}, 13_{6s}$ removed, from which the remaining entries can be sorted a by Lemma 8.

If $j = i$ and the entry is removed from S_i , sort the remaining entries as follows:

1. If 11_{6i} is removed, place $13_{6i}, 12_{6i}$ into stack R , output $8_{6i}, 9_{6i}, 10_{6i}$, then $12_{6i}, 13_{6i}, 14_{6i}, 15_{6i}$.

2. If $b \in \{14_{6i}, 15_{6i}\}$ is removed, place $13_{6i}, 12_{6i}$ into stack R , output $8_{6i}, 9_{6i}, 10_{6i}$, move 12_{6i} into stack L , place b on top of 13_{6i} in stack R , output 11_{6i} then 12_{6i} , move b into stack L , output 13_{6i} then b . \square

Theorem 10. *The set of permutations that can be sorted by a stack of depth 3 and an infinite stack in series has an infinite basis.*

Proof. Proposition 6 shows that each G_i cannot be sorted, and Proposition 9 shows that no G_i can contain G_j for $j \neq i$ as a subpermutation since any subpermutation of G_i can be sorted. Thus $\mathcal{G} = \{G_i \mid i \in \mathbb{N}\}$ is an infinite antichain in the basis for $S(3, \infty)$. \square

4 From Finite to Infinitely Based

Let \mathcal{B}_t be the basis for $S(t, \infty)$ for $t \in \mathbb{N}_+$. Modifying Lemma 1 in [7] for the sorting case, we have the following:

Lemma 11. *If $\sigma \in \mathcal{B}_t$ has length n then either σ or $(213)_n\sigma$ belongs to \mathcal{B}_{t+1} .*

Proof. If $\sigma \notin S(t+1, \infty)$ then since $\sigma \in \mathcal{B}_t$, deleting any entry gives a permutation in $S(t, \infty) \subseteq S(t+1, \infty)$, so $\sigma \in \mathcal{B}_{t+1}$. Else $\sigma \in S(t+1, \infty)$. In any sorting process for $(213)_n\sigma$ the entries $1_n, 2_n, 3_n$ cannot appear together in stack L , so at least one entry must remain in stack R which means we must sort σ with stack R of depth at most t , which is not possible, so $(213)_n\sigma$ cannot be sorted. If we remove an entry of the prefix then the two entries $a, b \in \{1_n, 2_n, 3_n\}$ can be placed in stack L in order, leaving stack R depth $t+1$ so the permutation can be sorted, and if an entry is removed from σ then since $\sigma \in \mathcal{B}_t$ it can be sorted with R having one space occupied. \square

Theorem 12. *The set of permutations that can be sorted using a stack of depth $t \in \mathbb{N}_+$ and an infinite stack in series is finitely based if and only if $t \in \{1, 2\}$.*

Proof. We have $|\mathcal{B}_1| = 1$ and $|\mathcal{B}_2| = 20$ [7, 11]. Theorem 10 shows that \mathcal{B}_3 is infinite. Lemma 11 implies if \mathcal{B}_t is infinite then so is \mathcal{B}_{t+1} . \square

A small modification of Propositions 6 and 9 shows that for $t \geq 4$ the set $\mathcal{G}_t = \{G_{i,t}\}$, where $G_{i,t} = P(x_0y_0) \dots (x_iy_i)(14 \ 15 \ 16 \ \dots \ 12_t \ 11)_{6i}$, is an explicit antichain in the basis of $S(t, \infty)$. Details can be seen in [10].

References

- Albert, M., Atkinson, M., Linton, S.: Permutations generated by stacks and deques. Ann. Comb. **14**(1), 3–16 (2010). <https://doi.org/10.1007/s00026-010-0042-9>
- Albert, M., Bousquet-Mélou, M.: Permutations sortable by two stacks in parallel and quarter plane walks. Eur. J. Comb. **43**, 131–164 (2015). <https://doi.org/10.1016/j.ejc.2014.08.024>

3. Atkinson, M.D., Livesey, M.J., Tulley, D.: Permutations generated by token passing in graphs. *Theor. Comput. Sci.* **178**(1–2), 103–118 (1997). [http://dx.doi.org/10.1016/S0304-3975\(96\)00057-6](http://dx.doi.org/10.1016/S0304-3975(96)00057-6)
4. Atkinson, M.D., Murphy, M.M., Ruškuc, N.: Sorting with two ordered stacks in series. *Theor. Comput. Sci.* **289**(1), 205–223 (2002). [http://dx.doi.org/10.1016/S0304-3975\(01\)00270-5](http://dx.doi.org/10.1016/S0304-3975(01)00270-5)
5. Bóna, M.: A survey of stack-sorting disciplines. *Electron. J. Comb.* **9**(2), A1 (2003)
6. Claesson, A., Dukes, M., Steingrímsson, E.: Permutations sortable by $n - 4$ passes through a stack. *Ann. Comb.* **14**(1), 45–51 (2010). <https://doi.org/10.1007/s00026-010-0044-7>
7. Elder, M.: Permutations generated by a stack of depth 2 and an infinite stack in series. *Electron. J. Comb.* **13**(1), Research Paper #68 (2006). http://www.combinatorics.org/Volume_13/Abstracts/v13i1r68.html
8. Elder, M., Lee, G., Rechnitzer, A.: Permutations generated by a depth 2 stack and an infinite stack in series are algebraic. *Electron. J. Comb.* **22**(2), Paper 2.16, 23 (2015)
9. Elvey-Price, A., Guttmann, A.J.: Permutations sortable by two stacks in series. *Adv. Appl. Math.* **83**, 81–96 (2017). <https://doi.org/10.1016/j.aam.2016.09.003>
10. Goh, Y.K.: Ph.D. thesis, University of Technology Sydney (2019, in preparation)
11. Knuth, D.E.: *The Art of Computer Programming: Sorting and Searching*, vol. 3. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Publishing Co., Reading (1973)
12. Murphy, M.M.: Restricted permutations, antichains, atomic classes, stack sorting. Ph.D. thesis, University of St Andrews (2002)
13. Pierrot, A., Rossin, D.: 2-stack pushall sortable permutations. *CoRR* abs/1303.4376 (2013). <http://arxiv.org/abs/1303.4376>
14. Pierrot, A., Rossin, D.: 2-stack sorting is polynomial. *Theory Comput. Syst.* **60**(3), 552–579 (2017). <https://doi.org/10.1007/s00224-016-9743-8>
15. Smith, R.: Two stacks in series: a decreasing stack followed by an increasing stack. *Ann. Comb.* **18**(2), 359–363 (2014). <https://doi.org/10.1007/s00026-014-0227-8>
16. Smith, R., Vatter, V.: The enumeration of permutations sortable by pop stacks in parallel. *Inf. Process. Lett.* **109**(12), 626–629 (2009). <https://doi.org/10.1016/j.ipl.2009.02.014>
17. Tarjan, R.: Sorting using networks of queues and stacks. *J. Assoc. Comput. Mach.* **19**, 341–346 (1972). <https://doi.org/10.1145/321694.321704>
18. West, J.: Sorting twice through a stack. *Theor. Comput. Sci.* **117**(1), 303–313 (1993)



On Periodicity Lemma for Partial Words

Tomasz Kociumaka^(✉), Jakub Radoszewski, Wojciech Rytter,
and Tomasz Waleń

Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw,
Banacha 2, 02-097 Warsaw, Poland
{kociumaka,jrad,rytter,walen}@mimuw.edu.pl

Abstract. We investigate the function $L(h, p, q)$, called here the *threshold function*, related to periodicity of partial words (words with holes). The value $L(h, p, q)$ is defined as the minimum length threshold which guarantees that a natural extension of the periodicity lemma is valid for partial words with h holes and (strong) periods p, q . We show how to evaluate the threshold function in $\mathcal{O}(\log p + \log q)$ time, which is an improvement upon the best previously known $\mathcal{O}(p + q)$ -time algorithm. In a series of papers, the formulae for the threshold function, in terms of p and q , were provided for each fixed $h \leq 7$. We demystify the generic structure of such formulae, and for each value h we express the threshold function in terms of a piecewise-linear function with $\mathcal{O}(h)$ pieces.

Keywords: Partial words · Words with don't cares
Periodicity lemma

1 Introduction

Consider a word X of length $|X| = n$, with its positions numbered 0 through $n - 1$. We say that X has a period p if $X[i] = X[i + p]$ for all $0 \leq i < n - p$. In this case, the prefix $P = X[0..p - 1]$ is called a *string period* of X . Our work can be seen as a part of the quest to extend Fine and Wilf's Periodicity Lemma [11], which is a ubiquitous tool of combinatorics on words, into partial words.

Lemma 1 (Periodicity Lemma [11]). If p, q are periods of a word X of length $|X| \geq p + q - \gcd(p, q)$, then $\gcd(p, q)$ is also a period of X .

A partial word is a word over the alphabet $\Sigma \cup \{\diamond\}$, where \diamond denotes a hole (a don't care symbol). In what follows, by n we denote the length of the partial word and by h the number of holes. For $a, b \in \Sigma \cup \{\diamond\}$, the relation of matching \approx is defined so that $a \approx b$ if $a = b$ or either of these symbols is a hole. A (solid) word P of length p is a *string period* of a partial word X if $X[i] \approx P[i \bmod p]$ for $0 \leq i < n$. In this case, we say that the integer p is a (*strong*) *period* of X .

Supported by the Polish National Science Center, grant no 2014/13/B/ST6/00770.

We aim to compute the optimal thresholds $L(h, p, q)$ which make the following generalization of the periodicity lemma valid:

Lemma 2 (Periodicity Lemma for Partial Words). *If X is a partial word with h holes with periods p, q and $|X| \geq L(h, p, q)$, then $\gcd(p, q)$ is also a period of X .*

If $\gcd(p, q) \in \{p, q\}$, then Lemma 2 trivially holds for each partial word X . Otherwise, as proved by Fine and Wilf [11], the threshold in Lemma 1 is known to be optimal, so $L(0, p, q) = p + q - \gcd(p, q)$.

Example 3. $L(1, 5, 7) = 12$, because:

- each partial word of length at least 12 with one hole and periods 5, 7 has also period $1 = \gcd(5, 7)$,
- the partial word $\text{ababaababa} \diamond$ of length 11 has periods 5, 7 and does not have period 1.

As our main aim, we examine the values $L(h, p, q)$ as a function of p, q for a given h . Closed-form formulae for $L(h, \cdot, \cdot)$ with $h \leq 7$ were given in [2, 5, 24]. In these cases, $L(h, p, q)$ can be expressed using a constant number of functions linear in p, q , and $\gcd(p, q)$. We discover a common pattern in such formulae which lets us derive a closed-form formula for $L(h, p, q)$ with arbitrary fixed h using a sequence of $\mathcal{O}(h)$ fractions. Our construction relies on the theory of continued fractions; we also apply this link to describe $L(h, p, q)$ in terms of standard Sturmian words.

As an intermediate step, we consider a dual *holes function* $H(n, p, q)$, which gives the minimum number of holes h for which there is a partial word of length n with h holes and periods p, q which do not satisfy Lemma 2.

Example 4. We have $H(11, 5, 7) = 1$ because:

- $H(11, 5, 7) \geq 1$: due to the classic periodicity lemma, every solid word of length 11 with periods 5 and 7 has period $1 = \gcd(5, 7)$, and
- $H(11, 5, 7) \leq 1$: $\text{ababaababa} \diamond$ is non-unary, has one hole and periods 5, 7.

We have $H(12, 5, 7) \leq H(11, 5, 7) + 1 = 2$ since appending \diamond preserves periods. In fact $H(12, 5, 7) = H(15, 5, 7) = 2$. However, there is no non-unary partial word of length 16 with 2 holes and periods 5, 7, so $L(2, 5, 7) = 16$; see Table 1.

Table 1. The optimal non-unary partial words with periods 5, 7 and $h = 0, \dots, 5$ holes (of length $L(h, 5, 7) - 1$) and the values $H(n, 5, 7)$ for $n = 10, \dots, 25$.

h	$L(h, 5, 7)$	example of length $L(h, 5, 7) - 1$
0	11	ababaababa
1	12	$\text{ababaababa} \diamond$
2	16	$\text{ababaababa} \diamond \diamond \text{aba}$
3	19	$\text{aaaabaaaa} \diamond \text{a} \diamond \text{aa} \diamond \text{aaa}$
4	21	$\text{aba} \diamond \diamond \text{ababaababa} \diamond \diamond \text{aba}$
5	25	$\text{aaaabaaaa} \diamond \text{a} \diamond \text{aa} \diamond \text{aaa} \diamond \diamond \text{aaaa}$
$n :$	10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25	
$H(n, 5, 7) :$	0 1 2 2 2 2 3 3 3 4 4 5 5 5 5 6	

For a function $f(n, p, q)$ monotone in n , we define its *generalized inverse* as:

$$\tilde{f}(h, p, q) = \min\{n : f(n, p, q) > h\}.$$

Observation 5. $L = \tilde{H}$.

As observed above, Lemma 2 becomes trivial if $p \mid q$. The case of $p \mid 2q$ is known to be special as well, but it has been fully described in [24]. Furthermore, it was shown in [5, 23] that the case of $\gcd(p, q) > 1$ is easily reducible to that of $\gcd(p, q) = 1$. We recall these existing results in Sect. 4, while in the other sections we assume that $\gcd(p, q) = 1$ and $p, q > 2$.

Previous Results. The study of periods in partial words was initiated by Berstel and Boasson [2], who proved that $L(1, p, q) = p + q$. They also showed that the same bound holds for *weak periods*¹ p and q . Shur and Konovalova [24] developed exact formulae for $L(2, p, q)$ and $L(h, 2, q)$, and an upper bound for $L(h, p, q)$. A formula for $L(h, p, q)$ with small values h was shown by Blanchet-Sadri et al. [3], whereas for large h , Shur and Gamzova [23] proved that the optimal counterexamples of length $L(h, p, q) - 1$ belong to a very restricted class of *special arrangements*. The latter contribution leads to an $\mathcal{O}(p + q)$ -time algorithm for computing $L(h, p, q)$. An alternative procedure with the same running time was shown by Blanchet-Sadri et al. [5], who also stated closed-form formulae for $L(h, p, q)$ with $h \leq 7$. Weak periods were further considered in [4, 6, 25].

Other known extensions of the periodicity lemma include a variant with three [8] and an arbitrary number of specified periods [13, 26], the so-called new periodicity lemma [1, 10], a periodicity lemma for repetitions with morphisms [18], extensions into abelian [9] and k -abelian [14] periodicity, into abelian periodicity for partial words [7], into bidimensional words [19], and other variations [12, 20].

Our Results. First, we show how to compute $L(h, p, q)$ using $\mathcal{O}(\log p + \log q)$ arithmetic operations, improving upon the state-of-the-art complexity $\mathcal{O}(p + q)$.

Furthermore, for any fixed h in $\mathcal{O}(h \log h)$ time we can compute a compact description of the threshold function $L(h, p, q)$. For the base case of $p < q$, $\gcd(p, q) = 1$, and $h < p + q - 2$, the representation is piecewise linear in p and q . More precisely, the interval $[0, 1]$ can be split into $\mathcal{O}(h)$ subintervals I so that $L(h, p, q)$ restricted to $\frac{p}{q} \in I$ is of the form $a \cdot p + b \cdot q + c$ for some integers a, b, c .

Overview of the Paper. We start by introducing two auxiliary functions H^s and H^d which correspond to two restricted families of partial words. Our first key step is to prove that the value $H(n, p, q)$ is always equal to $H^s(n, p, q)$ or $H^d(n, p, q)$ and to characterize the arguments n for which either case holds. The final function L is then obtained as a combination of the generalized inverses L^s and L^d of H^s and H^d , respectively. Developing the closed-form formula for L^d requires considerable effort; this is where continued fractions arise.

¹ An integer p is a weak period of X if $X[i] \approx X[i + p]$ for all $0 \leq i < n - p$.

2 Functions H^s and L^s

For relatively prime integers p, q , $1 < p < q$, and an integer $n \geq q$, let us define

$$H^s(n, p, q) = \left\lfloor \frac{n-q}{p} \right\rfloor + \left\lfloor \frac{n-q+1}{p} \right\rfloor.$$

We shall prove that $H(n, p, q) \leq H^s(n, p, q)$ for a suitable range of lengths n .

Fine and Wilf [11] constructed a word of length $p+q-2$ with periods p and q and without period 1. For given p, q we choose such a word $S_{p,q}$ and we define a partial word $W_{p,q}$ as follows, setting $k = \lfloor q/p \rfloor$ (see Fig. 1):

$$W_{p,q} = (S_{p,q}[0..p-3] \diamond \diamond)^k \cdot S_{p,q} \cdot (\diamond \diamond S_{p,q}[q..q+p-3])^k.$$

Intuitively, the partial word $W_{p,q}$ is an extension of $S_{p,q}$ preserving the period p , in which a small number of symbols is changed to holes to guarantee the periodicity with respect to q . This is formally proved in the full version [16].

Lemma 6. *The partial word $W_{p,q}$ has periods p and q .*

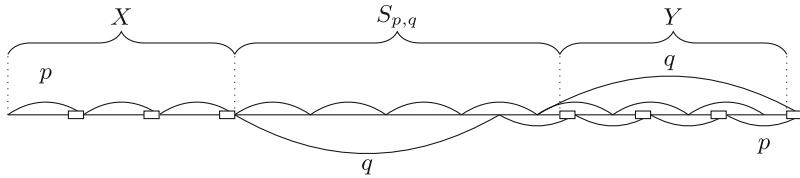


Fig. 1. The structure of the partial word $W_{p,q} \diamond \diamond = X \cdot S_{p,q} \cdot Y \diamond \diamond$ for $\lfloor q/p \rfloor = 3$. Tiny rectangles correspond to two holes $\diamond \diamond$. We have $|X| = |Y| = p \lfloor q/p \rfloor = 3p$ and $|W_{p,q}| = p + q + 2p \lfloor q/p \rfloor - 2 = q + 7p - 2$. There are $4 \cdot \lfloor q/p \rfloor = 12$ holes.

Example 7. For $p = 5$ and $q = 7$, we can take $S_{5,7} = \text{ababaababa}$ and

$$W_{5,7} = \text{aba} \diamond \diamond \text{ababaababa} \diamond \diamond \text{aba}.$$

This partial word has length 20 and 4 holes. Hence, $H(20, 5, 7) \leq 4 = H^s(20, 5, 7)$ and $L(4, 5, 7) \geq 21$. In fact, these bounds are tight; see Table 1.

We use the word $S_{p,q}$ and the partial word $W_{p,q} \diamond \diamond$ to show that H^s is an upper bound for H for all intermediate lengths n ($|S_{p,q}| \leq n \leq |W_{p,q} \diamond \diamond|$).

Lemma 8. *Let $1 < p < q$ be relatively prime integers. For each length $p+q-2 \leq n \leq p+q+2p \lfloor q/p \rfloor$, we have $H(n, p, q) \leq H^s(n, p, q)$.*

Proof. We extend $S_{p,q}$ to $W_{p,q} \diamond \diamond$ symbol by symbol, first prepending the characters before $S_{p,q}$, and then appending the characters after $S_{p,q}$. By Lemma 6, the resulting partial word has periods p and q because it is contained in $W_{p,q} \diamond \diamond$. Moreover, it is not unary because it contains $S_{p,q}$.

A hole is added at the first two iterations among every p iterations. Hence, the total number of holes is as claimed:

$$\left\lceil \frac{n - |S_{p,q}|}{p} \right\rceil + \left\lceil \frac{n - |S_{p,q}| - 1}{p} \right\rceil = \left\lfloor \frac{n - q + 1}{p} \right\rfloor + \left\lfloor \frac{n - q}{p} \right\rfloor = H^s(n, p, q),$$

because $\lceil \frac{x}{p} \rceil = \lfloor \frac{x+p-1}{p} \rfloor$ for every integer x . \square

Finally, the function $L^s = \widetilde{H}^s$ is very simple and easily computable.

Lemma 9. *If $h \geq 0$ is an integer, then $L^s(h, p, q) = \lceil \frac{h+1}{2} \rceil p + q - (h+1) \bmod 2$.*

Proof. We have to determine the smallest n such that $\left\lfloor \frac{n-q}{p} \right\rfloor + \left\lfloor \frac{n-q+1}{p} \right\rfloor = h+1$. There are two cases, depending on parity of h :

Case 1: $h = 2k$. In this case $\left\lfloor \frac{n-q}{p} \right\rfloor = k$ and $\left\lfloor \frac{n-q+1}{p} \right\rfloor = k+1$. Hence, $n - q + 1 = p(k+1)$, i.e., $n = p(k+1) + q - 1 = \lceil \frac{h+1}{2} \rceil p + q - (h+1) \bmod 2$.

Case 2: $h = 2k+1$. In this case $\left\lfloor \frac{n-q}{p} \right\rfloor = k+1$ and $\left\lfloor \frac{n-q+1}{p} \right\rfloor = k+1$. Hence, $n - q = p(k+1)$, i.e., $n = p(k+1) + q = \lceil \frac{h+1}{2} \rceil p + q - (h+1) \bmod 2$. \square

3 Functions H^d and L^d

In this section, we study a family of partial words corresponding to the *special arrangements* introduced in [23]. For relatively prime integers $p, q > 1$, we say that a partial word S of length $n \geq \max(p, q)$ is (p, q) -*special* if it has a position l such that for each position i :

$$S[i] = \begin{cases} a & \text{if } p \nmid (l-i) \text{ and } q \nmid (l-i), \\ b & \text{if } p \mid (l-i) \text{ and } q \mid (l-i), \\ \diamond & \text{otherwise.} \end{cases}$$

Let $H^d(n, p, q)$ be the minimum number of holes in a (p, q) -special partial word of length n .

Fact 10. *For each $n \geq \max(p, q)$, we have $H(n, p, q) \leq H^d(n, p, q)$.*

Proof. Observe that every (p, q) -special partial word has periods p and q . However, due to $p, q > 1$, it does not have period $1 = \gcd(p, q)$. \square

Example 11. The partial word aaaabaaaa◊a◊aa◊aaa is $(5, 7)$ -special (with $l = 4$), so $H(18, 5, 7) \leq H^d(18, 5, 7) \leq 3$ and $L(3, 5, 7) \geq 19$. In fact, these bounds are tight; see Table 1.

To derive a formula for $H^d(n, p, q)$, let us introduce an auxiliary function G , which counts integers $i \in \{1, \dots, n\}$ that are multiples of p or of q but not both:

$$G(n, p, q) = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{q} \right\rfloor - 2 \left\lfloor \frac{n}{pq} \right\rfloor.$$

The function H^d can be characterized using G , while the generalized inverse $L^d = \widetilde{H}^d$ admits a dual characterization in terms of \widetilde{G} ; see also Table 2.

Lemma 12. *Let $p, q > 1$ be relatively prime integers.*

- (a) *If $n \geq \max(p, q)$, then $H^d(n, p, q) = \min_{l=0}^{n-1} (G(l, p, q) + G(n-l-1, p, q))$.*
- (b) *If $h \geq 0$, then $L^d(h, p, q) = \max_{k=0}^h (\widetilde{G}(k, p, q) + \widetilde{G}(h-k, p, q))$.*

Proof. Let S be a (p, q) -special partial word of length n with h holes, k of which are located to the left of position l . Observe that $k = G(l, p, q)$ (so $l+1 \leq \widetilde{G}(k, p, q)$) and $h-k = G(n-l-1, p, q)$ (so $n-l \leq \widetilde{G}(h-k, p, q)$). Hence, $h = G(l, p, q) + G(n-l-1, p, q)$ and $n+1 \leq \widetilde{G}(k, p, q) + \widetilde{G}(h-k, p, q)$. The claimed equalities follow from the fact that these bounds can be attained for each l and k , respectively. \square

Table 2. Functions \widetilde{G} and L^d for $p = 5$, $q = 7$, and $h = 0, \dots, 15$. By Lemma 12, we have, for example, $L^d(8, 5, 7) = \max(\widetilde{G}(0, 5, 7) + \widetilde{G}(8, 5, 7), \dots, \widetilde{G}(4, 5, 7) + \widetilde{G}(4, 5, 7)) = \max(5+28, 7+25, 10+21, 14+20, 15+15) = 34$.

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\widetilde{G}(h, 5, 7)$	5	7	10	14	15	20	21	25	28	30	40	42	45	49	50	55
$L^d(h, 5, 7)$	10	12	15	19	21	25	28	30	34	35	45	47	50	54	56	60

4 Characterizations of H and L

Shur and Gamzova in [23] proved that $H(n, p, q) = H^d(n, p, q)$ for $n \geq 3q + p$. We give a complete characterization of H in terms of H^d and H^s , and we derive an analogous characterization of L in terms of L^d and L^s .

A tedious proof of the following theorem is given in the full version [16].

Theorem 13. *Let p and q be relatively prime integers such that $2 < p < q$. For each integer $n \geq p + q - 2$, we have*

$$H(n, p, q) = \begin{cases} H^s(n, p, q) & \text{if } n \leq q + p \lceil \frac{q}{p} \rceil - 1 \text{ or } 3q \leq n \leq q + 3p - 1, \\ H^d(n, p, q) & \text{otherwise.} \end{cases}$$

Moreover, for each integer $h \geq 0$:

$$L(h, p, q) = \begin{cases} L^s(h, p, q) & \text{if } \frac{q}{p} > \lceil \frac{h}{2} \rceil \text{ or } (h = 4 \text{ and } \frac{q}{p} < \frac{3}{2}) \\ L^d(h, p, q) & \text{otherwise.} \end{cases}$$

The remaining cases have already been well understood:

Fact 14 [5, 23]. If $p, q > 1$ are integers such that $\gcd(p, q) \notin \{p, q\}$, then

$$L(h, p, q) = \gcd(p, q) \cdot L\left(h, \frac{p}{\gcd(p, q)}, \frac{q}{\gcd(p, q)}\right).$$

Fact 15 [24]. If q, h are integers such that $q > 2$, $2 \nmid q$, and $h \geq 0$, then

$$L(h, 2, q) = (2p + 1) \left\lfloor \frac{h}{p} \right\rfloor + h \bmod p.$$

The results above lead to our first algorithm for computing $L(h, p, q)$.

Corollary 16. *Given integers $p, q > 1$ such that $\gcd(p, q) \notin \{p, q\}$ and an integer $h \geq 0$, the value $L(h, p, q)$ can be computed in $\mathcal{O}(h + \log p + \log q)$ time.*

Proof. First, we apply Fact 14 to reduce the computation to $L(h, p', q')$ such that $\gcd(p', q') = 1$ and, without loss of generality, $1 < p' < q'$. This takes $\mathcal{O}(\log p + \log q)$ time. If $p' = 2$, we use Fact 15, while for $p' > 2$ we rely on the characterization of Theorem 13, using Lemmas 9 and 12 for computing L^s and L^d , respectively. The values $\tilde{G}(h', p', q')$ form a sorted sequence of multiples of p' and q' , but not of $p'q'$. Hence, it takes $\mathcal{O}(h)$ time to generate them for $0 \leq h' \leq h$. The overall running time is $\mathcal{O}(h + \log p + \log q)$. \square

5 Faster Algorithm for Evaluating L

A more efficient algorithm for evaluating L relies on the theory of continued fractions; we refer to [15, 22] for a self-contained yet compact introduction. A finite continued fraction is a sequence $[\gamma_0; \gamma_1, \dots, \gamma_m]$, where $\gamma_0, m \in \mathbb{Z}_{\geq 0}$ and $\gamma_i \in \mathbb{Z}_{\geq 1}$ for $1 \leq i \leq m$. We associate it with the following rational number:

$$[\gamma_0; \gamma_1, \dots, \gamma_m] = \gamma_0 + \cfrac{1}{\gamma_1 + \cfrac{1}{\ddots + \cfrac{1}{\gamma_m}}}.$$

Depending on the parity of m , we distinguish odd and even continued fractions. Often, an improper continued fraction $[;] = \frac{1}{0}$ is also introduced and assumed to be odd. Each positive rational number has exactly two representations as a continued fraction, one as an even continued fraction, and one as an odd continued fraction. For example, $\frac{5}{7} = [0; 1, 2, 2] = [0; 1, 2, 1, 1]$.

Consider a continued fraction $[\gamma_0; \gamma_1, \dots, \gamma_m]$. Its *convergents* are continued fractions of the form $[\gamma_0; \gamma_1, \dots, \gamma_{m'}]$ for $0 \leq m' < m$, and $[;] = \frac{1}{0}$. The *semiconvergents* also include continued fractions of the form $[\gamma_0; \gamma_1, \dots, \gamma_{m'-1}, \gamma'_{m'}]$, where $0 \leq m' \leq m$ and $0 < \gamma'_{m'} < \gamma_{m'}$. The two continued fractions representing a positive rational number have the same semiconvergents.

Example 17. The semiconvergents of $[0; 1, 2, 2] = \frac{5}{7} = [0; 1, 2, 1, 1]$ are $[;] = \frac{1}{0}$, $[0;] = \frac{0}{1}$, $[0; 1] = \frac{1}{1}$, $[0; 1, 1] = \frac{1}{2}$, $[0; 1, 2] = \frac{2}{3}$, and $[0; 1, 2, 1] = \frac{3}{4}$.

Semiconvergents of $\frac{p}{q}$ can be generated using the (slow) *continued fraction algorithm*, which produces a sequence of *Farey pairs* $(\frac{a}{b}, \frac{c}{d})$ such that $\frac{a}{b} < \frac{p}{q} < \frac{c}{d}$.

Algorithm 1: Continued fraction algorithm for a rational number $\frac{p}{q} > 0$

```

 $(\frac{a}{b}, \frac{c}{d}) := (\frac{0}{1}, \frac{1}{0});$ 
while true do
    Report a Farey pair  $(\frac{a}{b}, \frac{c}{d})$ ;
    if  $\frac{a+c}{b+d} < \frac{p}{q}$  then  $\frac{a}{b} := \frac{a+c}{b+d}$ ;
    else if  $\frac{a+c}{b+d} = \frac{p}{q}$  then break;
    else  $\frac{c}{d} := \frac{a+c}{b+d}$ ;

```

Example 18. For $\frac{p}{q} = \frac{5}{7}$, the Farey pairs are $(\frac{0}{1}, \frac{1}{0}) \rightsquigarrow (\frac{0}{1}, \frac{1}{1}) \rightsquigarrow (\frac{1}{2}, \frac{1}{1}) \rightsquigarrow (\frac{2}{3}, \frac{1}{1}) \rightsquigarrow (\frac{2}{3}, \frac{3}{4})$. The process terminates at $\frac{2+3}{3+4} = \frac{5}{7}$.

Consider the set $\mathcal{F} = \{\frac{a}{b} : a, b \in \mathbb{Z}_{\geq 0}, \gcd(a, b) = 1\}$ of reduced fractions (including $\frac{1}{0}$). We denote $\mathcal{F}_k = \{\frac{a}{b} \in \mathcal{F} : a + b \leq k\}$ and, for each $x \in \mathbb{R}_+$:

$$\text{Left}_k(x) = \max\{a \in \mathcal{F}_k : a \leq x\} \quad \text{and} \quad \text{Right}_k(x) = \min\{a \in \mathcal{F}_k : a \geq x\}.$$

We say that $\frac{a}{b} < x$ is a *best left approximation* of x if $\frac{a}{b} = \text{Left}_k(x)$ for some $k \in \mathbb{Z}_{\geq 0}$. Similarly, $\frac{c}{d} > x$ is a *best right approximation* of x if $\frac{c}{d} = \text{Right}_k(x)$.

Example 19. We have $\mathcal{F}_7 = (\frac{0}{1}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{1}{1}, \frac{4}{3}, \frac{3}{2}, \frac{2}{1}, \frac{5}{2}, \frac{3}{1}, \frac{4}{1}, \frac{5}{1}, \frac{6}{1}, \frac{1}{0})$. Here, $\text{Left}_7(\frac{5}{7}) = \frac{2}{3}$ and $\text{Right}_7(\frac{5}{7}) = \frac{3}{4}$ are best approximations of $\frac{5}{7}$.

We heavily rely on the following extensive characterization of semiconvergents:

Fact 20 ([15], [21, Theorem 3.3], [22, Theorem 2]). *Let $\frac{p}{q} \in \mathcal{F} \setminus \{\frac{1}{0}, \frac{0}{1}\}$. The following conditions are equivalent for reduced fractions $\frac{a}{b} < \frac{p}{q}$:*

- (a) the Farey process for $\frac{p}{q}$ generates a pair $(\frac{a}{b}, \frac{c}{d})$ for some $\frac{c}{d} \in \mathcal{F}$,
- (b) $\frac{a}{b}$ is an even semiconvergent of $\frac{p}{q}$,
- (c) $\frac{a}{b}$ is a best left approximation of $\frac{p}{q}$,
- (d) $b = \lfloor \frac{aq}{p} \rfloor + 1$ and $aq \bmod p > iq \bmod p$ for $0 \leq i < a$.

By symmetry, the following conditions are equivalent for reduced fractions $\frac{c}{d} > \frac{p}{q}$:

- (a) the Farey process for $\frac{p}{q}$ generates a pair $(\frac{a}{b}, \frac{c}{d})$ for some $\frac{a}{b} \in \mathcal{F}$,
- (b) $\frac{c}{d}$ is an odd semiconvergent of $\frac{p}{q}$,
- (c) $\frac{c}{d}$ is a best right approximation of $\frac{p}{q}$,
- (d) $c = \lfloor \frac{dp}{q} \rfloor + 1$ and $dp \bmod q > ip \bmod q$ for $0 \leq i < d$.

Example 21. For $\frac{p}{q} = \frac{5}{7}$, the prefix maxima of $(iq \bmod p)_{i=0}^{p-1} = (0, 2, 4, 1, 3)$ are attained for $i = 0, 1, 2$ (numerators of $\frac{0}{1}, \frac{1}{2}, \frac{2}{3}$) while the prefix maxima of $(ip \bmod q)_{i=0}^{q-1} = (0, 5, 3, 1, 6, 4, 2)$ are attained for $i = 0, 1, 4$ (denominators of $\frac{1}{0}, \frac{1}{1}, \frac{3}{4}$).

Due to Fact 20, the best approximations can be efficiently computed using the *fast* continued fraction algorithm; see [22].

Corollary 22. *Given $\frac{p}{q} \in \mathcal{F}$ and a positive integer k , $1 \leq k < p + q$, the values $\text{Left}_k(\frac{p}{q})$ and $\text{Right}_k(\frac{p}{q})$ can be computed in $\mathcal{O}(\log k)$ time.*

In the full version [16], we characterize the function L^d as follows.

Lemma 23. *Let $p, q > 2$ be relatively prime integers and let $h < p + q - 3$. If $\frac{a}{b} = \text{Left}_{h+3}(\frac{p}{q})$ and $\frac{c}{d} = \text{Right}_{h+3}(\frac{p}{q})$, then, assuming $G(-1, p, q) = 0$:*

$$L^d(h, p, q) = \begin{cases} \tilde{G}(a + b - 2, p, q) + \tilde{G}(c + d - 2, p, q) & \text{if } a + b + c + d = h + 4, \\ \tilde{G}(h + 2, p, q) & \text{otherwise.} \end{cases}$$

Lemma 23 applies to $h < p + q - 3$; the following fact lets us deal with $h \geq p + q - 3$. It appeared in [5], but in the full version [16] we provide an alternative proof.

Fact 24 ([5, Theorem 4]). *Let p, q be relatively prime positive integers. For each $h \geq 0$, we have*

$$L^d(h, p, q) = L^d(h \bmod (p + q - 2), p, q) + \left\lfloor \frac{h}{p+q-2} \right\rfloor \cdot pq.$$

Moreover, $L^d(p + q - 3, p, q) = pq$.

Theorem 25. *Given integers $p, q \geq 1$ such that $\gcd(p, q) \notin \{p, q\}$ and an integer $h \geq 0$, the value $L(h, p, q)$ can be computed in $\mathcal{O}(\log p + \log q)$ time.*

Proof. We proceed as in the proof of Corollary 16, except that we apply Fact 24 and Lemma 23 to compute $L^d(h, p, q)$. Fact 24 reduces the problem to determining $L^d(h', p, q)$, where $h' = h \bmod (p + q - 2)$. We use Corollary 22 to compute $\text{Left}_{h'+3}(\frac{p}{q})$ and $\text{Right}_{h'+3}(\frac{p}{q})$ in $\mathcal{O}(\log h')$ time. The values $\tilde{G}(r, p, q)$ can be determined in $\mathcal{O}(\log r)$ time using binary search (restricted to multiples of p or q). The overall running time for $L^d(h, p, q)$ is $\mathcal{O}(\log h') = \mathcal{O}(\log p + \log q)$, so for $L(h, p, q)$ it is also $\mathcal{O}(\log p + \log q)$. \square

6 Closed-Form Formula for $L(h, \cdot, \cdot)$

In this section we show how to compute a compact representation of the function $L(h, \cdot, \cdot)$ in $\mathcal{O}(h \log h)$ time. We start with such representations for \tilde{G} and L^d .

Assume that $h < p + q - 3$. For $0 < i \leq h + 4$, let us define fractions

$$l_i = \frac{i-1}{h+4-i}, \quad m_i = \frac{i}{h+4-i},$$

called the h -special points and the h -middle points, respectively. As proved in the full version [16], the function \tilde{G} can now be expressed as follows; see Fig. 2.

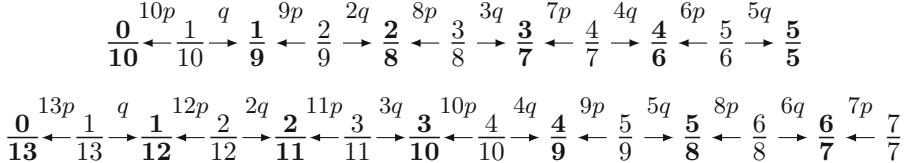


Fig. 2. Graphical representations of the closed-form formulae for $\tilde{G}(9, p, q)$ (above) and $\tilde{G}(12, p, q)$ (below) for $p < q$: partitions of $[0, 1]$ into intervals w.r.t. p/q and linear functions of p and q for each interval. The respective special points are shown in bold.

Lemma 26. *If $\gcd(p, q) = 1$ and $h < p + q - 3$, then*

$$\tilde{G}(h+2, p, q) = \begin{cases} (h+4-i) \cdot p & \text{if } l_i \leq \frac{p}{q} \leq m_i, \\ i \cdot q & \text{if } m_i \leq \frac{p}{q} \leq l_{i+1}. \end{cases}$$

Combined with Lemma 23, Lemma 26 yields a closed-form formula for L^d , which we formally derive in the full version [16]. Note that for each i , we have $l_i \leq \text{Left}_{h+3}(m_i) \leq m_i \leq \text{Right}_{h+3}(m_i) \leq l_{i+1}$, but none of the inequalities is strict in general. In particular, $\text{Left}_{h+3}(m_i) = m_i = \text{Right}_{h+3}(m_i)$ if $\gcd(i, h+4-i) > 1$.

Corollary 27. *Let p, q be relatively prime positive integers and let $h \leq p + q - 3$ be a non-negative integer. Suppose that $l_i \leq \frac{p}{q} \leq l_{i+1}$ and define reduced fractions $\frac{a_i}{b_i} = \text{Left}_{h+3}(m_i)$ and $\frac{c_i}{d_i} = \text{Right}_{h+3}(m_i)$. Then:*

$$L^d(h, p, q) = \begin{cases} (h+4-i) \cdot p & \text{if } l_i \leq \frac{p}{q} \leq \frac{a_i}{b_i}, \\ a_i q + d_i p & \text{if } \frac{a_i}{b_i} < \frac{p}{q} < \frac{c_i}{d_i}, \\ i \cdot q & \text{if } \frac{c_i}{d_i} \leq \frac{p}{q} \leq l_{i+1}. \end{cases}$$

Theorem 28. *Let $2 < p < q$ be relatively prime and let $4 < h < p + q - 2$. Suppose that $l_i \leq \frac{p}{q} \leq l_{i+1}$ and define reduced fractions $\frac{a_i}{b_i} = \text{Left}_{h+3}(m_i)$ and $\frac{c_i}{d_i} = \text{Right}_{h+3}(m_i)$. Then:*

$$L(h, p, q) = \begin{cases} \lceil \frac{h+1}{2} \rceil p + q - (h+1) \bmod 2 & \text{if } 0 < \frac{p}{q} < 1/\lceil \frac{h}{2} \rceil \text{ else} \\ (h+4-i) \cdot p & \text{if } l_i \leq \frac{p}{q} \leq \frac{a_i}{b_i}, \\ a_i q + d_i p & \text{if } \frac{a_i}{b_i} < \frac{p}{q} < \frac{c_i}{d_i}, \\ i \cdot q & \text{if } \frac{c_i}{d_i} \leq \frac{p}{q} \leq l_{i+1}. \end{cases}$$

This compact representation of $L(h, p, q)$ (see Fig. 3 for an example) for a given h has size $\mathcal{O}(h)$ and can be computed in time $\mathcal{O}(h \log h)$.

Proof. The formula follows from the formulae for L^s (Lemma 9) and L^d (Corollary 27) combined using Theorem 13. To compute the table for L efficiently, we determine $\frac{a_i}{b_i} = \text{Left}_{h+3}(m_i)$ and $\frac{c_i}{d_i} = \text{Right}_{h+3}(m_i)$ using Corollary 22. \square

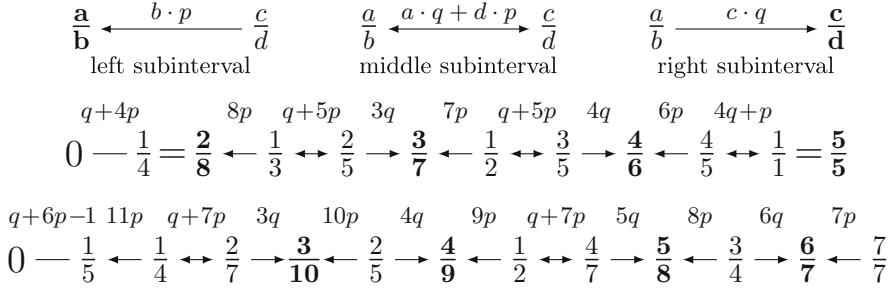


Fig. 3. Graphical representations of the closed-form formulae for $L(7, p, q)$ (middle) and $L(10, p, q)$ (below). Compared to $\tilde{G}(9, p, q)$ and $\tilde{G}(12, p, q)$, respectively, an initial subinterval and several middle subintervals are added. A general pattern for the left, middle, and right subintervals, is presented above. However, the left subinterval $(\frac{1}{5}, \frac{1}{4})$ within $L(10, p, q)$ is an exception because it has been trimmed by the initial interval.

7 Relation to Standard Sturmian Words

For a finite *directive sequence* $\gamma = (\gamma_1, \dots, \gamma_m)$ of positive integers, a Sturmian word $\text{St}(\gamma)$ is recursively defined as X_m , where $X_{-1} = \mathbf{q}$, $X_0 = \mathbf{p}$, and $X_i = X_{i-1}^{\gamma_i} X_{i-2}$ for $1 \leq i \leq m$; see [17, Chapter 2]. We classify directive sequences γ (and the Sturmian words $\text{St}(\gamma)$) into *even* and *odd* based on the *parity* of m .

Observation 29. *Odd Sturmian words of length at least 2 end with pq, while even Sturmian words of length at least 2 end with qp.*

For a directive sequence $\gamma = (\gamma_1, \dots, \gamma_m)$, we define $\text{fr}(\gamma) = [0; \gamma_1, \dots, \gamma_m]$.

Fact 30 [17, Proposition 2.2.24]. *If $\text{fr}(\gamma) = \frac{p}{q}$, then $\text{St}(\gamma)$ contains p characters \mathbf{q} and q characters \mathbf{p} .*

Example 31. We have $\frac{5}{7} = [0; 1, 2, 2] = [0; 1, 2, 1, 1]$, so the Sturmian words with 5 \mathbf{q} 's and 7 \mathbf{p} 's are: $\text{St}(1, 2, 2) = \mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}$ and $\text{St}(1, 2, 1, 1) = \mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}\mathbf{p}\mathbf{q}\mathbf{p}$.

For relatively prime integers $1 < p < q$, we define $\text{St}_{p,q}$ as a Sturmian word with $\text{fr}(\gamma) = \frac{p}{q}$. Note that we always have two possibilities for $\text{St}_{p,q}$ (one odd and one

Table 3. The Sturmian words $\text{St}_{p,q}$ for $p = 5$ and $q = 7$ and the corresponding values of $\tilde{G}(i, p, q)$ for $i < p + q - 2$.

	0	1	2	3	4	5	6	7	8	9	10	11
$\text{St}_{p,q}[i]$	\mathbf{p}	\mathbf{q}	\mathbf{p}	\mathbf{q}	\mathbf{p}	\mathbf{p}	\mathbf{q}	\mathbf{p}	\mathbf{q}	\mathbf{p}	\mathbf{p}/\mathbf{q}	\mathbf{q}/\mathbf{p}
$\tilde{G}(i, p, q)$	p	q	$2p$	$2q$	$3p$	$4p$	$3q$	$5p$	$4q$	$6p$		
$\tilde{G}(i, p, q)$	5	7	10	14	15	20	21	25	28	30		

even), but they differ in the last two positions only. In fact, the first $p + q - 2$ characters of $\text{St}_{p,q}$ are closely related to the values $\tilde{G}(i, p, q)$.

Fact 32 [17, Proposition 2.2.15]. *Let $1 < p < q$ be relatively prime integers. If $i \leq p + q - 3$, then*

$$\text{St}_{p,q}[i] = \begin{cases} p & \text{if } p \mid \tilde{G}(i, p, q), \\ q & \text{if } q \mid \tilde{G}(i, p, q). \end{cases}$$

As a result, the values $\tilde{G}(i, p, q)$ can be derived from $\text{St}_{p,q}$; see Table 3. The following theorem, formally proved in the full version [16], can be seen as a restatement of Lemma 23 in terms of the standard Sturmian words.

Theorem 33. *Let $\text{St}_{p,q}$ be a standard Sturmian word corresponding to $\frac{p}{q}$ and let $0 \leq h < p + q - 3$. If $\text{St}_{p,q}[0..h+3]$ is a Sturmian word, then $L^d(h, p, q) = \tilde{G}(l-2, p, q) + \tilde{G}(r-2, p, q)$, where l, r are the lengths of the longest proper Sturmian prefixes of $\text{St}_{p,q}[0..h+3]$ of different parities, and $\tilde{G}(-1, p, q) = 0$. Otherwise, $L^d(h, p, q) = \tilde{G}(h+2, p, q)$.*

Example 34. Consider a word $\text{St}_{5,7}$ as in Table 3. The lengths of its proper even Sturmian prefixes are 2, 7, whereas the lengths of its proper odd Sturmian prefixes are 1, 3, 5. Hence, $L^d(7, 5, 7) = \tilde{G}(9, 5, 7) = 30$, since $\text{St}_{5,7}[0..10]$ is not a Sturmian word. Moreover, $L^d(8, 5, 7) = \tilde{G}(5, 5, 7) + \tilde{G}(3, 5, 7) = 20 + 14 = 34$, since $\text{St}_{5,7}[0..11] = \text{St}_{5,7}$ is a Sturmian word.

References

1. Bai, H., Franek, F., Smyth, W.F.: The new periodicity lemma revisited. *Discrete Appl. Math.* **212**, 30–36 (2016)
2. Berstel, J., Boasson, L.: Partial words and a theorem of Fine and Wilf. *Theor. Comput. Sci.* **218**(1), 135–141 (1999)
3. Blanchet-Sadri, F., Bal, D., Sisodia, G.: Graph connectivity, partial words, and a theorem of Fine and Wilf. *Inf. Comput.* **206**(5), 676–693 (2008)
4. Blanchet-Sadri, F., Hegstrom, R.A.: Partial words and a theorem of Fine and Wilf revisited. *Theor. Comput. Sci.* **270**(1–2), 401–419 (2002)
5. Blanchet-Sadri, F., Mandel, T., Sisodia, G.: Periods in partial words: an algorithm. *J. Discrete Algorithms* **16**, 113–128 (2012)
6. Blanchet-Sadri, F., Oey, T., Rankin, T.D.: Fine and Wilf’s theorem for partial words with arbitrarily many weak periods. *Int. J. Found. Comput. Sci.* **21**(5), 705–722 (2010)
7. Blanchet-Sadri, F., Simmons, S., Tebbe, A., Veprauskas, A.: Abelian periods, partial words, and an extension of a theorem of Fine and Wilf. *RAIRO - Theor. Inform. Appl.* **47**(3), 215–234 (2013)
8. Castelli, M.G., Mignosi, F., Restivo, A.: Fine and Wilf’s theorem for three periods and a generalization of Sturmian words. *Theor. Comput. Sci.* **218**(1), 83–94 (1999)
9. Constantinescu, S., Ilie, L.: Fine and Wilf’s theorem for abelian periods. *Bull. EATCS* **89**, 167–170 (2006). <http://eatscs.org/images/bulletin/beatcs89.pdf>

10. Fan, K., Puglisi, S.J., Smyth, W.F., Turpin, A.: A new periodicity lemma. *SIAM J. Discrete Math.* **20**(3), 656–668 (2006)
11. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. *Proc. Am. Math. Soc.* **16**(1), 109–114 (1965)
12. Giancarlo, R., Mignosi, F.: Generalizations of the periodicity theorem of Fine and Wilf. In: Tison, S. (ed.) CAAp 1994. LNCS, vol. 787, pp. 130–141. Springer, Heidelberg (1994). <https://doi.org/10.1007/BFb0017478>
13. Justin, J.: On a paper by Castelli, Mignosi, Restivo. *RAIRO - Theor. Inform. Appl.* **34**(5), 373–377 (2000)
14. Karhumäki, J., Puzynina, S., Saarela, A.: Fine and Wilf's theorem for k -abelian periods. *Int. J. Found. Comput. Sci.* **24**(7), 1135–1152 (2013)
15. Khinchin, A.Y.: Continued Fractions. Dover Publications, New York (1997)
16. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: On periodicity lemma for partial words. ArXiv preprint. <http://arxiv.org/abs/1801.01096>
17. Lothaire, M.: Algebraic Combinatorics on Words: Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge (2002)
18. Manea, F., Mercaş, R., Nowotka, D.: Fine and Wilf's theorem and pseudo-repetitions. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 668–680. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32589-2_58
19. Mignosi, F., Restivo, A., Silva, P.V.: On Fine and Wilf's theorem for bidimensional words. *Theor. Comput. Sci.* **292**(1), 245–262 (2003)
20. Mignosi, F., Shallit, J., Wang, M.: Variations on a theorem of Fine & Wilf. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44683-4_45
21. van Ravenstein, T.: The three gap theorem (Steinhaus conjecture). *J. Aust. Math. Soc.* **45**(3), 360–370 (1988)
22. Richards, I.: Continued fractions without tears. *Math. Mag.* **54**(4), 163–171 (1981)
23. Shur, A.M., Gamzova, Y.V.: Partial words and the interaction property of periods. *Izv. Math.* **68**, 405–428 (2004)
24. Shur, A.M., Konovalova, Y.V.: On the periods of partial words. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 657–665. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44683-4_57
25. Smyth, W.F., Wang, S.: A new approach to the periodicity lemma on strings with holes. *Theor. Comput. Sci.* **410**(43), 4295–4302 (2009)
26. Tijdeman, R., Zamboni, L.Q.: Fine and Wilf words for any periods II. *Theor. Comput. Sci.* **410**(30–32), 3027–3034 (2009)



Measuring Closeness Between Cayley Automatic Groups and Automatic Groups

Dmitry Berdinsky^{1,2} and Phongpitak Trakuldit^{1,2}

¹ Department of Mathematics, Faculty of Science,
Mahidol University, Bangkok, Thailand
berdinsky@gmail.com, p.trakuldit@gmail.com

² Centre of Excellence in Mathematics, Commission on Higher Education,
Bangkok, Thailand

Abstract. In this paper we introduce a way to estimate a level of closeness of Cayley automatic groups to the class of automatic groups using a certain numerical characteristic. We characterize Cayley automatic groups which are not automatic in terms of this numerical characteristic and then study it for the lamplighter group, the Baumslag–Solitar groups and the Heisenberg group.

Keywords: Automatic groups · Cayley automatic groups
Automatic structures · Numerical characteristics of groups
Lamplighter group · Heisenberg group · Baumslag–Solitar groups

1 Introduction

Cayley automatic groups had been introduced by Kharlampovich, Khoussainov and Miasnikov as a generalization of automatic groups [9]. They are all finitely generated groups for which their directed labeled Cayley graphs are finite automata presentable structures (automatic structures) [10]. In particular, Cayley automatic groups include all automatic groups in the sense of Thurston [7]. Cayley automatic groups inherit the key algorithmic properties of automatic groups: the first order theory for a directed labeled Cayley graph of a Cayley automatic groups is decidable, the word problem in a Cayley automatic group is decidable in quadratic time [9]. The set of Cayley automatic groups comprise all finitely generated nilpotent groups of nilpotency class at most two [9], the Baumslag–Solitar groups [2] and all fundamental groups of 3–dimensional manifolds. This shows that Cayley automatic groups include important classes of groups.

In this paper we introduce the classes of Cayley automatic groups \mathcal{B}_f defined by non-decreasing and non-negative functions f . Informally speaking, for any given group $G \in \mathcal{B}_f$, the function f shows an upper bound for a level of closeness of the group G to the class of automatic groups. In particular, if f is identically equal to zero, then G must be automatic. So, similarly to a growth function, one

can consider f as a numerical characteristic of the group G . Studying numerical characteristics of groups and relations between them is an important topic in group theory [13]. In this paper we initiate study of this numerical characteristic. We first characterize non-automatic groups in terms of this characteristic. Then we study this characteristic for some non-automatic groups, namely, the lamplighter group $\mathbb{Z}_2 \wr \mathbb{Z}$, the Baumslag–Solitar groups $BS(p, q)$, with $1 \leq p < q$, and the Heisenberg group $\mathcal{H}_3(\mathbb{Z})$. Another motivation to introduce this numerical characteristic is to address the problem of finding characterization for Cayley automatic groups by studying classes \mathcal{B}_f for some functions f .

The paper is organized as follows. In Sect. 2 we recall the definitions of automatic and Cayley automatic groups. Then we give the definition of the classes of Cayley automatic groups \mathcal{B}_f and show that it does not depend on the choice of generators. In Sect. 3 we give a characterization of non-automatic groups by showing that if $G \in \mathcal{B}_f$ is non-automatic, then f must be unbounded. In Sects. 4 and 5 we show that the Baumslag–Solitar groups $BS(p, q)$, with $1 \leq p < q$, and the lamplighter group $\mathbb{Z}_2 \wr \mathbb{Z}$ are in the class \mathcal{B}_i , where i is the identity function: $i(n) = n$. Moreover, we show that these groups cannot be elements of any class \mathcal{B}_f , if the function f is less than i in coarse sense (see Definition 3). In Sect. 6 we show that the Heisenberg group $\mathcal{H}_3(\mathbb{Z})$ is in the class \mathcal{B}_e , where e is the exponential function: $e(n) = \exp(n)$. We then show that $\mathcal{H}_3(\mathbb{Z})$ cannot be an element of any class \mathcal{B}_f , if f is less than the cubic root function $\sqrt[3]{n}$ in coarse sense. Section 7 concludes the paper.

2 Preliminaries

Let G be a finitely generated infinite group. Let $A \subseteq G$ be a finite generating set of the group G . We denote by S the set $S = A \cup A^{-1}$, where A^{-1} is the set of the inverses of elements of A . For given elements $g_1, g_2 \in G$, we denote by $d_A(g_1, g_2)$ the distance between the elements g_1 and g_2 in the Cayley graph $\Gamma(G, A)$. Similarly, we denote by $d_A(g) = d_A(e, g)$ the word length of g with respect the generating set A . We denote by $\pi : S^* \rightarrow G$ the canonical mapping which sends every word $w \in S^*$ to the corresponding group element $\pi(w) = \overline{w} \in G$. We assume that the reader is familiar with the notion of finite automata and regular languages. For a given finite alphabet Σ we put $\Sigma_\diamond = \Sigma \cup \{\diamond\}$, where $\diamond \notin \Sigma$ is a padding symbol. The convolution of n words $w_1, \dots, w_n \in \Sigma^*$ is the string $w_1 \otimes \dots \otimes w_n$ of length $\max\{|w_1|, \dots, |w_n|\}$ over the alphabet Σ_\diamond^n defined as follows. The k th symbol of the string is $(\sigma_1, \dots, \sigma_n)^\top$, where σ_i , $i = 1, \dots, n$ is the k th symbol of w_i if $k \leq |w_i|$ and \diamond otherwise. The convolution $\otimes R$ of a n -ary relation $R \subseteq \Sigma^{*n}$ is defined as $\otimes R = \{w_1 \otimes \dots \otimes w_n | (w_1, \dots, w_n) \in R\}$. We recall that a n -tape synchronous finite automaton is a finite automaton over the alphabet $\Sigma_\diamond^n \setminus \{(\diamond, \dots, \diamond)\}$. We say that a n -ary relation $R \subseteq \Sigma^{*n}$ is regular if $\otimes R$ is accepted by a n -tape synchronous finite automaton. Below we give a definition of automatic groups in the sense of Thurston [7].

Definition 1. We say that G is automatic if there exists a regular language $L \subseteq S^*$ such that $\varphi = \pi|_L : L \rightarrow G$ is a bijection and for every $a \in A$ the binary relation $R_a = \{(\varphi^{-1}(g), \varphi^{-1}(ga)) | g \in G\} \subseteq L \times L$ is regular.

In Definition 1 we forced φ to be a bijection, so this definition of automatic groups is different from the original one [7, Definition 2.3.1]. However, it can be verified that both definitions are equivalent. We denote by \mathcal{A} the class of all automatic groups. Below we give a definition of Cayley automatic groups [9].

Definition 2. We say that G is Cayley automatic if there exist a regular language $L \subseteq S^*$ and a bijection $\psi : L \rightarrow G$ such that for every $a \in A$ the binary relation $R_a = \{(\psi^{-1}(g), \psi^{-1}(ga)) | g \in G\} \subseteq L \times L$ is regular. We call $\psi : L \rightarrow G$ a Cayley automatic representation of G .

In Definition 2 we forced L to be a language over the alphabet S and ψ to be a bijection, so this definition of Cayley automatic groups is different from the original one [9, Definition 6.4]. However, because the cardinality of S is greater than or equal to 2, it can be verified that both definitions are equivalent. We denote by \mathcal{C} the class of all Cayley automatic groups.

Clearly, $\mathcal{A} \subseteq \mathcal{C}$. However, \mathcal{A} is a proper subset of \mathcal{C} : for example, the lamplighter group, the Baumslag–Solitar groups and the Heisenberg group $\mathcal{H}_3(\mathbb{Z})$ are Cayley automatic, but not automatic. We will refer to \mathbb{N} as the set of all positive integers. We denote by \mathbb{R}^+ the set of all non-negative real numbers. Let \mathfrak{F} be the following set of non-decreasing functions:

$$\mathfrak{F} = \{f : [Q, +\infty) \rightarrow \mathbb{R}^+ | [Q, +\infty) \subseteq \mathbb{N} \wedge \forall n (n \in \text{dom } f \implies f(n) \leq f(n+1))\}.$$

Definition 3. Let $f, h \in \mathfrak{F}$. We say that $h \preceq f$ if there exist positive integers K, M and N such that $[N, +\infty) \subseteq \text{dom } h \cap \text{dom } f$ and $h(n) \leq Kf(Mn)$ for every integer $n \geq N$. We say that $h \asymp f$ if $h \preceq f$ and $f \preceq h$. We say that $h \prec f$ if $h \preceq f$ and $h \not\asymp f$.

Let $G \in \mathcal{C}$ be a Cayley automatic group and $f \in \mathfrak{F}$. Let us choose some finite generating set $A \subseteq G$. For a given language $L \subseteq S^*$ and $n \in \mathbb{N}$ we denote by $L^{\leq n}$ the set of all words of length less than or equal to n from the language L , i.e., $L^{\leq n} = \{w \in L | |w| \leq n\}$.

Definition 4. We say that $G \in \mathcal{B}_f$ if there exist a regular language $L \subseteq S^*$ and a Cayley automatic representation $\psi : L \rightarrow G$ such that for the function $h \in \mathfrak{F}$, defined by the equation

$$h(n) = \max\{d_A(\pi(w), \psi(w)) | w \in L^{\leq n}\}, \quad (1)$$

the inequality $h \preceq f$ holds.

We denote by \mathcal{B}_f the class of all Cayley automatic groups G for which $G \in \mathcal{B}_f$. Proposition 5 below shows that Definition 4 does not depend on the choice of generating set A .

Proposition 5. *Definition 4 does not depend on the choice of generating set.*

Proof. Let $A' \subseteq G$ be another generating set of $G \in \mathcal{B}_f$. We put $S' = A' \cup A'^{-1}$. In order to simplify an exposition of our proof, we will assume that $e \in A'$. Let us represent every element $g \in S$ by a word $w_g \in S'^*$ (i.e., $\pi(w_g) = g$) for which the lengths of the words $|w_g|$ are the same for all $g \in S$. In order to make the lengths $w_g, g \in S$ equal, one can use $e \in S'$ as a padding symbol. Let us canonically extend the mapping $g \mapsto w_g, g \in S$ to the monoid homomorphism $\xi : S^* \rightarrow S'^*$. We remark that the definition of ξ ensures that $\pi(\xi(w)) = \pi(w)$ for $w \in S^*$. For a given Cayley automatic representation $\psi : L \rightarrow G$ for which $h \preceq f$, we construct a new Cayley automatic representation $\psi' : L' \rightarrow G$ as follows. We put $L' = \xi(L) \subseteq S'^*$ and define a bijection $\psi' : L' \rightarrow G$ as $\psi' = \psi \circ \tau$, where $\tau = (\xi|_L)^{-1}$. It can be seen that ψ' is a Cayley automatic representation of G . Furthermore, for the function $h' \in \mathfrak{F}$ defined by (1) with respect to ψ' we obtain that $h' \preceq h$ which implies that $h' \preceq f$. This proof can be generalized for the case when $e \notin A'$.

We denote by $\mathbf{z} \in \mathfrak{F}$ the zero function: $\mathbf{z}(n) = 0$ for all $n \in \mathbb{N}$. By Definition 4, we have that $\mathcal{B}_{\mathbf{z}} = \mathcal{A}$. Proposition 6 below shows some elementary properties of the classes \mathcal{B}_f .

Proposition 6. *If $f \preceq g$, then $\mathcal{A} \subseteq \mathcal{B}_f \subseteq \mathcal{B}_g \subseteq \mathcal{C}$. If $f \asymp g$, then $\mathcal{B}_f = \mathcal{B}_g$.*

Proof. By definition, every group of the class \mathcal{B}_g is Cayley automatic, i.e., $\mathcal{B}_g \subseteq \mathcal{C}$. The inclusion $\mathcal{A} \subseteq \mathcal{B}_f$ follows from the fact that $\mathbf{z} \preceq f$ for every $f \in \mathfrak{F}$. The transitivity of the relation \preceq on \mathfrak{F} implies that if $f \preceq g$, then $\mathcal{B}_f \subseteq \mathcal{B}_g$. The fact that $f \asymp g$ implies $\mathcal{B}_f = \mathcal{B}_g$ is straightforward.

3 Characterizing Non-Automatic Groups

Let G be a Cayley automatic group, $A \subseteq G$ be a finite generating set and $S = A \cup A^{-1}$. Given a word $w \in S^*$, for a non-negative integer t we put $w(t)$ to be the prefix of w of a length t , if $t \leq |w|$, and $w(t) = w$, if $t > |w|$. Following notation from [7], we denote by $\widehat{w} : [0, \infty) \rightarrow \Gamma(G, A)$ the corresponding path in the Cayley graph $\Gamma(G, A)$ defined as follows. If $t \geq 0$ is an integer, then $\widehat{w}(t) = \pi(w(t))$, and \widehat{w} is extended to non-integer values of t by moving along the respective edges with unit speed. Given words $w_1, w_2 \in S^*$ and a constant $C_0 \geq 0$, we say that the paths $\widehat{w_1}$ and $\widehat{w_2}$ are a uniform distance less than or equal to C_0 apart if $d_A(\widehat{w_1}(t), \widehat{w_2}(t)) \leq C_0$ for all non-negative integers t .

Theorem 7 below is a simplified modification of the theorem characterizing automatic groups due to Epstein et al. [7, Theorem 2.3.5]. This theorem follows from the existence of standard automata [7, Definition 2.3.3] for all elements of A . For the existence of standard automata it is enough to assume the solvability of the word problem in G . We recall that for Cayley automatic the word problem in G is decidable [9, Theorem 8.1].

Theorem 7. ([7, Theorem 2.3.5]) Let $L \subseteq S^*$ be a regular language such that $\pi : L \rightarrow G$ is surjective. Assume that there is a constant C_0 such that for every $w_1, w_2 \in L$ and $a \in A$ for which $\pi(w_1)a = \pi(w_2)$, the paths $\widehat{w_1}$ and $\widehat{w_2}$ are a uniform distance less than or equal to C_0 apart. Then G is an automatic group.

Let $d \in \mathfrak{F}$ be any bounded function which is not identically equal to the zero function \mathbf{z} . Although $\mathbf{z} \prec d$, the theorem below shows that the class \mathcal{B}_d does not contain any non-automatic group.

Theorem 8. The class $\mathcal{B}_d = \mathcal{A}$. In particular, if for any function $f \in \mathfrak{F}$ the class \mathcal{B}_f contains a non-automatic group, then f must be unbounded.

Proof. Let us show that $\mathcal{B}_d = \mathcal{A}$. By Proposition 6, we only need to show that $\mathcal{B}_d \subseteq \mathcal{A}$. Assume that $G \in \mathcal{B}_d$. By Definition 4, there exists a Cayley automatic representation $\psi_0 : L_0 \rightarrow G$ for some $L_0 \subseteq S^*$ such that, for the function $h_0(n) = \max\{d_A(\pi(w), \psi_0(w)) | w \in L_0^{\leq n}\}$, $h_0 \preceq d$. This implies that $d_A(\psi_0(w), \pi(w))$ is bounded from above by some constant K_0 for all $w \in L_0$. We put $L_1 = S^{*\leq K_0}$. Let $L = L_0L_1$ be the concatenation of L_0 and L_1 . The language L is regular. For any given $g \in G$, $d_A(\pi(\psi_0^{-1}(g)), g) \leq K_0$. This implies that there is a word $u \in L_1$ such that, for the concatenation $w = \psi_0^{-1}(g)u$, $\pi(w) = g$. Therefore, the map $\pi : L \rightarrow G$ is surjective. Let $w_1, w_2 \in L$ be some words for which $\pi(w_1)a = \pi(w_2)$, $a \in A$. There exist words $v_1, v_2 \in L_0$ and $u_1, u_2 \in L_1$ for which $w_1 = v_1u_1$ and $w_2 = v_2u_2$. We obtain that $d_A(\psi_0(v_1), \psi_0(v_2)) \leq d_A(\pi_0(v_1), \pi_0(v_2)) + 2K_0 \leq d_A(\pi(w_1), \pi(w_2)) + 2K_0 + 2K_0 \leq 4K_0 + 1$. That is, there exists $g \in G$, for which $d_A(g) \leq 4K_0 + 1$, such that $\psi_0(v_1)g = \psi_0(v_2)$. The pair (v_1, v_2) is accepted by some two-tape synchronous automaton M_g . Let N_g be the number of states of M_g . Given a non-negative integer t , there exist words $p_1, p_2 \in S^*$, for which the lengths $|p_1|, |p_2|$ are bounded from above by N_g , such that the pair $(v_1(t)p_1, v_2(t)p_2)$ is accepted by M_g ; in particular, $v_1(t)p_1, v_2(t)p_2 \in L_0$. We obtain that $d_A(\pi(v_1(t)), \pi(v_2(t))) \leq d_A(\pi(v_1(t)p_1), \pi(v_2(t)p_2)) + |p_1| + |p_2| \leq d_A(\psi_0(v_1(t)p_1), \psi_0(v_2(t)p_2)) + 2K_0 + 2N_g \leq d_A(g) + 2K_0 + 2N_g \leq 6K_0 + 2N_g + 1$. Therefore, $d_A(\widehat{w_1(t)}, \widehat{w_2(t)}) = d_A(\pi(w_1(t)), \pi(w_2(t))) \leq d_A(\pi(v_1(t)), \pi(v_2(t))) + 2K_0 \leq 8K_0 + 2N_g + 1$. There are only finitely many g for which $d_A(g) \leq 4K_0 + 1$, so N_g can be bound by some constant N_0 . Thus, for $C_0 = 8K_0 + 2N_0 + 1$, we obtain that $d_A(\widehat{w_1(t)}, \widehat{w_2(t)}) \leq C_0$, that is, the paths $\widehat{w_1}$ and $\widehat{w_2}$ are a uniform distance C_0 apart. By Theorem 7, the group G is automatic. The second statement of the theorem is straightforward.

4 The Baumslag–Solitar Groups

Let us consider the Baumslag–Solitar groups $BS(p, q) = \langle a, t | ta^pt^{-1} = a^q \rangle$ with $1 \leq p < q$. These groups are not automatic due to Epstein et al. [7, Section 7.4], but they are Cayley automatic [2, Theorem 3]. The Cayley automatic representations of the Baumslag–Solitar groups constructed in [2, Theorem 3] use the normal form obtained from representing these groups as the HNN extensions [2, Corollary 2]. In [5, Theorem 3.2] Burillo and Elder provide the metric estimates

for the groups $BS(p, q)$ using the normal form [5, Lemma 3.1]. We note that the normal forms shown in [2, Corollary 2] and [5, Lemma 3.1], up to changing n to p and m to q , are the same. This normal form is shown in the following proposition.

Proposition 9. *Any element $g \in BS(p, q)$ for $1 \leq p \leq q$ can be written uniquely as $g = \tilde{w}(a, t)a^k$, where $\tilde{w}(a, t) \in \{t, at, \dots, a^{q-1}t, t^{-1}, at^{-1}, \dots, a^{p-1}t^{-1}\}^*$ is freely reduced and $k \in \mathbb{Z}$.*

Let us now describe a modification of the Cayley automatic representation of $BS(p, q)$ constructed in [2, Theorem 3.2] which is compatible with Definition 2. We put $a_1 = a, \dots, a_{q-1} = a^{q-1}$. Let $A = \{a_0, a_1, \dots, a_{q-1}, t\}$ and $S = A \cup A^{-1} = \{e, a_1, a_2, \dots, a_{q-1}, a_1^{-1}, \dots, a_{q-1}^{-1}, t, t^{-1}\}$. Given an element $g = \tilde{w}(a, t)a^k \in BS(p, q)$, we construct the word $w = uv$ which is the concatenation of two words $u, v \in S^*$ defined as follows. The word $u \in \{t, t^{-1}, a_1, \dots, a_{q-1}\}^*$ is obtained from the corresponding word $\tilde{w}(a, t)$ by changing the subwords $at^\epsilon, \dots, a^{q-1}t^\epsilon$ to the subwords $a_1t^\epsilon, \dots, a_{q-1}t^\epsilon$, respectively, where $\epsilon = +1$ or $\epsilon = -1$. The word v is obtained from the q -ary representation of $|k|$ by changing the 0 to e and $1, \dots, q-1$ to a_1, \dots, a_{q-1} and $a_1^{-1}, \dots, a_{q-1}^{-1}$, if $k \geq 0$ and $k < 0$, respectively. The set of all such words w is a regular language $L \subseteq S^*$. Thus, we have constructed the bijection $\psi : L \rightarrow BS(p, q)$. By [2, Theorem 3.2], ψ provides a Cayley automatic representation of $BS(p, q)$. It is worth noting that if $g \in BS(p, q)$ is an element for which $k = 0$, then for $w = \psi^{-1}(g)$ we obtain that $\psi(w) = \pi(w)$. Let $\tilde{A} = \{a, t\}$. We have the following metric estimates for the groups $BS(p, q)$.

Theorem 10. ([5, Theorem 3.2]) *There exist constants $C_1, C_2, D_1, D_2 > 0$ such that for every element $g \in BS(p, q)$ for $1 \leq p < q$ written as $\tilde{w}(a, t)a^k$, we have: $C_1(|\tilde{w}| + \log(|k| + 1)) - D_1 \leq d_{\tilde{A}}(g) \leq C_2(|\tilde{w}| + \log(|k| + 1)) + D_2$.*

It follows from Theorem 10 that there exist constants $C'_1, C'_2, D'_1, D'_2 > 0$ such that for every element $g \in BS(p, q)$ and for the corresponding word $\psi^{-1}(g) = uv$ we have

$$C'_1(|u| + |v|) - D'_1 \leq d_A(g) \leq C'_2(|u| + |v|) + D'_2. \quad (2)$$

Theorem 11. *Given p and q with $1 \leq p < q$, the Baumslag–Solitar group $BS(p, q) \in \mathcal{B}_i$. Moreover, for any $f \prec i$, $BS(p, q) \notin \mathcal{B}_f$.*

Proof. For given p and q with $1 \leq p < q$ let us consider the Cayley automatic representation $\psi : L \rightarrow BS(p, q)$ constructed above. Let h be the function given by (1) with respect to this Cayley automatic representation. We will show that $h \preceq i$ (in fact one can verify that $h \asymp i$). Let $w = uv \in L^{\leq n}$ and $g = \psi(w)$ be the corresponding group element of $BS(p, q)$. By (2), there exists a constant C such that $d_A(g) \leq C(|u| + |v|) = C|w|$. Therefore, $d_A(\pi(w), \psi(w)) \leq n + d_A(g) \leq (C+1)n$. Therefore, $h \preceq i$ which implies that $BS(p, q) \in \mathcal{B}_i$. Let us show now the second statement of the theorem. Suppose that $BS(p, q) \in \mathcal{B}_f$ for some $f \prec i$. Then there exists a Cayley automatic representation $\psi' : L' \rightarrow BS(p, q)$ for

which $h' \preceq f$, where h' is given by (1). We have $h' \prec i$. We recall that for a group $\langle X|R \rangle$ given by a set of generators X and a set of relators R the Dehn function is given by $D(n) = \max_{u \in U_n} \{\text{area}(u)\}$, where $U_n = \{u \in (X \cup X^{-1})^* | \pi(u) = e \wedge |u| \leq n\}$ is the set of words of the length at most n representing the identity of the group $\langle X|R \rangle$ and $\text{area}(u)$ is the combinatorial area of u which is the minimal k for which $u = \prod_{i=1}^k v_i r_i^{\pm 1} v_i^{-1}$ in the free group $F(X)$, where $r_i \in R$. Let $w \in \{a, a^{-1}, t, t^{-1}\}^*$ be a word representing the identity in $BS(p, q)$ for which $|w| \leq n$. The word w corresponds to a loop in the Cayley graph $BS(p, q)$ with respect to the generators a, t . Repeating an argument of [7, Theorem 2.3.12] (showing that the Dehn function for an automatic group is at most quadratic), it can be seen that the loop w can be subdivided into at most $K_0 n^2$ loops of length at most $\ell(n) = 4h'(K_0 n) + K_1$ for some integer constants K_0 and K_1 . Therefore, $D(n) \leq K_0 n^2 D(\ell(n))$ which implies that $D(n) \preceq n^2 D(\ell(n))$. For the group $BS(p, q)$ the Dehn function is at most exponential (see [7, § 7.4]), i.e., $D(n) \leq \lambda^n$ for some constant λ . Therefore, $D(n) \preceq n^2 \lambda^{\ell(n)}$. Clearly, $\ell \preceq h'$ which implies that $\ell \prec i$. Let us show that $n^2 \lambda^{\ell(n)} \prec \epsilon$. It can be seen that $n^2 \lambda^{\ell(n)} \preceq \epsilon$. Assume that $\epsilon \preceq n^2 \lambda^{\ell(n)}$. Then, for all sufficiently large n and some constants K and M we have: $\exp(n) \leq Kn^2 \lambda^{\ell(Mn)}$. This implies that $n - 2 \ln n - \ln K \leq (\ln \lambda) \ell(Mn)$. Clearly, $\frac{n}{2} \leq n - 2 \ln n - \ln K$ for all sufficiently large n , and, therefore, $n \leq (2 \ln \lambda) \ell(Mn)$. This implies that $i \preceq \ell$ which contradicts to the inequality $\ell \prec i$. Thus, $D(n) \preceq n^2 \lambda^{\ell(n)} \prec \epsilon$ which implies that $D(n) \prec \epsilon$. The last inequality contradicts to the fact that for the group $BS(p, q)$ the Dehn function is at least exponential, i.e., $D(n) \geq \mu^n$ for some constant μ (see [7, Sect. 7.4]) which implies that $\epsilon \preceq D(n)$.

5 The Lamplighter Group

The lamplighter group is the wreath product $\mathbb{Z}_2 \wr \mathbb{Z}$ of the cyclic group \mathbb{Z}_2 and the infinite cyclic group \mathbb{Z} . For the definition of the wreath product of groups we refer the reader to [8]. Let t be a generator of the cyclic group $\mathbb{Z} = \langle t \rangle$ and a be the nontirival element of the group \mathbb{Z}_2 . The canonical embeddings of the groups \mathbb{Z}_2 and \mathbb{Z} into the wreath product $\mathbb{Z}_2 \wr \mathbb{Z}$ enable us to consider \mathbb{Z}_2 and \mathbb{Z} as the subgroups of $\mathbb{Z}_2 \wr \mathbb{Z}$. With respect to the generators a and t , the lamplighter group has the presentation $\langle a, t \mid [t^i a t^{-i}, t^j a t^{-j}], a^2 \rangle$. The lamplighter group is not finitely presented [1], and, therefore, it is not automatic due to [7, Theorem 2.3.12].

The elements of the lamplighter group have the following geometric interpretation. Every element of the lamplighter group corresponds to a bi-infinite string of lamps, indexed by integers $i \in \mathbb{Z}$, each of which is either lit or unlit, such that only finite number of lamps are lit, and the lamplighter pointing at the current lamp $i = m$. The identity of the lamplighter group corresponds to the configuration when all lamps are unlit and the lamplighter points at the lamp positioned at the origin $m = 0$. The right multiplication by a changes the state of the current lamp. The right multiplication by t (or t^{-1}) moves the lamplighter to the right $m \mapsto m+1$ (or to the left $m \mapsto m-1$). The elements of the subgroup

$\mathbb{Z} \leqslant \mathbb{Z}_2 \wr \mathbb{Z}$ are the configurations for which all lamps are unlit. For the elements of the subgroup $\mathbb{Z}_2 \leqslant \mathbb{Z}_2 \wr \mathbb{Z}$ all lamps, apart from the one at the origin, are unlit and the lamplighter points at the lamp positioned at the origin, which can be either lit or unlit.

For any given integer $i \in \mathbb{Z}$ we put $a_i = t^i a t^{-i}$. The group element a_i corresponds to the configuration when the lamp at the position i is lit, all other lamps are unlit and the lamplighter points at the origin $m = 0$. Let g be an element of the lamplighter group. The ‘right–first’ and the ‘left–first’ normal forms of g are defined as follows:

$$rf(g) = a_{i_1} a_{i_2} \dots a_{i_k} a_{-j_1} a_{-j_2} \dots a_{-j_l} t^m,$$

$$lf(g) = a_{-j_1} a_{-j_2} \dots a_{-j_l} a_{i_1} a_{i_2} \dots a_{i_k} t^m,$$

where $i_k > \dots > i_2 > i_1 \geqslant 0$, $j_l > \dots > j_1 > 0$ and the lamplighter points at the position m (see [6]). For the element g the lit lamps are at the positions $-j_l, \dots, -j_1, i_1, \dots, i_k$ and the lamplighter points at the position m . In ‘right–first’ normal form the lamplighter moves to the right illuminating the appropriate lamps until it reaches the lamp at the position i_k . Then it moves back to the origin, and then further to the left illuminating the appropriate lamps until it reaches the lamp at the position $-j_l$. After that the lamplighter moves to the position m . Let $A = \{a, t\}$ and $S = \{a, a^{-1}, t, t^{-1}\}$.

Proposition 12. ([6, Proposition 3.2]) *The word length of the element g with respect to the generating set A is given by*

$$d_A(g) = k + l + \min\{2i_k + j_l + |m + j_l|, 2j_l + i_k + |m - i_k|\}.$$

Some Cayley automatic representations of $\mathbb{Z}_2 \wr \mathbb{Z}$ had been obtained in [2, 3, 9]. Let us now construct a new Cayley automatic representation of $\mathbb{Z}_2 \wr \mathbb{Z}$ using the ‘right–first’ normal form which is compatible with Definition 2. For a given element g of the lamplighter group we construct the word $w = u'v'$ which is the concatenation of two words $u', v' \in S^*$. The words u' and v' are obtained from the words u and v , defined below, by canceling adjacent opposite powers of t . Assume first that $m \geqslant 0$.

- Suppose that $\{i_1, \dots, i_k\} = \emptyset$ or $\{i_1, \dots, i_k\} \neq \emptyset$ and $m > i_k$. We put $u = t^{i_1} a t^{-i_1} \dots t^{i_k} a t^{-i_k} t^m aa$. We put $v = t^{-j_1} a t^{j_1} \dots t^{-j_l} a$.
- Suppose that $\{i_1, \dots, i_k\} \neq \emptyset$ and $m \leqslant i_k$. If $m = i_n$ for some $n = 1, \dots, k$, then we put $u = t^{i_1} a t^{-i_1} \dots t^{i_n} a a a t^{-i_n} \dots t^{i_k} a$. Otherwise, either $m < i_1$ or there exists $q = 1, \dots, k - 1$ for which $i_q < m < i_{q+1}$. In the first case we put $u = t^m a a t^{-m} t^{i_1} a t^{-i_1} \dots t^{i_k} a$. In the latter case we put $u = t^{i_1} a t^{-i_1} \dots t^{i_q} a t^{-i_q} t^m a a t^{-m} t^{i_{q+1}} a t^{-i_{q+1}} \dots t^{i_k} a$. The word v is the same as above.

Assume now that $m < 0$.

- Suppose that $\{j_1, \dots, j_l\} = \emptyset$ or $\{j_1, \dots, j_l\} \neq \emptyset$ and $m < -j_l$. We put $v = t^{-j_1} a t^{j_1} \dots t^{-j_l} a t^{j_l} t^m aa$. We put $u = t^{i_1} a t^{-i_1} \dots t^{i_k} a$.

- Suppose that $\{j_1, \dots, j_l\} \neq \emptyset$ and $m \geq -j_l$. If $m = -j_n$ for some $n = 1, \dots, l$, then we put $v = t^{-j_1}at^{j_1} \dots t^{-j_n}aat^{j_n} \dots t^{-j_l}a$. Otherwise, either $m > -j_1$ or there exists $q = 1, \dots, l-1$ for which $-j_q > m > -j_{q+1}$. In the first case we put $v = t^m aat^{-m}t^{-j_1}at^{j_1} \dots t^{-j_l}at^{j_l}$. In the latter case we put $v = t^{-j_1}at^{j_1} \dots t^{-j_q}at^{j_q}t^m aat^{-m}t^{-j_{q+1}}at^{j_{q+1}} \dots t^{-j_l}a$. The word u is the same as above.

Let us show two simple examples. Suppose first that the lit lamps are at the positions $-1, 0, 2$ and the lamplighter is at the position $m = 1$. Then, for the corresponding group element, the word w is $ataatat^{-1}a$. Suppose now that the lit lamps are at the positions $-1, 1$ and the lamplighter is at the position $m = -1$. Then, for the corresponding group element, the word w is $tat^{-1}aaa$. The set of all such words w forms some language $L \subseteq S^*$. Thus, we have constructed the bijection $\psi : L \rightarrow \mathbb{Z}_2 \wr \mathbb{Z}$. It can be verified that L is a regular language and ψ provides a Cayley automatic representation of the lamplighter group in the sense of Definition 2. We note that in the Cayley automatic representation $\psi : L \rightarrow \mathbb{Z}_2 \wr \mathbb{Z}$ constructed above we use the subwords aa and aaa to specify the lamplighter position. We use aa and aaa if the lamp, the lamplighter is pointing at, is unlit and lit, respectively. It is worth noting that if $g \in \mathbb{Z}_2 \wr \mathbb{Z}$ is an element for which all lamps at negative positions $j < 0$ are unlit and $m \geq i_k$, then for $w = \psi^{-1}(g)$ we obtain that $\pi(w) = \psi(w)$. That is, on a certain infinite subset of L the maps π and ψ coincide.

Theorem 13. *The lamplighter group $\mathbb{Z}_2 \wr \mathbb{Z} \in \mathcal{B}_i$. Moreover, for any $f \prec i$, $\mathbb{Z}_2 \wr \mathbb{Z} \notin \mathcal{B}_f$.*

Proof. Let us consider the Cayley automatic representation $\psi : L \rightarrow \mathbb{Z}_2 \wr \mathbb{Z}$ constructed above. Let h be the function given by (1) with respect to the Cayley automatic representation ψ . We will show that $h \preceq i$ (in fact one can verify that $h \asymp i$). For a given n let $w \in L^{\leq n}$ be a word and $g = \psi(w)$ be the corresponding group element of $\mathbb{Z}_2 \wr \mathbb{Z}$. Clearly, we have that $d_A(\pi(w), \psi(w)) \leq n + d_A(g)$. Therefore, it suffices to show that $d_A(g) \leq Cn$ for some constant C . It follows from the construction of $w = \psi^{-1}(g)$ that if $m \geq 0$, then $|w| = k+l+\max\{m, i_k\}+j_l+2$, and if $m < 0$, then $|w| = k+l+\max\{-m, j_l\}+i_k+2$. By Proposition 12, we obtain that $d_A(g) \leq 3|w| \leq 3n$. Therefore, $h \preceq i$ which implies that $\mathbb{Z}_2 \wr \mathbb{Z} \in \mathcal{B}_i$. Let us show the second statement of the theorem. For a given $m > 0$, let R_m be the following set of relations $R_m = \{a^2\} \cup \{[t^i a t^{-i}, t^j a t^{-j}] \mid -m \leq i < j \leq m\}$. We first notice that for any loop $w \in S^*$, $|w| \leq l$ in the lamplighter group $\mathbb{Z}_2 \wr \mathbb{Z}$ the word w can be represented as a product of conjugates of the relations from R_l , i.e., the identity $w = \prod_{i=1}^k v_i r_i^{\pm 1} v_i^{-1}$ holds in the free group $F(A)$ for some $v_i \in S^*$ and $r_i \in R_l$, $i = 1, \dots, k$. Suppose now that $\mathbb{Z}_2 \wr \mathbb{Z} \in \mathcal{B}_f$ for some $f \prec i$. Similarly to Theorem 11, we obtain that then there exists a function $\ell \prec i$ such that any loop w of the length less than or equal to n can be subdivided into loops of the length at most $\ell(n)$. Therefore, for any loop given by a word $w \in S^*$, $|w| \leq n$, the identity $w = \prod_{i=1}^k v_i r_i^{\pm 1} v_i^{-1}$ holds in the free group $F(A)$ for some $v_i \in S^*$ and $r_i \in R_{\ell(n)}$, $i = 1, \dots, k$. In particular, every relation from R_n can be expressed as a product of conjugates of the relations

from $R_{\ell(8n+4)}$ (the longest relation from R_n is $[t^{-n}at^n, t^nat^{-n}]$ which has the length $8n + 4$). However, not every relation from R_n can be expressed as a product of conjugates of the relations from $R_{n-1} \subset R_n$ because the groups $G_n = \langle a, t | R_n \rangle$ and $G_{n-1} = \langle a, t | R_{n-1} \rangle$ are not isomorphic. This implies the inequality $\ell(8n+4) \geq n$ leading to a contradiction with $\ell \prec i$. The fact that $G_n = \langle a, t | R_n \rangle$ and $G_{n-1} = \langle a, t | R_{n-1} \rangle$ are not isomorphic can be shown as follows. The group G_n can be represented as $G_n = \langle a_{-n}, \dots, a_0, \dots, a_n | a_0^2; a_{i-1} = t^{-1}a_i t, i = -(n-1), \dots, n; [a_i, a_j], i, j = -n, \dots, n \rangle$, so G_n is the HNN extension of the base group $\bigoplus_{i=-n}^n \mathbb{Z}_2 = \langle a_{-n}, \dots, a_n | a_i^2, [a_i, a_j] \rangle$ relative to the isomorphism φ_n between the subgroups $A_n, B_n \leq G_n$ generated by $a_{-(n-1)}, \dots, a_n$ and a_{-n}, \dots, a_{n-1} , respectively, for which $\varphi_n : a_i \mapsto a_{i-1}, i = -(n-1), \dots, n$. As a consequence of Britton's lemma [11], we have the property that every finite subgroup of an HNN extension is conjugate to a finite subgroup of its base group. Therefore, assuming that G_{n+1} and G_n are isomorphic, we obtain that $\bigoplus_{i=-(n+1)}^{n+1} \mathbb{Z}_2$ can be embedded into $\bigoplus_{i=-n}^n \mathbb{Z}_2$ which leads to a contradiction.

6 The Heisenberg Group

The Heisenberg group $\mathcal{H}_3(\mathbb{Z})$ is the group of all matrices of the form:

$$\begin{pmatrix} 1 & x & z \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix},$$

where x, y and z are integers. Every element $g \in \mathcal{H}_3(\mathbb{Z})$ corresponds to a triple (x, y, z) . Let s be a group element of \mathcal{H}_3 corresponding to the triple $(1, 0, 0)$, p corresponding to $(0, 1, 0)$, and q corresponding to $(0, 0, 1)$. If g corresponds to a triple (x, y, z) , then gs, gp and gq correspond to the triples $(x+1, y, z)$, $(x, y+1, x+z)$ and $(x, y, z+1)$, respectively. The observation that \mathcal{H}_3 is not an automatic group but its Cayley graph is automatic was first made by Séniergues.

The Heisenberg group \mathcal{H}_3 is isomorphic to the group $\langle s, p, q | s^{-1}p^{-1}sp = q, sq = qs, pq = qp \rangle$, and it can be generated by the elements s and p . The exact distance formula on $\mathcal{H}_3(\mathbb{Z})$ for the generating set $\{s, p\}$ is obtained in [4, Theorem 2.2]. However, for our purposes it is enough to have the metric estimates which the reader can find in [12, Proposition 1.38]. Let $A = \{e, s, p, q\}$ and $S = A \cup A^{-1} = \{e, s, p, q, s^{-1}, p^{-1}, q^{-1}\}$.

Proposition 14. ([12, Proposition 1.38]) *There exist constants C_1 and C_2 such that for an element $g \in \mathcal{H}_3$ corresponding to a triple (x, y, z) we have*

$$C_1(|x| + |y| + \sqrt{|z|}) \leq d_A(g) \leq C_2(|x| + |y| + \sqrt{|z|}).$$

Proof. We first get an upper bound. Every group element $g \in \mathcal{H}_3$ can be represented as $s^n p^m q^l$ corresponding to the triple $(x, y, z) = (n, m, nm + l)$. It can be verified that $s^k p^k s^{-k} p^{-k} = q^{k^2}$. Therefore, the length of q^l is at most $6\sqrt{|l|} \leq 6\sqrt{|z|} + 3|n| + 3|m|$. For $C_2 = 6$ we obtain the required upper bound.

Let us prove now a lower bound. If $d_A(g) = r$ for an element g corresponding to a triple (x, y, z) , then $|x|, |y| \leq r$ and $|z| \leq r + r^2$. For $C_1 = \frac{1}{4}$ we obtain the required lower bound.

Let us construct a Cayley automatic representation of the Heisenberg group \mathcal{H}_3 which is compatible with Definition 2. For a given $g \in \mathcal{H}_3$ corresponding to a triple (x, y, z) we construct the word $w = uv$ which is the concatenation of two words $u, v \in S^*$ constructed as follows. We put $u = p^y$. Let b_x and b_z be the binary representations of the integers $|x|$ and $|z|$ (with the least significant digits first). We put b to be $b_x \otimes b_z$ with the padding symbol \diamond changed to 0. The word b is a word over the alphabet consisting of the symbols $(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}), (\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}), (\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}), (\begin{smallmatrix} 1 \\ 1 \end{smallmatrix})$. Replacing the symbols $(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}), (\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}), (\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}), (\begin{smallmatrix} 1 \\ 1 \end{smallmatrix})$ in b by the words ee, eq, se and sq we obtain a word $b' \in \{e, s, q\}^*$. If $x \geq 0$ and $z \geq 0$, then we put $v = b'$. If $x < 0$ or $z < 0$, then v is obtained from b' by replacing the symbols s and q to the symbols s^{-1} and q^{-1} , respectively. For example, the triple $(3, -3, -4)$ is represented by the word $p^{-1}p^{-1}p^{-1}seseeq^{-1}$. The set of all such words w is a regular language $L \subseteq S^*$. Thus, we have constructed the bijection $\psi : L \rightarrow \mathcal{H}_3$. It can be verified that ψ provides a Cayley automatic representation of the Heisenberg group \mathcal{H}_3 . It is worth noting that if $g \in \mathcal{H}_3$ corresponds to a triple $(0, y, 0)$, then for the word $w = \psi^{-1}(g)$ we have $\psi(w) = \pi(w)$. That is, the maps π and ψ coincide if restricted on the cyclic subgroup $\langle p \rangle \leq \mathcal{H}_3$.

Theorem 15. *The Heisenberg group $\mathcal{H}_3 \in \mathcal{B}_\epsilon$. Moreover, for any $f \prec \sqrt[3]{n}$, $\mathcal{H}_3 \notin \mathcal{B}_f$.*

Proof. Let h be the function given by (1) with respect to the Cayley automatic representation $\psi : L \rightarrow \mathcal{H}_3$ constructed above. We will show that $h \asymp \epsilon$. Although for the first statement of the theorem it is enough to show that $h \preceq \epsilon$, the inequality $\epsilon \preceq h$ guarantees that we cannot get a better result using just the representation ψ . Let $w = uv \in L^{\leq n}$ and $g = \psi(w)$ be the group element of \mathcal{H}_3 corresponding to a triple (x, y, z) . By Proposition 14, there exists a constant C_2 such that $d_A(g) \leq C_2(|x| + |y| + \sqrt{|z|}) \leq C_2(2^{|v|} + |u| + \sqrt{2^{|v|}}) \leq 2C_22^{|u|+|v|} \leq 2C_2 \exp(|w|) \leq 2C_2 \exp(n)$. Therefore, $h \preceq \epsilon$ which implies that $\mathcal{H}_3 \in \mathcal{B}_\epsilon$. Let us show now that $\epsilon \preceq h$. Let $g_i = s^i, i \geq 2$. The length of the corresponding word $w_i = \psi^{-1}(g_i)$ is equal to the doubled length of the binary representation of i . We have $d_A(\pi(w_i), \psi(w_i)) = d_A(\pi(w_i)^{-1}s^i) = d_A(s^{n_i})$ for some positive integer n_i . Clearly, there exists a constant C such that $n_i \geq Ci$. The group element s^{n_i} corresponds to the triple $(n_i, 0, 0)$. By Proposition 14, we have $d_A(s^{n_i}) \geq C_1n_i$. Therefore, there exists a constant $C' > 0$ such that $d_A(s^{n_i}) \geq C'2^{\frac{|w_i|}{2}}$ for all $i \geq 2$. This implies that $\epsilon \preceq h$. Therefore, $h \asymp \epsilon$. Let us show now the second statement of the theorem. Repeating exactly the same argument as used in Theorem 11, we conclude that there exists a function $\ell(n) \prec \sqrt[3]{n}$ for which the inequality $D(n) \preceq n^2D(\ell(n))$ holds, where $D(n)$ is the Dehn function of \mathcal{H}_3 . For the group \mathcal{H}_3 the Dehn function is at most cubic; specifically for the presentation $\mathcal{H}_3 = \langle s, p, q | s^{-1}p^{-1}sp = q, sq = qs, pq = qp \rangle$, $D(n) \leq n^3$ (see [7, § 8.1]). Therefore, $D(n) \preceq n^2\ell(n)^3$. Let us show that $n^2\ell(n)^3 \prec n^3$. It can be

seen that $n^2\ell(n)^3 \preceq n^3$. Assume that $n^3 \preceq n^2\ell(n)^3$. Then, for all sufficiently large n and some constants K and M we have: $n^3 \leq Kn^2\ell(Mn)^3$. This implies that $\sqrt[3]{n} \leq \sqrt[3]{K}\ell(Mn)$. Therefore, $\sqrt[3]{n} \preceq \ell(n)$ which contradicts to the inequality $\ell(n) \prec \sqrt[3]{n}$. Thus, $D(n) \preceq n^2\ell(n)^3 \prec n^3$ which implies that $D(n) \prec n^3$. The last inequality contradicts to the fact that the Dehn function is at least cubic (see [7, § 8.1]) which implies that $n^3 \preceq D(n)$.

7 Discussion

In this paper we proposed a way to measure closeness of Cayley automatic groups to the class of automatic groups. We did this by introducing the classes of Cayley automatic groups \mathcal{B}_f for the functions $f \in \mathfrak{F}$. In Theorem 8 we characterized non-automatic groups by showing that for any such group G in some class \mathcal{B}_f the function f must be unbounded. We studied then the cases of the Baumslag–Solitar groups $BS(p, q)$, $1 \leq p < q$, the lamplighter group and the Heisenberg group \mathcal{H}_3 . In Theorems 11 and 13 we proved that the Baumslag–Solitar groups and the lamplighter group are in the class \mathcal{B}_i and they cannot belong to any class \mathcal{B}_f for which $f \prec i$. For the Heisenberg group \mathcal{H}_3 in Theorem 15 we proved that $\mathcal{H}_3 \in \mathcal{B}_e$, but we could only prove that it cannot belong to any class \mathcal{B}_f for which $f \prec \sqrt[3]{n}$. The following questions are apparent from the results obtained in this paper.

- Is there any unbounded function $f \prec i$ for which the class \mathcal{B}_f contains a non-automatic group?
- Is there any function $f \prec e$ for which $\mathcal{H}_3 \in \mathcal{B}_f$?

Acknowledgments. The authors thank the referees for useful comments.

References

1. Baumslag, G., Solitar, D.: Some two-generator one-relator non-Hopfian groups. *Bull. Am. Math. Soc.* **68**, 199–201 (1962)
2. Berdinsky, D., Khoussainov, B.: On automatic transitive graphs. In: Shur, A.M., Volkov, M.V. (eds.) *DLT 2014. LNCS*, vol. 8633, pp. 1–12. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09698-8_1
3. Berdinsky, D., Khoussainov, B.: Cayley automatic representations of wreath products. *Int. J. Found. Comput. Sci.* **27**(2), 147–159 (2016)
4. Blachere, S.: Word distance on the discrete Heisenberg group. *Colloq. Math.* **95**(1), 21–36 (2003)
5. Burillo, J., Elder, M.: Metric properties of Baumslag–Solitar groups. *Int. J. Algebra Comput.* **25**(5), 799–811 (2015)
6. Cleary, S., Taback, J.: Dead end words in lamplighter groups and other wreath products. *Q. J. Math.* **56**(2), 165–178 (2005)
7. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V.F., Paterson, M.S., Thurston, W.P.: *Word Processing in Groups*. Jones and Barlett Publishers, Boston (1992)

8. Kargapolov, M.I., Merzljakov, J.I.: Fundamentals of the Theory of Groups. Springer, Heidelberg (1979)
9. Kharlampovich, O., Khoussainov, B., Miasnikov, A.: From automatic structures to automatic groups. *Groups, Geom. Dyn.* **8**(1), 157–198 (2014)
10. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60178-3_93
11. Lyndon, R.C., Schupp, P.E.: Combinatorial Group Theory. Springer, Heidelberg (1977). <https://doi.org/10.1007/978-3-642-61896-3>
12. Roe, J.: Lectures on Coarse Geometry. University Lecture Series, vol. 31. American Mathematical Society, Providence (2003)
13. Vershik, A.: Numerical characteristics of groups and corresponding relations. *J. Math. Sci.* **107**(5), 4147–4156 (2001)



Pomsets and Unfolding of Reset Petri Nets

Thomas Chatain^{1(✉)}, Maurice Comlan^{2(✉)}, David Delfieu^{3(✉)},
Loïg Jezequel^{3(✉)}, and Olivier H. Roux^{3(✉)}

¹ LSV – ENS Cachan, Cachan, France

Thomas.chatain@lsv.ens-cachan.fr

² LETIA, Université d'Abomey-Calavi, Cotonou, Bénin
comlan@hotmail.fr

³ Université de Nantes and École Centrale de Nantes,
LS2N UMR 6004, Nantes, France

{david.delfieu,loig.jezequel}@ls2n.fr, olivier-h.roux@ec-nantes.fr

Abstract. Reset Petri nets are a particular class of Petri nets where transition firings can remove all tokens from a place without checking if this place actually holds tokens or not. In this paper we look at partial order semantics of such nets. In particular, we propose a pomset bisimulation for comparing their concurrent behaviours. Building on this pomset bisimulation we then propose a generalization of the standard finite complete prefixes of unfolding to the class of safe reset Petri nets.

1 Introduction

Petri nets are a well suited formalism for specifying, modeling, and analyzing systems with conflicts, synchronization and concurrency. Many interesting properties of such systems (reachability, boundedness, liveness, deadlock, ...) are decidable for Petri nets. Over time, many extensions of Petri nets have been proposed in order to capture specific, possibly quite complex, behaviors in a more direct manner. These extensions offer more compact representations and/or increase expressive power. One can notice, in particular, a range of extensions adding new kinds of arcs to Petri nets: read arcs and inhibitor arcs [3, 11] (allowing to read variables values without modifying them), and reset arcs [1] (allowing to modify variables values independently of their previous value). Reset arcs increase the expressiveness of Petri nets, but they compromise analysis techniques. For example, boundedness [6] and reachability [1] are undecidable. For bounded reset Petri nets, more properties are decidable, as full state spaces can be computed.

Full state-space computations (i.e. using state graphs) do not preserve partial order semantics. To face this problem, Petri nets unfolding has been proposed and has gained the interest of researchers in verification [7], diagnosis [4], and planning [9]. This technique keeps the intrinsic parallelism and prevents the combinatorial interleaving of independent events. While the unfolding of a Petri

net can be infinite, there exist algorithms for constructing finite prefixes of it [8, 10]. Unfolding have the strong interest of preserving more behavioral properties of Petri nets than state graphs. In particular they preserve concurrency and its counterpart: causality. Unfolding techniques have also been developed for extensions of Petri nets, and in particular Petri nets with read arcs [2].

Our Contribution: Reachability analysis is known to be feasible on bounded reset Petri nets, however, as far as we know, no technique for computing finite prefixes of unfolding exists yet, and so, no technique preserving concurrency and causality exists yet. This is the aim of this paper to propose one. For that, we characterise the concurrent behaviour of reset Petri nets by defining a notion of pomset bisimulation. This has been inspired by several works on pomset behaviour of concurrent systems [5, 12, 14]. From this characterization we can then express what should be an unfolding preserving the concurrent behaviour of a reset Petri net. We show that it is not possible to remove reset arcs from safe reset Petri nets while preserving their behaviours with respect to this pomset bisimulation. Then we propose a notion of finite complete prefixes of unfolding of safe reset Petri nets that allows for reachability analysis while preserving pomset behaviour. As a consequence of the two other contributions, these finite complete prefixes do have reset arcs.

This paper is organized as follows: We first give basic definitions and notations for (safe) reset Petri nets. Then, in Sect. 3, we propose the definition of a pomset bisimulation for reset Petri nets. In Sect. 4 we show that, in general, there is no Petri net without resets which is pomset bisimilar to a given reset Petri net. Finally, in Sect. 5 – building on the results of Sect. 4 – we propose a finite complete prefix construction for reset Petri nets.

2 Reset Petri Nets

Definition 1 (structure). A reset Petri net structure is a tuple (P, T, F, R) where P and T are disjoint sets of places and transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, and $R \subseteq P \times T$ is a set of reset arcs.

An element $x \in P \cup T$ is called a *node* and has a *preset* $\bullet x = \{y \in P \cup T : (y, x) \in F\}$ and a *postset* $x^\bullet = \{y \in P \cup T : (x, y) \in F\}$. If, moreover, x is a transition, it has a set of resets ${}^\diamond x = \{y \in P : (y, x) \in R\}$.

For two nodes $x, y \in P \cup T$, we say that: x is a *causal predecessor* of y , noted $x \prec y$, if there exists a sequence of nodes $x_1 \dots x_n$ with $n \geq 2$ so that $\forall i \in [1..n-1], (x_i, x_{i+1}) \in F$, $x_1 = x$, and $x_n = y$. If $x \prec y$ or $y \prec x$ we say that x and y are *in causal relation*. The nodes x and y are *in conflict*, noted $x \# y$, if there exists two sequences of nodes $x_1 \dots x_n$ with $n \geq 2$ and $\forall i \in [1..n-1], (x_i, x_{i+1}) \in F$, and $y_1 \dots y_m$ with $m \geq 2$ and $\forall i \in [1..m-1], (y_i, y_{i+1}) \in F$, so that $x_1 = y_1$ is a place, $x_2 \neq y_2$, $x_n = x$, and $y_m = y$.

A *marking* is a set $M \subseteq P$ of places. It *enables* a transition $t \in T$ if $\forall p \in \bullet t, p \in M$. In this case, t can be *fired* from M , leading to the new marking $M' = (M \setminus (\bullet t \cup {}^\diamond t)) \cup t^\bullet$. The fact that M enables t and that firing t leads to M' is denoted by $M[t]M'$.

Definition 2 (reset Petri net). A reset Petri net is a tuple (P, T, F, R, M_0) where (P, T, F, R) is a reset Petri net structure and M_0 is a marking called the initial marking.

Figure 1 (left) is a graphical representation of a reset Petri net. It has five places (circles) and three transitions (squares). Its set of arcs contains seven elements (arrows) and there is one reset arc (line with a diamond).

A marking M is said to be *reachable* in a reset Petri net if there exists a sequence $M_1 \dots M_n$ of markings so that: $\forall i \in [1..n-1], \exists t \in T, M_i[t] M_{i+1}$ (each marking enables a transition that leads to the next marking in the sequence), $M_1 = M_0$ (the sequence starts from the initial marking), and $M_n = M$ (the sequence leads to M). The set of all markings reachable in a reset Petri net \mathcal{N}_R is denoted by $[\mathcal{N}_R]$.

A reset Petri net with an empty set of reset arcs is simply called a *Petri net*.

Definition 3 (underlying Petri net). Given $\mathcal{N}_R = (P, T, F, R, M_0)$ a reset Petri net, we call its underlying Petri net the Petri net $\mathcal{N} = (P, T, F, \emptyset, M_0)$.

The above formalism is in fact a simplified version of the general formalism of reset Petri nets: arcs have no multiplicity and markings are sets of places rather than multisets of places. We use it because it suffices for representing *safe* nets.

Definition 4 (safe reset Petri net). A reset Petri net (P, T, F, R, M_0) is said to be safe if for any reachable marking M and any transition $t \in T$, if M enables t then $(t^\bullet \setminus (\bullet t \cup {}^t)) \cap M = \emptyset$.

The reader familiar with Petri nets will notice that our results generalize to larger classes of nets: unbounded reset Petri nets for our pomset bisimulation (Sect. 3), and bounded reset Petri nets for our prefix construction (Sect. 5).

In the rest of the paper, unless the converse is specified, we consider reset Petri nets so that the preset of each transition t is non-empty: $\bullet t \neq \emptyset$. Notice that this is not a restriction to our model: one can equip any transition t of a reset Petri net with a place p_t so that p_t is in the initial marking and $\bullet p_t = p_t^\bullet = \{t\}$.

One may need to express that two (reset) Petri nets have the same behaviour. This is useful in particular for building minimal (or at least small, that is with few places and transitions) representatives of a net; or for building simple (such as loop-free) representatives of a net. A standard way to do so is to define a bisimulation between (reset) Petri nets, and state that two nets have the same behaviour if they are bisimilar.

The behaviour of a net will be an observation of its transition firing, this observation being defined thanks to a labelling of nets associating to each transition an observable label or the special unobservable label ε .

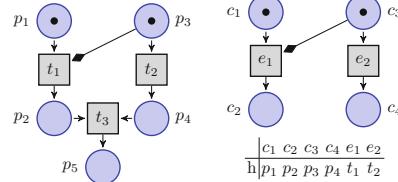


Fig. 1. A reset Petri net (left) and one of its processes (right)

Definition 5 (labelled reset Petri net). A labelled reset Petri net is a tuple $(\mathcal{N}_R, \Sigma, \lambda)$ so that: $\mathcal{N}_R = (P, T, F, R, M_0)$ is a reset Petri net, Σ is a set of transition labels, and $\lambda : T \rightarrow \Sigma \cup \{\varepsilon\}$ is a labelling function.

In such a labelled net we extend the labelling function λ to sequences of transitions in the following way: given a sequence $t_1 \dots t_n$ (with $n \geq 2$) of transitions, $\lambda(t_1 \dots t_n) = \lambda(t_1)\lambda(t_2 \dots t_n)$ if $\lambda(t_1) \in \Sigma$ and $\lambda(t_1 \dots t_n) = \lambda(t_2 \dots t_n)$ else (that is if $\lambda(t_1) = \varepsilon$). From that, one can define bisimulation as follows.

Definition 6 (bisimulation). Let $(\mathcal{N}_{R,1}, \Sigma_1, \lambda_1)$ and $(\mathcal{N}_{R,2}, \Sigma_2, \lambda_2)$ be two labelled reset Petri nets with $\mathcal{N}_{R,i} = (P_i, T_i, F_i, R_i, M_{0,i})$. They are bisimilar if and only if there exists a relation $\rho \subseteq [\mathcal{N}_{R,1}] \times [\mathcal{N}_{R,2}]$ (a bisimulation) so that:

1. $(M_{0,1}, M_{0,2}) \in \rho$,
2. if $(M_1, M_2) \in \rho$, then
 - (a) for every transition $t \in T_1$ so that $M_1[t]M_{1,n}$ there exists a sequence $t_1 \dots t_n$ of transitions from T_2 and a sequence $M_{2,1} \dots M_{2,n}$ of markings of $\mathcal{N}_{R,2}$ so that: $M_2[t_1]M_{2,1}[t_2] \dots [t_n]M_{2,n}$, $\lambda_2(t_1 \dots t_n) = \lambda_1(t)$, and $(M_{1,n}, M_{2,n}) \in \rho$
 - (b) the other way around (for every transition $t \in T_2 \dots$)

This bisimulation however hides an important part of the behaviours of (reset) Petri nets: transition firings may be *concurrent* when transitions are not in causal relation nor in conflict. For example, consider Fig. 2 where $\mathcal{N}_{R,1}$ and $\mathcal{N}_{R,2}$ are bisimilar (we identify transition names and labels). In $\mathcal{N}_{R,1}$, t_1 and t_2 are not in causal relation while in $\mathcal{N}_{R,2}$ they are in causal relation.

To avoid this loss of information, a standard approach is to define bisimulations based on partially ordered sets of transitions rather than totally ordered sets of transitions (the transition sequences used in the above definition). Such bisimulations are usually called pomset bisimulations.

3 Pomset Bisimulation for Reset Petri Nets

In this section, we propose a definition of pomset bisimulation for reset Petri nets. It is based on an ad hoc notion of processes (representations of the executions of a Petri net, concurrent counterpart of paths in automata).

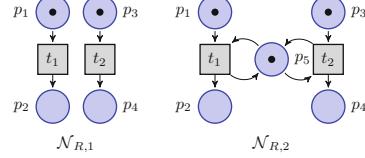


Fig. 2. Two bisimilar nets

3.1 Processes of Reset Petri Nets

We recall a standard notion of processes of Petri nets and show how it can be extended to reset Petri nets. As a first step, we define *occurrence nets* which are basically Petri nets without loops.

Definition 7 (occurrence net). An occurrence net is a (reset) Petri net $(B, E, F^{\mathcal{O}}, R^{\mathcal{O}}, M_0^{\mathcal{O}})$ so that, $\forall b \in B, \forall x \in B \cup E$: (1) $|\bullet b| \leq 1$, (2) x is not in causal relation with itself, (3) x is not in conflict with itself, (4) $\{y \in B \cup E : y \prec x\}$ is finite, (5) $b \in M_0^{\mathcal{O}}$ if and only if $\bullet b = \emptyset$.

Places of an occurrence net are usually referred to as *conditions* and transitions as *events*. In an occurrence net, if two nodes $x, y \in B \cup E$ are so that $x \neq y$, are not in causal relation, and are not in conflict, they are said to be *concurrent*. Moreover, in occurrence net, the causal relation is a partial order.

There is a price to pay for having reset arcs in occurrence nets. With no reset arcs, checking if a set E of events together form a feasible execution (i.e. checking that the events from E can all be ordered so that they can be fired in this order starting from the initial marking) is linear in the size of the occurrence net (it suffices to check that E is causally closed and conflict free). With reset arcs the same task is NP-complete as stated in the below proposition.

Proposition 1. The problem of deciding if a set E of events of an occurrence net with resets forms a feasible execution is NP-complete.

Proof. (Sketch) Graph 3-coloring reduces to executability of an occurrence net.

The branching processes of a Petri net are then defined as particular occurrence nets linked to the original net by *homomorphisms*.

Definition 8 (homomorphism of nets). Let \mathcal{N}_1 and \mathcal{N}_2 be two Petri nets such that $\mathcal{N}_i = (P_i, T_i, F_i, \emptyset, M_{0,i})$. A mapping $h : P_1 \cup T_1 \rightarrow P_2 \cup T_2$ is an homomorphism of nets from \mathcal{N}_1 to \mathcal{N}_2 if $\forall p_1 \in P_1, \forall p_2 \in P_2, \forall t \in T_1$: (1) $h(p_1) \in P_2$, (2) $h(t) \in T_2$, (3) $p_2 \in \bullet h(t) \Leftrightarrow \exists p'_1 \in \bullet t, h(p'_1) = p_2$, (4) $p_2 \in h(t)^{\bullet} \Leftrightarrow \exists p'_1 \in t^{\bullet}, h(p'_1) = p_2$, (5) $p_2 \in M_{0,2} \Leftrightarrow \exists p'_1 \in M_{0,1}, h(p'_1) = p_2$.

Definition 9 (processes of a Petri net). Let $\mathcal{N} = (P, T, F, \emptyset, M_0)$ be a Petri net, $\mathcal{O} = (B, E, F^{\mathcal{O}}, \emptyset, M_0^{\mathcal{O}})$ be an occurrence net, and h be an homomorphism of nets from \mathcal{O} to \mathcal{N} . Then (\mathcal{O}, h) is a branching process of \mathcal{N} if $\forall e_1, e_2 \in E, (\bullet e_1 = \bullet e_2 \wedge h(e_1) = h(e_2)) \Rightarrow e_1 = e_2$. If, moreover, $\forall b \in B, |\bullet b| \leq 1$, then (\mathcal{O}, h) is a process of \mathcal{N} .

Finally, a process of a reset Petri net is obtained by adding reset arcs to a process of the underlying Petri net (leading to what we call below a potential process) and checking that all its events can still be enabled and fired in some order.

Definition 10 (potential processes of a reset Petri net). Let $\mathcal{N}_R = (P, T, F, R, M_0)$ be a reset Petri net and \mathcal{N} be its underlying Petri net, let $\mathcal{O} = (B, E, F^\mathcal{O}, R^\mathcal{O}, M_0^\mathcal{O})$ be an occurrence net, and h be an homomorphism of nets from \mathcal{O} to \mathcal{N}_R . Then (\mathcal{O}, h) is a potential process of \mathcal{N}_R if (1) (\mathcal{O}', h) is a process of \mathcal{N} with $\mathcal{O}' = (B, E, F^\mathcal{O}, \emptyset, M_0^\mathcal{O})$, (2) $\forall b \in B, \forall e \in E, (b, e) \in R^\mathcal{O}$ if and only if $(h(b), h(e)) \in R$.

Definition 11 (processes of a reset Petri net). Let $\mathcal{N}_R = (P, T, F, R, M_0)$ be a reset Petri net, $\mathcal{O} = (B, E, F^\mathcal{O}, R^\mathcal{O}, M_0^\mathcal{O})$ be an occurrence net, and h be an homomorphism of nets from \mathcal{O} to \mathcal{N}_R . Then (\mathcal{O}, h) is a process of \mathcal{N}_R if (1) (\mathcal{O}, h) is a potential process of \mathcal{N}_R , and (2) if $E = \{e_1, \dots, e_n\}$ then $\exists M_1, \dots, M_n \subseteq B$ so that $M_0^\mathcal{O}[e_{k_1}]M_1[e_{k_2}] \dots [e_{k_n}]M_n$ with $\{k_1, \dots, k_n\} = \{1, \dots, n\}$.

Notice that processes of reset Petri nets and processes of Petri nets do not exactly have the same properties. In particular, two properties are central in defining pomset bisimulation for Petri nets and do not hold for reset Petri nets.

Property 1. In any process of a Petri net with set of events E , consider any sequence of events $e_1e_2\dots e_n$ (1) that contains all the events in E and (2) such that $\forall i, j \in [1..n]$ if $e_i \prec e_j$ then $i < j$. Necessarily, there exist markings M_1, \dots, M_n so that $M_0^\mathcal{O}[e_1]M_1[e_2] \dots [e_n]M_n$.

This property (which, intuitively, expresses that processes are partially ordered paths) is no longer true for reset Petri nets. Consider for example the reset Petri net of Fig. 1 (left). Figure 1 (right) is one of its processes (the occurrence net with the homomorphism h below). As not $e_2 \prec e_1$, their should exist markings M_1, M_2 so that $M_0[e_1]M_1[e_2]M_2$. However, $M_0 = \{c_1, c_3\}$ indeed enables e_1 , but the marking M_1 such that $M_0[e_1]M_1$ is $\{c_2\}$, which does not enable e_2 .

Property 2. In a process of a Petri net all the sequences of events $e_1e_2\dots e_n$ verifying (1) and (2) of Property 1 lead to the same marking (i.e. M_n is always the same), thus uniquely defining a notion of maximal marking of a process.

This property defines the marking reached by a process. As a corollary of Property 1 not holding for reset Petri nets, there is no uniquely defined notion of maximal marking in their processes. Back to the example $\{c_2\}$ is somehow maximal (no event can be fired from it) as well as $\{c_2, c_4\}$.

To transpose the spirit of Properties 1 and 2 to processes of reset Petri nets, we define below a notion of maximal markings in such processes.

Definition 12 (maximal markings). Let $\mathcal{P} = (\mathcal{O}, h)$ be a process with set of events $E = \{e_1, \dots, e_n\}$ and initial marking $M_0^\mathcal{O}$ of a reset Petri net. The set $M_{\max}(\mathcal{P})$ of maximal markings of \mathcal{P} contains exactly the markings M so that $\exists M_1, \dots, M_{n-1}$, verifying $M_0^\mathcal{O}[e_{k_1}]M_1[e_{k_2}] \dots M_{n-1}[e_{k_n}]M$ for some $\{k_1, \dots, k_n\} = \{1, \dots, n\}$.

In other words, the maximal markings of a process are all the marking that are reachable in it using all its events. This, in particular, excludes $\{c_2\}$ in the above example.

3.2 Abstracting Processes

We show how processes of labelled reset Petri nets can be abstracted as partially ordered multisets (pomsets) of labels.

Definition 13 (pomset abstraction of processes). Let $(\mathcal{N}_R, \Sigma, \lambda)$ be a labelled reset Petri net and (\mathcal{O}, h) be a process of \mathcal{N}_R with $\mathcal{O} = (B, E, F^\mathcal{O}, R^\mathcal{O}, M_0^\mathcal{O})$. Define $E' = \{e \in E : \lambda(h(e)) \neq \varepsilon\}$. Define $\lambda' : E' \rightarrow \Sigma$ as the function so that $\forall e \in E', \lambda'(e) = \lambda(h(e))$. Define moreover $< \subseteq E' \times E'$ as the relation so that $e_1 < e_2$ if and only if $e_1 \prec e_2$ (e_1 is a causal predecessor of e_2 in \mathcal{O}). Then, $(E', <, \lambda')$ is the pomset abstraction of (\mathcal{O}, h) .

This abstraction $(E, <, \lambda')$ of a process is called its pomset abstraction because it can be seen as a multiset of labels (several events may have the same associated label by λ') that are partially ordered by the $<$ relation. In order to compare processes with respect to their pomset abstractions, we also define the following equivalence relation.

Definition 14 (pomset equivalence). Let $(E, <, \lambda)$ and $(E', < ', \lambda')$ be the pomset abstractions of two processes \mathcal{P} and \mathcal{P}' . These processes are pomset equivalent, noted $\mathcal{P} \equiv \mathcal{P}'$ if and only if there exists a bijection $f : E \rightarrow E'$ so that $\forall e_1, e_2 \in E$: (1) $\lambda(e_1) = \lambda'(f(e_1))$, and (2) $e_1 < e_2$ if and only if $f(e_1) < ' f(e_2)$.

Intuitively, two processes are pomset equivalent if their pomset abstractions define the same pomset: same multisets of labels with same partial orderings. Finally, we also need to be able to abstract processes as sequences of labels.

Definition 15 (linear abstraction). Let $(\mathcal{N}_R, \Sigma, \lambda)$ be a labelled reset Petri net, let $\mathcal{P} = (\mathcal{O}, h)$ be a process of \mathcal{N}_R with $\mathcal{O} = (B, E, F^\mathcal{O}, R^\mathcal{O}, M_0^\mathcal{O})$, and let M be a reachable marking in \mathcal{O} . Define $\lambda' : E \rightarrow \Sigma$ as the function so that $\forall e \in E, \lambda'(e) = \lambda(h(e))$. The linear abstraction of \mathcal{P} with respect to M is the set $\text{lin}(M, \mathcal{P})$ so that a sequence of labels ω is in $\text{lin}(M, \mathcal{P})$ if and only if in \mathcal{O} there exist markings M_1, \dots, M_{n-1} and events e_1, \dots, e_n so that $M_0^\mathcal{O}[e_1] M_1[e_2] \dots M_{n-1}[e_n] M$ and $\lambda'(e_1 \dots e_n) = \omega$.

3.3 Pomset Bisimulation

We now define a notion of pomset bisimulation between reset Petri nets, inspired by [5, 12, 14]. Intuitively, two reset Petri nets are pomset bisimilar if there exists a relation between their reachable markings so that the markings that can be reached by pomset equivalent processes from two markings in relation are themselves in relation. This is formalized by the below definition.

Definition 16 (pomset bisimulation for reset nets). Let $(\mathcal{N}_{R,1}, \Sigma_1, \lambda_1)$ and $(\mathcal{N}_{R,2}, \Sigma_2, \lambda_2)$ be two labelled reset Petri nets with $\mathcal{N}_{R,i} = (P_i, T_i, F_i, R_i, M_{0,i})$. They are pomset bisimilar if and only if there exists a relation $\rho \subseteq [\mathcal{N}_{R,1}] \times [\mathcal{N}_{R,2}]$ (called a pomset bisimulation) so that:

1. $(M_{0,1}, M_{0,2}) \in \rho$,
2. if $(M_1, M_2) \in \rho$, then
 - (a) for every process \mathcal{P}_1 of $(P_1, T_1, F_1, R_1, M_1)$ there exists a process \mathcal{P}_2 of $(P_2, T_2, F_2, R_2, M_2)$ so that $\mathcal{P}_1 \equiv \mathcal{P}_2$ and
 - $\forall M'_1 \in M_{max}(\mathcal{P}_1), \exists M'_2 \in M_{max}(\mathcal{P}_2)$ so that $(M'_1, M'_2) \in \rho$,
 - $\forall M'_1 \in M_{max}(\mathcal{P}_1), \forall M'_2 \in M_{max}(\mathcal{P}_2), (M'_1, M'_2) \in \rho \Rightarrow lin(M'_1, \mathcal{P}_1) = lin(M'_2, \mathcal{P}_2)$.
 - (b) the other way around (for every process $\mathcal{P}_2 \dots$)

Notice that, in the above definition, taking the processes \mathcal{P}_1 and \mathcal{P}_2 bisimilar (using the standard bisimulation relation for Petri nets) rather than comparing $lin(M'_1, \mathcal{P}_1)$ and $lin(M'_2, \mathcal{P}_2)$ would lead to an equivalent definition.

Remark that pomset bisimulation implies bisimulation, as expressed by the following proposition. The converse is obviously not true.

Proposition 2. Let $(\mathcal{N}_{R,1}, \Sigma_1, \lambda_1)$ and $(\mathcal{N}_{R,2}, \Sigma_2, \lambda_2)$ be two pomset bisimilar labelled reset Petri nets, then $(\mathcal{N}_{R,1}, \Sigma_1, \lambda_1)$ and $(\mathcal{N}_{R,2}, \Sigma_2, \lambda_2)$ are bisimilar.

Proof. It suffices to notice that Definition 6 can be obtained from Definition 16 by restricting the processes considered, taking only those with exactly one transition whose label is different from ε .

4 Reset Arcs Removal and Pomset Bisimulation

From now on, we consider that (reset) Petri nets are finite, i.e. their sets of places and transitions are finite.

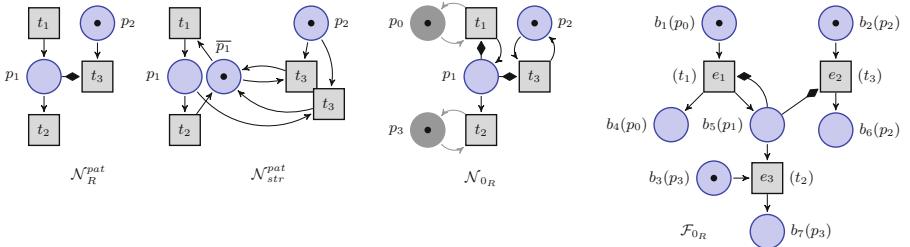


Fig. 3. A remarkable pattern \mathcal{N}_R^{pat} and its structural transformation \mathcal{N}_{str}^{pat} , a labelled reset Petri net \mathcal{N}_{0R} including the pattern \mathcal{N}_R , and a finite complete prefix \mathcal{F}_{0R} of \mathcal{N}_{0R} . Transition labels are given on transitions.

In this section, we prove that it is, in general, not possible to remove reset arcs from safe reset Petri nets while preserving their behaviours with respect to this pomset bisimulation. More precisely, we prove that it is not possible to build a safe labelled Petri net (while this is out of the scope of this paper, the reader familiar with Petri nets may notice that this is the case for bounded labelled

Petri net) without reset arcs which is pomset bisimilar to a given safe labelled reset Petri net. For that, we exhibit a particular pattern – Fig. 3 (left) – and show that a reset Petri net including this pattern cannot be pomset bisimilar to a Petri net without reset arcs.

As a first intuition of this fact, let us consider the following structural transformation that removes reset arcs from a reset Petri net.

Definition 17 (Structural transformation). Let $(\mathcal{N}_R, \Sigma, \lambda)$ be a labelled reset Petri net such that $\mathcal{N}_R = (P, T, F, R, M_0)$, its structural transformation is the labelled Petri net $(\mathcal{N}_{R,str}, \Sigma_{str}, \lambda_{str})$ where $\mathcal{N}_{R,str} = (P_{str}, T_{str}, F_{str}, \emptyset, M_{0,str})$ so that:

$$P_{str} = P \cup \overline{P} \text{ with } \overline{P} = \{\bar{p} : p \in P \wedge \exists t \in T, (p, t) \in R\},$$

$$T_{str} = T \cup \overline{T} \text{ with } \overline{T} = \{\bar{t} : t \in T \wedge {}^{\diamond}t \neq \emptyset\},$$

$$F_{str} = F \cup \{(p, \bar{t}) : \bar{t} \in \overline{T}, (p, t) \in F\} \cup \{(\bar{t}, p) : \bar{t} \in \overline{T}, (t, p) \in F\} \quad (1)$$

$$\cup \{(\bar{p}, t) : \bar{p} \in \overline{P}, (t, p) \in F\} \cup \{(t, \bar{p}) : \bar{p} \in \overline{P}, (p, t) \in F\} \quad (2)$$

$$\cup \{(\bar{p}, \bar{t}) \in \overline{P} \times \overline{T} : (t, p) \in F\} \cup \{(\bar{t}, \bar{p}) \in \overline{T} \times \overline{P} : (p, t) \in F\} \quad (3)$$

$$\cup \{(p, t), (\bar{p}, \bar{t}), (t, \bar{p}), (\bar{t}, \bar{p}) : (p, t) \in R\}, \quad (4)$$

$$M_{0,str} = M_0 \cup \{\bar{p} \in \overline{P} : p \notin M_0\},$$

and moreover, $\Sigma_{str} = \Sigma$, $\forall t \in T, \lambda_{str}(t) = \lambda(t)$, and $\forall \bar{t} \in \overline{T}, \lambda_{str}(\bar{t}) = \lambda(t)$.

Intuitively, in this transformation, for each reset arc (p, t) , a copy \bar{p} of p and a copy \bar{t} of t are created. The two places are so that p is marked if and only if \bar{p} is not marked, the transition t will perform the reset when p is marked and \bar{t} will perform it when p is not marked (i.e. when \bar{p} is marked). For that, new arcs are added to F so that: \bar{t} mimics t (1), the link between p and \bar{p} is enforced (2, 3), and the resets are either performed by t or \bar{t} depending of the markings of p and \bar{p} (4). This is exemplified in Fig. 3 (left and middle left).

Lemma 1. A labelled reset Petri net $(\mathcal{N}_R, \Sigma, \lambda)$ and its structural transformation $(\mathcal{N}_{R,str}, \Sigma_{str}, \lambda_{str})$ as defined in Definition 17 are bisimilar.

Proof. (Sketch) The bisimulation relation is $\rho \subseteq [\mathcal{N}_{R,1}] \times [\mathcal{N}_{R,2}]$ defined by $(M, M_{struct}) \in \rho$ iff $\forall p \in P, M(p) = M_{struct}(p)$ and $\forall p \in P$ such that $\bar{p} \in \overline{P}$, we have $M_{struct}(p) + M_{struct}(\bar{p}) = 1$.

For the transformation of Definition 17, a reset Petri net and its transformation are bisimilar but not always pomset bisimilar. This can be remarked on any safe reset Petri net including the pattern \mathcal{N}_R^{pat} of Fig. 3. Indeed, this transformation adds in \mathcal{N}_{str}^{pat} a causality relation between the transition labelled by t_1 and each of the two transitions labelled by t_3 . From the initial marking of \mathcal{N}_{str}^{pat} , for any process whose pomset abstraction includes both t_1 and t_3 , these two labels are causally ordered. While, from the initial marking of \mathcal{N}_R^{pat} there is a process which pomset abstraction includes both t_1 and t_3 but does not order them. We now generalize this result.

Let us consider the labelled reset Petri Net \mathcal{N}_{0_R} of Fig. 3 (middle right). It uses the pattern \mathcal{N}_R^{pat} of Fig. 3 in which t_1 and t_3 can be fired in different order infinitely often. In this net, the transitions with labels t_1 and t_3 are not in causal relation.

Proposition 3. *There is no finite safe labelled Petri net (i.e. without reset arc) which is pomset bisimilar to the labelled reset Petri net \mathcal{N}_{0_R} .*

Proof. We simply remark that any finite safe labelled Petri net with no reset arcs which is bisimilar to \mathcal{N}_{0_R} has a causal relation between two transitions labelled by t_1 and t_3 respectively (Lemma 2). From that, by Proposition 2, we get that any such labelled Petri net \mathcal{N} which would be pomset bisimilar to \mathcal{N}_{0_R} would have a process from its initial marking whose pomset abstraction is such that some occurrence of t_1 and some occurrence of t_3 are ordered, while this is never the case in the processes of \mathcal{N}_{0_R} . This prevents \mathcal{N} from being pomset bisimilar to \mathcal{N}_{0_R} , and thus leads to a contradiction, proving the proposition.

Lemma 2. *Any safe labelled Petri net with no reset arcs which is bisimilar (see definition 6) to \mathcal{N}_{0_R} has a causal relation between two transitions labelled by t_1 and t_3 respectively.*

Proof. (Sketch) The firing of t_3 prevents the firing of t_2 ; then t_3 and t_2 are in conflict and share an input place which has to be marked again after the firing of t_1 . This place generates a causality between t_1 and t_3 .

5 Finite Complete Prefixes of Unfolding of Reset Petri Nets

In this section, we propose a notion of finite complete prefixes of unfolding of safe reset Petri nets preserving reachability of markings and pomset behaviour. As a consequence of the previous section, these finite complete prefixes do have reset arcs.

The unfolding of a Petri net is a particular branching process (generally infinite) representing all its reachable markings and ways to reach them. It also preserves concurrency.

Definition 18 (Unfolding of a Petri net). *The unfolding of a net can be defined as the union of all its branching processes [7] or equivalently its largest branching process (with respect to inclusion).*

In the context of reset Petri nets, no notion of unfolding has been defined yet. Accordingly to our notion of processes for reset Petri nets and because of Proposition 4 below we propose Definition 19. In it and the rest of the paper, nets and labelled nets are identified (each transition is labelled by itself) and labellings of branching processes are induced by homomorphisms (as for pomset abstraction).

Definition 19 (Unfolding of a reset Petri net). Let \mathcal{N}_R be a safe reset Petri net and \mathcal{N} be its underlying Petri net. Let \mathcal{U} be the unfolding of \mathcal{N} . The unfolding of \mathcal{N}_R is \mathcal{U}_R , obtained by adding reset arcs to \mathcal{U} according to (2) in Definition 10.

Proposition 4. Any safe (labelled) reset Petri net \mathcal{N}_R and its unfolding \mathcal{U}_R are pomset bisimilar.

Proof. (Sketch) This extends a result of [13], stating that two Petri nets having the same unfolding (up to isomorphism) are pomset bisimilar (for a notion of bisimulation coping with our in absence of resets).

Petri nets unfolding is however unpractical for studying Petri nets behaviour as it is generally an infinite object. In practice, finite complete prefixes of it are preferred [8, 10].

Definition 20 (finite complete prefix, reachable marking preservation). A finite complete prefix of the unfolding of a safe Petri net \mathcal{N} is a finite branching processes (\mathcal{O}, h) of \mathcal{N} verifying the following property of reachable marking preservation: a marking M is reachable in \mathcal{N} if and only if there exists a reachable marking M' in \mathcal{O} so that $M = \{h(b) : b \in M'\}$.

In this section, we propose an algorithm for construction of finite complete prefixes for safe reset Petri nets. For that, we assume the existence of a black-box algorithm for building finite complete prefixes of safe Petri nets (without reset arcs). Notice that such algorithms indeed do exist [8, 10].

Because of Proposition 3, we know that such finite prefixes should have reset arcs to preserve pomset behaviour. We first remark that directly adding reset arcs to finite complete prefixes of underlying nets would not work.

Proposition 5. Let \mathcal{U} be the unfolding of the underlying Petri Net \mathcal{N} of a safe reset Petri net \mathcal{N}_R , let \mathcal{F} be one of its finite and complete prefixes. Let \mathcal{F}' be the object obtained by adding reset arcs to \mathcal{F} according to (2) in Definition 10. The reachable marking preservation is in general not verified by \mathcal{F}' (with respect to \mathcal{N}_R).

The proof of this proposition relies on the fact that some reachable markings of \mathcal{N}_R are not represented in \mathcal{F}' . This suggests that this prefix is not big enough. We however know an object that contains, for sure, every reachable marking of \mathcal{N}_R along with a way to reach each of them: its structural transformation $\mathcal{N}_{R,str}$ (Definition 17). We thus propose to compute finite prefixes of reset Petri nets from their structural transformations: in the below algorithm, \mathcal{F}_{str} is used to determine the deepness of the prefix (i.e. the length of the longest chain of causally ordered transitions).

Algorithm 1 (Finite complete prefix construction for reset Petri nets). Let \mathcal{N}_R be a safe reset Petri net, (step 1) compute the structural transformation $\mathcal{N}_{R,str}$ of \mathcal{N}_R , (step 2) compute a finite complete prefix \mathcal{F}_{str} of $\mathcal{N}_{R,str}$, (step 3)

compute a finite prefix \mathcal{F} of \mathcal{U} (the unfolding of the underlying net \mathcal{N}) that simulates \mathcal{F}_{str} (a labelled net \mathcal{N}_2 simulates a labelled net \mathcal{N}_1 if they verify Definition 6 except for condition 2.b.), (step 4) compute \mathcal{F}_R by adding reset arcs from \mathcal{N}_R to \mathcal{F} according to (2) in Definition 10. The output of the algorithm is \mathcal{F}_R .

Applying this algorithm to the net \mathcal{N}_{0_R} of Fig. 3 (middle right) – using the algorithm from [8] at step 2 – leads to the reset Petri net \mathcal{F}_{0_R} of Fig. 3 (right).

Notice that the computation of \mathcal{F}_{str} – step 1 and 2 – can be done in exponential time and space with respect to the size of \mathcal{N}_R . The computation of \mathcal{F} from \mathcal{F}_{str} (step 3) is linear in the size of \mathcal{F} . And, the addition of reset arcs (step 4) is at most quadratic in the size of \mathcal{F} .

We conclude this section by showing that Algorithm 1 actually builds finite complete prefixes of reset Petri nets.

Proposition 6. *The object \mathcal{F}_R obtained by Algorithm 1 from a safe reset Petri net \mathcal{N}_R is a finite and complete prefix of the unfolding of \mathcal{N}_R .*

Proof. Notice that if \mathcal{N}_R is safe, then $\mathcal{N}_{R,str}$ is safe as well. Thus \mathcal{F}_{str} is finite by definition of finite complete prefixes of Petri nets (without reset arcs). \mathcal{F}_{str} is finite and has no node in causal relation with itself (i.e. no cycle), hence any net bisimilar with it is also finite, this is in particular the case of \mathcal{F} . Adding reset arcs to a finite object does not break its finiteness, so \mathcal{F}_R is finite.

Moreover, \mathcal{F}_{str} is complete by definition of finite complete prefixes of Petri nets (without reset arcs). As \mathcal{F} simulates \mathcal{F}_{str} it must also be complete (it can only do more). The reset arcs addition removes semantically to \mathcal{F} only the unexpected sequences (i.e. the sequence which are possible in \mathcal{F} but not in \mathcal{F}_{str}). Therefore, \mathcal{F}_R is complete.

6 Conclusion

Our contribution in this paper is three-fold. First, we proposed a notion of pomset bisimulation for reset Petri nets. This notion is, in particular, inspired from a similar notion that has been defined for Petri nets (without reset arcs) in [5]. Second, we have shown that it is not possible to remove reset arcs from safe reset Petri nets while preserving their behaviours with respect to this pomset bisimulation. And, third, we proposed a notion of finite complete prefixes of unfolding of safe reset Petri nets that allows for reachability analysis while preserving pomset behaviour. As a consequence of the two other contributions, these finite complete prefixes do have reset arcs.

References

1. Araki, T., Kasami, T.: Some decision problems related to the reachability problem for Petri nets. *Theor. Comput. Sci.* **3**(1), 85–104 (1976)
2. Baldan, P., Bruni, A., Corradini, A., König, B., Rodríguez, C., Schwoon, S.: Efficient unfolding of contextual Petri nets. *Theor. Comput. Sci.* **449**, 2–22 (2012)
3. Baldan, P., Corradini, A., Montanari, U.: Contextual Petri nets, asymmetric event structures and processes. *Inf. Comput.* **171**(1), 1–49 (2001)
4. Benveniste, A., Fabre, E., Haar, S., Jard, C.: Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *IEEE TAC* **48**(5), 714–727 (2003)
5. Best, E., Devillers, R.R., Kiehn, A., Pomello, L.: Concurrent bisimulations in Petri nets. *Acta Inf.* **28**(3), 231–264 (1991)
6. Dufourd, C., Schnoebelen, P., Jančar, P.: Boundedness of reset P/T nets. In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 301–310. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48523-6_27
7. Esparza, J., Heljanko, K.: Unfoldings - A Partial-Order Approach to Model Checking. Springer, Heidelberg (2008). <https://doi.org/10.1007/978-3-540-77426-6>
8. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan’s unfolding algorithm. *Form. Methods Syst. Des.* **20**(3), 285–310 (2002)
9. Hickmott, S., Rintanen, J., Thiébaut, S., White, L.: Planning via Petri net unfolding. In: IJCAI, pp. 1904–1911 (2007)
10. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: von Bochmann, G., Probst, D.K. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56496-9_14
11. Montanari, U., Rossi, F.: Contextual nets. *Acta Inf.* **32**(6), 545–596 (1995)
12. van Glabbeek, R., Goltz, U.: Equivalence notions for concurrent systems and refinement of actions. In: Kreczmar, A., Mirkowska, G. (eds.) MFCS 1989. LNCS, vol. 379, pp. 237–248. Springer, Heidelberg (1989). https://doi.org/10.1007/3-540-51486-4_71
13. van Glabbeek, R., Vaandrager, F.: Petri net models for algebraic theories of concurrency. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE 1987. LNCS, vol. 259, pp. 224–242. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-17945-3_13
14. Vogler, W.: Bisimulation and action refinement. *Theor. Comput. Sci.* **114**(1), 173–200 (1993)



Timed Comparisons of Semi-Markov Processes

Mathias R. Pedersen¹ , Nathanaël Fijalkow², Giorgio Bacci¹ ,
Kim G. Larsen¹, and Radu Mardare¹

¹ Department of Computer Science, Aalborg University,
Selma Lagerlöfsvej 300, Aalborg, Denmark
`{mfp,grbacci,kgl,mardare}@cs.aau.dk`

² The Alan Turing Institute, 96 Euston Road, London NW1 2DB, UK
`nfijalkow@turing.ac.uk`

Abstract. Semi-Markov processes are Markovian processes in which the firing time of transitions is modelled by probabilistic distributions over positive reals interpreted as the probability of firing a transition at a certain moment in time.

In this paper we consider the trace-based semantics of semi-Markov processes, and investigate the question of how to compare two semi-Markov processes with respect to their time-dependent behaviour. To this end, we introduce the relation of being “faster than” between processes and study its algorithmic complexity. Through a connection to probabilistic automata we obtain hardness results showing in particular that this relation is undecidable. However, we present an additive approximation algorithm for a time-bounded variant of the faster-than problem over semi-Markov processes with slow residence-time functions, and a **coNP** algorithm for the exact faster-than problem over unambiguous semi-Markov processes.

Keywords: Automata for system analysis and programme verification · Real-time systems · Semi-Markov processes · Probabilistic automata

1 Introduction

Semi-Markov processes are Markovian stochastic systems that model the firing time of transitions as probabilistic distribution over positive reals; thus, one can encode the probability of firing a certain transition within a certain time interval. For example, continuous-time Markov processes are particular case of semi-Markov processes where the timing distributions are always exponential.

Semi-Markov processes have been used extensively to model real-time systems such as power plants [15] and power supply units [16]. For such real-time systems, non-functional requirements are becoming increasingly important.

This work was supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1, the Danish FTP project ASAP, the ERC Advanced Grant LASSO, and the Sino-Danish Basic Research Center IDEA4CPS.

Many of these requirements, such as response time and throughput, depend heavily on the timing behaviour of the system in question. It is therefore natural to understand and be able to compare the timing behaviour of different systems.

Moller and Tofts [11] proposed the notion of a *faster-than* relation for systems with discrete-time in the context of process algebras. Their goal was to be able to compare processes that are functionally behaviourally equivalent, except that one process may execute actions faster than the other. This line of study was continued by Lüttgen and Vogler [10], who moreover considered upper bounds on time, in order to allow for reasoning about worst-case timing behaviours. For timed automata, Guha et al. [9] introduced a bisimulation-like faster-than relation and studied its compositional properties. For continuous-time probabilistic systems, Baier et al. [3] considered a simulation relation where the timing distribution on each state is required to stochastically dominate the other. They introduced both a weak and a strong version of their simulation relation, and gave a logical characterization of these in terms of the logic CSL.

In the literature, less attention has been drawn to trace-based notions of faster-than relations although trace equivalence and inclusion are important concepts when considering linear-time properties such as liveness or safety [2]. In this paper we propose a simple and intuitive notion of trace inclusion for semi-Markov processes, which we call *faster-than* relation, that compares the relative speed of processes with respect to the execution of arbitrary sequences of actions.

Differently from trace inclusion, our relation does not make a step-wise comparison of the timing delays for each individual action in a sequence, but over the overall execution time of the sequence. As an example, consider the semi-Markov process in Fig. 1. The states s and s' , although performing the same sequences of actions, are not related by trace inclusion because the first two actions in any sequence are individually executed at opposite order of speeds (here governed by exponential-time distributions). Instead, according to our relation, s is faster-than s' (but not vice versa) because it executes single-action sequences at a faster rate than s' , and action sequences of length greater than one at the same speed — this is due to the fact that the execution time of each action is governed by random variables that are independent of each other and the sum of independent random variables is commutative.

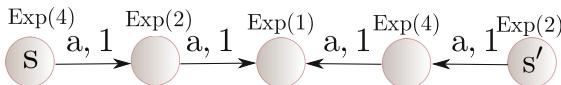


Fig. 1. A semi-Markov process where s is faster than s' . The states of the process are annotated with their timing distributions and each action-labelled transition is decorated with its probability to be executed.

In this paper we investigate the algorithmic complexity of various problems regarding the faster-than relation, emphasising their connection with classical algorithmic problems over Rabin’s probabilistic automata. In particular, we

prove that the faster-than problem over generic semi-Markov processes is undecidable and that it is Positivity-hard when restricted to processes with only one action label. The reduction from the Positivity problem is important because it relates the faster-than problem to the Skolem problem, an important problem in number theory, whose decidability status has been an open problem for at least 80 years [1, 12].

We show that undecidability for the faster-than problem can not be tackled even by approximation techniques: via the same connection with probabilistic automata we are able to prove that the faster-than problem can not be approximated up to a multiplicative constant. However, as a positive result, we show that a time-bounded variant of the faster-than problem, which compares processes up to a given finite time bound, although still undecidable, admits approximated solutions up to an *additive* constant over semi-Markov processes with slow residence-time distributions. These include the important cases of uniform and exponential distributions.

Finally, we present a **coNP** algorithm for solving the faster-than problem exactly over unambiguous semi-Markov processes, where a process is unambiguous if every transition to a next state is unambiguously determined by the label that it outputs.

A full version of the paper with proofs and additional material can be found in [14].

2 Semi-Markov Processes and Faster-than Relation

For a finite set S we let $\mathcal{D}(S)$ denote the set of subdistributions over S , i.e. functions $\delta : S \rightarrow [0, 1]$ such that $\sum_{s \in S} \delta(s) \leq 1$. The subset of total distributions is $\mathcal{D}_{\leq 1}(S)$. We let \mathbb{N} denote the natural numbers and $\mathbb{R}_{\geq 0}$ denote the non-negative real numbers. We equip $\mathbb{R}_{\geq 0}$ with the Borel σ -algebra \mathcal{B} , so that $(\mathbb{R}_{\geq 0}, \mathcal{B})$ is a measurable space. Let $\mathcal{D}(\mathbb{R}_{\geq 0})$ denote the set of (sub)distributions over $(\mathbb{R}_{\geq 0}, \mathcal{B})$, i.e. measures $\mu : \mathcal{B} \rightarrow [0, 1]$ such that $\mu(\mathbb{R}_{\geq 0}) \leq 1$. Throughout the paper we will write $\mu(t)$ for $\mu([0, t])$. To avoid confusion we will refer to μ in $\mathcal{D}(\mathbb{R}_{\geq 0})$ as timing distributions, and to δ in $\mathcal{D}(S)$ as distributions.

Definition 1 (Semi-Markov process). A semi-Markov process is a tuple $\mathcal{M} = (S, \text{Out}, \Delta, \rho)$ where

- S is a (finite) set of states,
- Out is a (finite) set of output labels,
- $\Delta : S \rightarrow \mathcal{D}(S \times \text{Out})$ is a transition function,
- $\rho : S \rightarrow \mathcal{D}(\mathbb{R}_{\geq 0})$ is a residence-time function.

The operational behaviour of a semi-Markov process can be described as follows. In a given state $s \in S$, the process fires a transition within time t with probability $\rho(s)(t)$, leading to the state $s' \in S$ while outputting the label $a \in \text{Out}$ with probability $\Delta(s)(s', a)$.

We aim at defining $\mathbb{P}_{\mathcal{M}}(s, w, t)$, the probability that from the state s , the output of the semi-Markov process \mathcal{M} within time t starts with the word w . It

is important to note here that time is accumulated: we sum together the time spent in all states along the way, and ask that this total time is less than the specified bound t .

In order to account for the accumulated time in the probability, we need the notion of convolution. The convolution of two timing distributions μ and ν is $\mu * \nu$ defined, for any Borel set $E \subseteq \mathbb{R}_{\geq 0}$, as follows

$$(\mu * \nu)(E) = \int_0^\infty \nu(E - x)\mu(dx) .$$

Convolution is both associative and commutative. Let X and Y be two independent random variables with timing distributions μ and ν , i.e. $\mathbb{P}(X \in E) = \mu(E)$ and $\mathbb{P}(Y \in E) = \nu(E)$, then $\mathbb{P}(X + Y \in E) = (\mu * \nu)(E)$.

Definition 2 (Probability). Consider a semi-Markov process \mathcal{M} . We define the timing distribution $\mathbb{P}_{\mathcal{M}}(s, w)$ inductively on w , as follows, for any word $w \in \text{Out}^*$, label $a \in \text{Out}$, and time $t \in \mathbb{R}_{\geq 0}$

$$\begin{aligned} \mathbb{P}_{\mathcal{M}}(s, \varepsilon)(t) &= 1 \\ \mathbb{P}_{\mathcal{M}}(s, aw)(t) &= \sum_{s' \in S} \Delta(s)(s', a) \cdot (\rho(s) * \mathbb{P}_{\mathcal{M}}(s', w))(t) . \end{aligned}$$

We will then write $\mathbb{P}_{\mathcal{M}}(s, w, t)$ to mean $\mathbb{P}_{\mathcal{M}}(s, w)(t)$.

2.1 Timed Comparisons

We introduce the following relation which will be the focus of our paper.

Definition 3 (Faster-than relation). Consider a semi-Markov process \mathcal{M} and two states s and s' . We say that s is faster than s' , denoted $s \preceq s'$, if for all words $w \in \text{Out}^*$, for all time $t \in \mathbb{R}_{\geq 0}$,

$$\mathbb{P}_{\mathcal{M}}(s, w, t) \geq \mathbb{P}_{\mathcal{M}}(s', w, t) .$$

The algorithmic problem we consider in this paper is the *faster-than problem*: given a semi-Markov process and two states s and s' , determine whether $s \preceq s'$.

2.2 Algorithmic Considerations

The definition we use for semi-Markov processes is very general, because we allow for any residence-time function. The aim of the paper is to give generic algorithmic results which apply to *effective* classes of timing distributions, a notion we define now. Recall that a residence-time function associates with each state a timing distribution. We first give some examples of classical timing distributions.

- The prime example is exponential distributions, defined by the timing distribution $\mu(t) = 1 - e^{-\lambda t}$ for some parameter $\lambda > 0$ usually called the rate.

- Another interesting example is piecewise polynomial distributions. Consider finitely many polynomials P_1, \dots, P_n and a finite set of pairwise disjoint intervals $I_1 \cup I_2 \cup \dots \cup I_n$ covering $[0, \infty)$ such that for every k , P_k is non-negative over I_k and $\sum_k \int_{I_k} P_k = 1$. This induces the timing distribution

$$\mu(t) = \sum_k \int_{I_k \cap [0, t]} P_k(t) .$$

- Another important special case of piecewise polynomial distributions are the uniform distributions with parameters $0 \leq a < b$.
- The simplest example is given by Dirac distributions defined for the parameter a by $\mu(E) = 1$ if a is in E , and 0 otherwise.

The following definition captures these examples, and more. For a class \mathcal{C} of timing distributions, we let $\text{Convex}(\mathcal{C})$ be the smallest class of timing distributions containing \mathcal{C} and closed under convex combinations, and similarly $\text{Conv}(\mathcal{C})$ adding closure under convolutions.

Lemma 4. *Let \mathcal{C} be a class of timing distributions. Consider a semi-Markov process \mathcal{M} whose residence-time function uses timing distributions from \mathcal{C} . Then, for any state $s \in \mathcal{M}$ and word $w \in \text{Out}^*$, $\text{IP}_{\mathcal{M}}(s, w) \in \text{Conv}(\mathcal{C})$.*

In the rest of the paper we will consider only distributions that are suitable for algorithmic manipulation. Clearly, we must be able to give them as input to a computational device, hence we assume they can be described by finitely many rational parameters. Moreover, we require that testing inequalities between them is decidable, since this is essential for determining the faster-than relation. The next definition formalises this intuition.

Definition 5 (Effective timing distributions). *A class \mathcal{C} of timing distributions is effective if, for any $\varepsilon \geq 0$, $b \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, and $\mu_1, \mu_2 \in \text{Conv}(\mathcal{C})$, it is decidable whether $\mu_1(t) \geq \mu_2(t) - \varepsilon$, for all $t \leq b$.*

Proposition 6. *The following classes of timing distributions are effective: exponential, piecewise polynomial, uniform, and Dirac distributions.*

We do not provide in the conference version a full proof of Proposition 6, as it is mostly folklore but rather tedious. In particular, for exponential and piecewise polynomial distributions one relies on decidability results for the existential theory of the reals [17], implying that the most demanding operations above can be performed in polynomial space [4].

Although in this paper we give algorithmic results for generic effective classes of timing distributions, the semi-Markov processes we will focus on have only finitely many states, and hence can only use finitely many timing distributions from the same class. For our decidability results we will therefore focus on finite classes of timing distributions.

Moreover, in our complexity analyses, we will always assume that the operations on the timing distributions have a unit cost.

3 Hardness Results

We start the technical part of this article by presenting a series of hardness results for semi-Markov processes inherited from Markov processes.

A Markov process is a tuple $\mathcal{M} = (S, \text{Out}, \Delta)$ consisting of a (finite) set of states S , a (finite) set of labels Out , and a transition function $\Delta : S \rightarrow \mathcal{D}(S \times \text{Out})$. For a Markov process \mathcal{M} we define the probability $\mathbb{P}_{\mathcal{M}}(s)$ on Out^* inductively, for $a \in \text{Out}$ and $w \in \text{Out}^*$, as follows

$$\mathbb{P}_{\mathcal{M}}(s, \varepsilon) = 1 \quad \text{and} \quad \mathbb{P}_{\mathcal{M}}(s, aw) = \sum_{s' \in S} \Delta(s)(s', a) \cdot \mathbb{P}_{\mathcal{M}}(s')(w) .$$

The faster-than relation \preceq for Markov processes is defined similarly to the case of semi-Markov processes: $s \preceq s'$ if, for all words w , $\mathbb{P}_{\mathcal{M}}(s, w) \geq \mathbb{P}_{\mathcal{M}}(s', w)$.

We show that the faster-than problem for Markov processes, and hence also for semi-Markov processes, is (i) undecidable, (ii) can not be multiplicatively approximated, and (iii) is Positivity-hard even over the restricted case of Markov processes with one single output label. These limitations shape and motivate our positive results, which will be the topic of the remaining sections.

We first explain how hardness results for Markov processes directly imply hardness results for semi-Markov processes. The following lemma formalises the two ways semi-Markov processes subsume Markov processes.

Lemma 7. *Consider a semi-Markov process $\mathcal{M} = (S, \text{Out}, \Delta, \rho)$ and its induced Markov process $\mathcal{M}' = (S, \text{Out}, \Delta)$.*

- If ρ is constant, i.e. for all s, s' we have $\rho(s) = \rho(s')$, then for all w , for all t , we have $\mathbb{P}_{\mathcal{M}}(s, w, t) = \mathbb{P}_{\mathcal{M}'}(s, w) \cdot \underbrace{(\rho(s) * \dots * \rho(s))}_{|w|\text{ times}}(t)$.
- If for all s , $\rho(s)$ is the Dirac distribution for 0, then for all w , for all t , we have $\mathbb{P}_{\mathcal{M}}(s, w, t) = \mathbb{P}_{\mathcal{M}'}(s, w)$.

In particular in both cases, the following holds: for s, s' two states, we have $s \preceq s'$ in \mathcal{M} if, and only if, $s \preceq s'$ in \mathcal{M}' .

The hardness results of this section will be based on a connection to Rabin's probabilistic automata. A probabilistic automaton is given by

$$\mathcal{A} = (Q, A, q_0, \Delta : Q \times A \rightarrow \mathcal{D}_{=1}(Q), F),$$

where Q is the set of states, A is the alphabet, q_0 is an initial state, Δ is the transition function, and F is a set of final or accepting states. Any probabilistic automaton \mathcal{A} induces the probability $\mathbb{P}_{\mathcal{A}}(w)$ that a run over $w \in A^*$ is accepting, i.e. starts in q_0 and ends in F .

The key property of probabilistic automata that we will exploit is the undecidability of the universality problem, which was proved in [13], see also [8]. The universality problem is as follows: given a probabilistic automaton \mathcal{A} , determine whether $\mathbb{P}_{\mathcal{A}}(w) \geq \frac{1}{2}$, for all nonempty words $w \in A^+$.

Given a probabilistic automaton \mathcal{A} we define the *derived Markov process* $\mathcal{M}(\mathcal{A})$ as follows. The set of states of $\mathcal{M}(\mathcal{A})$ is $Q \times \{\ell, r\} \cup \{\top\}$, where \top is a new state; the set of output labels is A , and the transition function Δ' is defined as follows, for $p, q \in Q$ and $a \in \text{Out}$:

$$\begin{aligned}\Delta'(p, \ell)((q, \ell), a) &= \frac{1}{2|A|} \Delta(p, a)(q) & \Delta'(p, \ell)(\top, a) &= \frac{1}{2} \text{ if } p \in F \\ \Delta'(p, r)((q, r), a) &= \frac{1}{2|A|} \Delta(p, a)(q) & \Delta'(p, r)(\top, a) &= \frac{1}{2}.\end{aligned}$$

Let $s = (q_0, \ell)$ and $s' = (q_0, r)$, where q_0 is the initial state of \mathcal{A} . Then, for the Markov process $\mathcal{M}(\mathcal{A})$, we can then verify the following equalities:

$$\mathbb{P}_{\mathcal{M}(\mathcal{A})}(s, wa) = \frac{1}{(2|A|)^{|w|}} \mathbb{P}_{\mathcal{A}}(w) \quad \text{and} \quad \mathbb{P}_{\mathcal{M}(\mathcal{A})}(s', wa) = \frac{1}{(2|A|)^{|w|}} \frac{1}{2}.$$

Theorem 8. *The faster-than problem is undecidable for Markov processes.*

We discuss three approaches to recover decidability. A first approach is to look for *structural restrictions* on the underlying graph. However, the undecidability result above for probabilistic automata is quite robust in this respect, as it already applies when the underlying graph is acyclic, meaning that the only loops are self-loops. In spite of this, we present in Sect. 5 an algorithm to solve the faster-than problem for *unambiguous* semi-Markov processes.

A second approach is to restrict the *observations*. Interestingly, specialising the construction above to only one output letter yields a reduction from the Positivity problem. Formally, the Positivity problem reads: given a linear recurrence sequence, are all terms of the sequence non-negative? It has been shown that the universality problem for probabilistic automata with one letter alphabet is equivalent to the Positivity problem [1]. Thus, using again the derived Markov process $\mathcal{M}(\mathcal{A})$ for a probabilistic automaton \mathcal{A} with only one label, we obtain the following result.

Theorem 9. *The faster-than problem is Positivity-hard over Markov processes with one output label.*

A third approach is *approximations*. However, we can exploit further the connection we made with probabilistic automata, obtaining an impossibility result for *multiplicative approximation*. We rely on the following celebrated theorem for probabilistic automata due to Condon and Lipton [5]. The following formulation of their theorem is described in detail in [6].

Theorem 10 [5]. *Let $0 < \alpha < \beta < 1$ be two constants. There is no algorithm which, given a probabilistic automaton \mathcal{A} ,*

- if for all w we have $\mathbb{P}_{\mathcal{A}}(w) \geq \beta$, returns YES,
- if there exists w such that $\mathbb{P}_{\mathcal{A}}(w) \leq \alpha$, returns NO.

Theorem 11. Let $0 < \varepsilon < \frac{1}{3}$ be a constant. There is no algorithm which, given a Markov process \mathcal{M} and two states s, s' ,

- if for all w we have $\mathbb{P}_{\mathcal{M}}(s, w) \geq \mathbb{P}_{\mathcal{M}}(s', w)$, returns YES,
- if there exists w such that $\mathbb{P}_{\mathcal{M}}(s, w) \leq \mathbb{P}_{\mathcal{M}}(s', w) \cdot (1 - \varepsilon)$, returns NO.

From these hardness results for Markov processes together with Lemma 7, we get the following hardness results for semi-Markov processes.

Corollary 12. The following holds for semi-Markov processes for any class of timing distributions.

- The faster-than problem is undecidable.
- The faster-than problem with only one output label is Positivity-hard.
- The faster-than problem can not be multiplicatively approximated.

4 Time-Bounded Additive Approximation

Instead of considering multiplicative approximation, we can also consider additive approximation, meaning that we want to decide whether for all w and t we have $\mathbb{P}_{\mathcal{M}}(s, w, t) \geq \mathbb{P}_{\mathcal{M}}(s', w, t) - \varepsilon$ for some constant $\varepsilon > 0$. In this section, we present an algorithm to solve the problem of approximating additively the faster-than relation with two assumptions:

- *time-bounded*: we only look at the behaviours up to a given bound b in $\mathbb{IR}_{\geq 0}$,
- *slow residence-time functions*: each transition takes *some* time to fire.

As we will show, the combination of these two assumptions imply that the relevant words have bounded length. This is in contrast to the impossibility of approximating the faster-than relation multiplicatively showed in Sect. 3.

More precisely, we consider the *time-bounded* variant of the faster-than problem: given a time bound $b \in \mathbb{IR}_{\geq 0}$, and two states s and s' in \mathcal{M} determine whether for all $t \leq b$ and w it holds that $\mathbb{P}_{\mathcal{M}}(s, w, t) \geq \mathbb{P}_{\mathcal{M}}(s', w, t)$.

We first observe that this restriction of the faster-than problem does not make any of the problems in Sect. 3 easier for semi-Markov processes. Indeed, if the residence-time functions are all Dirac distributions at 0, then all transitions are fired instantaneously, and the time-bounded restriction is immaterial. Thus we focus on distributions that do not fire instantaneously, as made precise by the following definition.

Definition 13 (Slow distributions). We say that a class \mathcal{C} of timing distributions is slow if for all finite subset \mathcal{C}_0 of \mathcal{C} , there exists a computable function $\varepsilon : \mathbb{IN} \times \mathbb{IR}_{\geq 0} \rightarrow [0, 1]$ such that for all n, t , and $\mu_1, \dots, \mu_n \in \text{Convex}(\mathcal{C}_0)$ we have $(\mu_1 * \dots * \mu_n)(t) \leq \varepsilon(n, t)$ and $\lim_{n \rightarrow \infty} \varepsilon(n, t) = 0$.

Given a slow and effective class \mathcal{C} of timing distributions, we can do additive approximation of the time-bounded faster-than problem in the following way.

We introduce the following notation. Fix a semi-Markov process \mathcal{M} . Let $\mathcal{C}_{\mathcal{M}} = \text{Convex}(\{\rho(s) \mid s \in S\})$, and $n \in \mathbb{N}$. We define the timing distribution $F_{\mathcal{M},n}$ by $F_{\mathcal{M},n}(t) = 1$ if $n = 0$ and otherwise

$$F_{\mathcal{M},n}(t) = \sup \{(\mu_1 * \dots * \mu_n)(t) \mid \mu_1, \dots, \mu_n \in \mathcal{C}_{\mathcal{M}}\}.$$

Lemma 14. *For all s and all w , we have $\mathbb{P}_{\mathcal{M}}(s, w) \leq F_{\mathcal{M},|w|}$.*

Theorem 15. *For a constant $\varepsilon > 0$, there exists an algorithm which, given a semi-Markov process \mathcal{M} with slow and effective timing distributions, two states s, s' , and a bound $b \in \mathbb{R}_{\geq 0}$, determines whether*

$$\forall w, \forall t \leq b, \mathbb{P}_{\mathcal{M}}(s, w, t) \geq \mathbb{P}_{\mathcal{M}}(s', w, t) - \varepsilon.$$

Proof. Let $\mathcal{C}_{\mathcal{M}} = \text{Convex}(\{\rho(s) \mid s \in S\})$, since S is finite there exists a computable function $\varepsilon : \mathbb{N} \times \mathbb{R}_{\geq 0} \rightarrow [0, 1]$ such that for all n, t , and $\mu_1, \dots, \mu_n \in \mathcal{C}_{\mathcal{M}}$ we have $(\mu_1 * \dots * \mu_n)(t) \leq \varepsilon(n, t)$ and $\lim_{n \rightarrow \infty} \varepsilon(n, t) = 0$. Given $\varepsilon > 0$, there exists N such that $\varepsilon(N, b) < \varepsilon$. For $n \geq N$. By assumption $(\mu_1 * \dots * \mu_n)(b) \leq \varepsilon(n, b) \leq \varepsilon(N, b) < \varepsilon$ for all $\mu_1, \dots, \mu_n \in \mathcal{C}_{\mathcal{M}}$. Taking the supremum over μ_1, \dots, μ_n , we then get $F_{\mathcal{M},n}(b) < \varepsilon$, and by Lemma 14, this means that for all w of length at least N , we have $\mathbb{P}_{\mathcal{M}}(s', w, b) < \varepsilon$. Hence it holds trivially that for all $t \leq b$ and w of length at least N , we have $\mathbb{P}_{\mathcal{M}}(s, w, t) \geq \mathbb{P}_{\mathcal{M}}(s', w, t) - \varepsilon$.

Thus the algorithm checks whether for all words of length less than N , for all $t \leq b$, we have $\mathbb{P}_{\mathcal{M}}(s, w, t) \geq \mathbb{P}_{\mathcal{M}}(s', w, t) - \varepsilon$, which is decidable thanks to the effectiveness of \mathcal{C} . \square

Next we show that there are interesting classes of timing distributions that are indeed slow. For this we introduce a class of timing distributions that are not just slow, but furthermore are guaranteed to converge to zero rapidly. We say that a timing distribution μ is *very slow* if there exists a computable function $\varepsilon : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$ such that $\lim_{t \rightarrow 0} \frac{\varepsilon(t)}{t} = 0$ and for all t , we have $\mu(t) \leq \varepsilon(t)$.

Theorem 16. *The following classes of timing distributions are slow: very slow, uniform, and exponential distributions.*

The proof of Theorem 16 depends on closed forms for the n -fold convolution of exponential distributions and uniform distributions, both of which converge to 0 as n goes to infinity. For exponential distributions, this closed form is the well-known Gamma distribution.

5 Unambiguous Semi-Markov Processes

In order to regain decidability of the faster-than relation, we can look at structurally simpler special cases of semi-Markov processes. Here we will focus on semi-Markov processes such that each output word induces at most one trace of states. More precisely, we will say that a semi-Markov process is *unambiguous* if for every state s and output label $a \in \text{Out}$, there exists at most one state s' such

that $\Delta(s)(s', a) \neq 0$. A related notion of bounded ambiguity has been utilised to obtain decidability results in the context of probabilistic automata [7]. We introduce the following notation for unambiguous semi-Markov processes: $T(s, w)$ is the state reached after emitting w from s .

Example 17. Figure 2 gives an example of an unambiguous semi-Markov process. For each of the three states, there is at most one state that can be reached by a given output label. However, there need not be a transition for each output label from every state. In this example, the state s_2 has no b -transition, so for instance $T(s_1, ab) = s_2$, but $T(s_1, abb)$ is undefined.

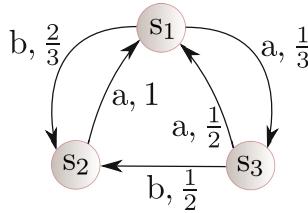


Fig. 2. An example of an unambiguous semi-Markov process.

Theorem 18. *The faster-than problem is decidable in **cONP** over unambiguous semi-Markov processes for all effective classes of timing distributions.*

Theorem 18 follows from the next proposition.

Proposition 19. *Consider an unambiguous semi-Markov process \mathcal{M} and two states s, s' . Let $L(s, s')$ be the set of loops reachable from (s, s') :*

$$\left\{ (p, p', v) \in S^2 \times \text{Out}^{\leq S^2} \mid \exists w \in \text{Out}^{\leq S^2}, \begin{array}{l} T(s, w) = p, T(s', w) = p', \\ T(p, v) = p, T(p', v) = p' \end{array} \right\}.$$

We have $s \preceq s'$ if, and only if

- for all $w \in \text{Out}^{\leq S^2}$, we have $\mathbb{P}_{\mathcal{M}}(s, w) \geq \mathbb{P}_{\mathcal{M}}(s', w)$, and
- for all $(p, p', v) \in L(s, s')$, we have $\mathbb{P}_{\mathcal{M}}(p, v) \geq \mathbb{P}_{\mathcal{M}}(p', v)$.

Before going into the proof, we explain how to use Proposition 19 to construct an algorithm solving the faster-than problem over unambiguous semi-Markov processes.

1. The first step is to compute $L(s, s')$, which can be done in polynomial time using a simple graph analysis,
2. The second step is to check the two properties, which both can be reduced to exponentially many queries of the form: $\mu_1 \geq \mu_2$ for μ_1, μ_2 in $\text{Conv}(\mathcal{C})$.

To obtain a **coNP** algorithm, in the second step we guess which of the two properties is not satisfied and a witness of polynomial length, which is either a word of quadratic length for the first property, or two states and a word of quadratic length for the second property.

Proof (of Proposition 19). (\implies) Assume that s is faster than s' and let (p, p') be in $L(s, s')$. There exist $w, v \in \text{Out}^*$ such that $T(s, w) = p$, $T(s', w) = p'$, $T(p, v) = p$, $T(p', v) = p'$. Let $n \in \mathbb{N}$. Since s is faster than s' , we have $\mathbb{P}_{\mathcal{M}}(s, wv^n) \geq \mathbb{P}_{\mathcal{M}}(s', wv^n)$. We have

$$\begin{aligned}\mathbb{P}_{\mathcal{M}}(s, wv^n) &= \mathbb{P}_{\mathcal{M}}(s, w) * \underbrace{\mathbb{P}_{\mathcal{M}}(p, v) * \cdots * \mathbb{P}_{\mathcal{M}}(p, v)}_{n \text{ times}} \\ \mathbb{P}_{\mathcal{M}}(s', wv^n) &= \mathbb{P}_{\mathcal{M}}(s', w) * \underbrace{\mathbb{P}_{\mathcal{M}}(p', v) * \cdots * \mathbb{P}_{\mathcal{M}}(p', v)}_{n \text{ times}} .\end{aligned}$$

Let $X_{s,w}$ be the random variable measuring the time elapsed from s emitting w . Similarly, we define $X_{p,v}$, $Y_{s',w}$ and $Y_{p',v}$. We have: for all $n \in \mathbb{N}$, for all t ,

$$\mathbb{P}_{\mathcal{M}}(X_{s,w} + nX_{p,v} \leq t) \geq \mathbb{P}_{\mathcal{M}}(Y_{s',w} + nY_{p',v} \leq t) ,$$

Dividing both sides by n yields

$$\mathbb{P}_{\mathcal{M}}\left(\frac{X_{s,w}}{n} + X_{p,v} \leq \frac{t}{n}\right) \geq \mathbb{P}_{\mathcal{M}}\left(\frac{Y_{s',w}}{n} + Y_{p',v} \leq \frac{t}{n}\right) .$$

We make the change of variables $x = \frac{t}{n}$: for all $n \in \mathbb{N}$, for all x we have

$$\mathbb{P}_{\mathcal{M}}\left(\frac{X_{s,w}}{n} + X_{p,v} \leq x\right) \geq \mathbb{P}_{\mathcal{M}}\left(\frac{Y_{s',w}}{n} + Y_{p',v} \leq x\right) .$$

Letting $n \rightarrow \infty$, we then obtain, for all x $\mathbb{P}_{\mathcal{M}}(X_{p,v} \leq x) \geq \mathbb{P}_{\mathcal{M}}(Y_{p',v} \leq x)$, which is equivalent to $\mathbb{P}_{\mathcal{M}}(p, v) \geq \mathbb{P}_{\mathcal{M}}(p', v)$.

(\Leftarrow) We prove that for all w , we have $\mathbb{P}_{\mathcal{M}}(s, w) \geq \mathbb{P}_{\mathcal{M}}(s', w)$ by induction on the length of w . For w of length at most S^2 , this is ensured by the first assumption. Let w be a word longer than S^2 . There exist two states p, p' such that p is reached by s and p' by s' after emitting i letters of w and again after emitting j letters of w , with j at most S^2 . Let $w = w_1 v w_2$ where v starts at position i and ends at position j . By construction (p, p', v) is in $L(s, s')$. We have

$$\begin{aligned}\mathbb{P}_{\mathcal{M}}(s, w) &= \mathbb{P}_{\mathcal{M}}(s, w_1) * \mathbb{P}_{\mathcal{M}}(p, v) * \mathbb{P}_{\mathcal{M}}(p, w_2) \\ &= \mathbb{P}_{\mathcal{M}}(s, w_1 w_2) * \mathbb{P}_{\mathcal{M}}(p, v) \\ &\geq \mathbb{P}_{\mathcal{M}}(s', w_1 w_2) * \mathbb{P}_{\mathcal{M}}(p', v) \quad (\text{inductive hypothesis}) \\ &= \mathbb{P}_{\mathcal{M}}(s', w) .\end{aligned}$$

□

6 Conclusion and Open Problems

We have introduced a trace-based relation on semi-Markov processes called the faster-than relation which asks that for any time bound, the probability of outputting any word within the time bound is higher in the faster process than in the slower process. We have shown through a connection to probabilistic automata that the faster-than relation is highly undecidable. It is undecidable in general, and remains Positivity-hard even restricting to processes with one output label. Furthermore, approximating the faster-than relation up to a multiplicative constant is shown to be impossible.

However, we constructed algorithms for special cases of the faster-than problem. We have shown that if one considers approximating up to an additive constant rather than a multiplicative constant, and if one gives a bound on the time up to which one is interested in comparing the two processes, then approximation can be done for timing distributions in which we are sure to spend some amount of time to take a transition. In addition, we have shown that the faster-than relation over unambiguous processes is decidable and in **coNP**.

In this paper, we have focused on the generative model, where the labels are treated as outputs. An alternative viewpoint would be to consider reactive models, where the labels are instead treated as inputs [18]. While all the undecidability and hardness results we have shown can also easily be shown to hold for reactive Markov processes, the same is not true for the algorithms we have constructed. It is non-trivial to extend these algorithms for the case of reactive semi-Markov processes: the main obstacle is that for reactive systems, one has to also handle schedulers, often uncountably many. It is therefore still an open question whether our decidability results carry over to reactive models.

References

1. Akshay, S., Antonopoulos, T., Ouaknine, J., Worrell, J.: Reachability problems for Markov chains. *Inf. Process. Lett.* **115**(2), 155–158 (2015)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Baier, C., Katoen, J.P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. *Inf. Comput.* **200**(2), 149–214 (2005)
4. Canny, J.F.: Some algebraic and geometric computations in PSPACE. In: STOC, ACM, pp. 460–467 (1988)
5. Condon, A., Lipton, R.J.: On the complexity of space bounded interactive proofs (extended abstract). In: FOCS (1989)
6. Fijalkow, N.: Undecidability results for probabilistic automata. *SIGLOG News* **4**(4), 10–17 (2017)
7. Fijalkow, N., Riveros, C., Worrell, J.: Probabilistic automata of bounded ambiguity. In: CONCUR, pp. 19:1–19:14 (2017)
8. Gimbert, H., Oualhadj, Y.: Probabilistic automata on finite words: decidable and undecidable problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 527–538. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14162-1_44

9. Guha, S., Narayan, C., Arun-Kumar, S.: On decidability of prebisimulation for timed automata. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 444–461. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_33
10. Lütten, G., Vogler, W.: A faster-than relation for asynchronous processes. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 262–276. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44685-0_18
11. Moller, F., Tofts, C.: Relating processes with respect to speed. In: Baeten, J.C.M., Groote, J.F. (eds.) CONCUR 1991. LNCS, vol. 527, pp. 424–438. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-54430-5_104
12. Ouaknine, J., Worrell, J.: Positivity problems for low-order linear recurrence sequences. In: SODA (2014)
13. Paz, A.: Introduction to Probabilistic Automata. Academic Press, New York (1971)
14. Pedersen, M.R., Fijalkow, N., Bacci, G., Larsen K.G., Mardare, R.: Timed comparisons of semi-Markov processes. CoRR (2017). <http://arxiv.org/abs/1711.10216>
15. Perman, M., Senegacnik, A., Tuma, M.: Semi-Markov models with an application to power-plant reliability analysis. IEEE Trans. Reliab. **46**(4), 526–532 (1997)
16. Pievatolo, A., Tironi, E., Valade, I.: Semi-Markov processes for power system reliability assessment with application to uninterruptible power supply. IEEE Trans. Power Syst. **19**(3), 1326–1333 (2004)
17. Tarski, A.: A decision method for elementary algebra and geometry. University of California Press, Berkeley (1951)
18. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. Inf. Comput. **121**(1), 59–80 (1995)



Handling Ties Correctly and Efficiently in Viterbi Training Using the Viterbi Semiring

Markus Saers^(✉) and Dekai Wu

Department of Computer Science and Engineering,
Human Language Technology Center,
The Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong
`{masaers,dekai}@cs.ust.hk`

Abstract. The handling of ties between equiprobable derivations during Viterbi training is often glossed over in research paper, whether they are broken randomly when they occur, or on an *ad-hoc* basis decided by the algorithm or implementation, or whether all equiprobable derivations are enumerated with the counts uniformly distributed among them, is left to the readers imagination. The first hurts rarely occurring rules, which run the risk of being randomly eliminated, the second suffers from algorithmic biases, and the last is correct but potentially very inefficient. We show that it is possible to Viterbi train correctly without enumerating all equiprobable best derivations. The method is analogous to expectation maximization, given that the automatic differentiation view is chosen over the reverse value/outside probability view, as the latter calculates the wrong quantity for reestimation under the Viterbi semiring. To get the automatic differentiation to work we devise an unbiased subderivative for the max function.

Keywords: Parsing · Viterbi training · Automatic differentiation
Deductive systems · Semiring parsing

1 Introduction

Conventional wisdom has it that expectation maximization is preferable over Viterbi training for reestimating generative models because all possible configurations of the hidden variables (paths through a lattice/trees in a forest) contribute proportionally to the reestimation, but Viterbi training has the potential of being significantly faster, since only the best paths/trees are needed. Goodman [6] showed that the necessary quantities for expectation maximization (EM) can be derived automatically in the form of reverse values, a generalization of backward/outside probabilities. Eisner *et al.* [5], Li and Eisner [8], and Smith [14] showed that automatic differentiation [2] of the sentence probability with respect to the rule probabilities is equivalent to reverse values, and Eisner [4]

pointed out that it is all essentially backpropagation as we know it from neural networks [10]. In this paper we show that the equivalence between reverse values and derivatives fails to generalize away from the probability semiring, specifically: applying the two methods to the Viterbi semiring gives very different results. We further show that the backpropagation approach is indeed as helpful to Viterbi training as it is to EM training.

Viterbi training (not to be confused with Viterbi decoding: finding the best path/tree in a lattice/forest) is similar to EM in that the rule probabilities are iteratively reestimated through counts obtained by reconstructing the hidden lattice/forest using the model itself. The difference lies in how the counts are obtained: EM lets the entire lattice/forest contribute proportionally, producing expected counts, whereas only the best path/tree is counted in Viterbi training. This makes it attractive because it opens up for more efficient algorithms to be used.

There is a hidden problem with performing Viterbi training based on Viterbi decoding, in that a decoder must return either (a) *a single* path/tree, or (b) *enumerate all* best paths/trees, which harms training of highly ambiguous models. These models can have a (potentially very) large number of best paths/trees, making (a) highly approximative, and (b) highly inefficient. It is not difficult to code around these problems, but as we show in this paper, it is possible to use automatic differentiation to do it correctly and efficiently with the exact same code as EM by substituting the probability semiring with the Viterbi semiring, and differentiating it carefully.

Although this paper is framed in terms of reestimation over parse forests or lattices, the techniques are equally valid for any model that can be described in terms of a deductive system (Sect. 2.3).

2 Background

This paper synthesizes expectation maximization and Viterbi training under deductive systems using automatic differentiation; all of which are known and established methods that we will briefly review in this section.

2.1 Expectation Maximization

Expectation maximization (EM) is a method for reestimating model parameters towards a local optimum of the marginal log likelihood of some data when part of the data is hidden and has to be reconstructed by the model [3]. Rather than optimizing the likelihood directly, the Q -function—the expectation of the log likelihood—is optimized iteratively in two steps: the **expectation** step (E) calculates the Q -function, and the **maximization** step (M) reestimates the model parameters to maximize Q :

$$\text{expectation:} \quad Q\left(\theta|\theta^{(t)}\right) = \mathbb{E}_{Z|X,\theta^{(t)}} [\log L(\theta; X, Z)] \quad (1)$$

$$\text{maximization:} \quad \theta^{(t+1)} = \underset{\theta}{\operatorname{argmax}} Q\left(\theta|\theta^{(t)}\right) \quad (2)$$

where X is the observed data, Z is the hidden data, and θ is the model parameters. Instead of calculating the Q -function explicitly, which is frequently intractable, the sufficient statistics q needed for the M-step is enough:

$$\text{expectation: } q^{(t)} = \mathbb{E}_Z \left[\log P(X, Z | \theta^{(t)}) \right] \quad (3)$$

$$\text{maximization: } \theta^{(t+1)} = \text{MLE} \left(q^{(t)} \right) \quad (4)$$

The sufficient statistics for context-free grammars are the fractional rule counts obtained from inside/forward and outside/backward probabilities which are calculated by integrating out the hidden structure (forest/lattice), and can be generalized as such:

$$q(A \rightarrow \phi^0 \cdots \phi^{R-1}) = p(A \rightarrow \phi^0 \cdots \phi^{R-1}) \sum_{w_{0..T} \in \mathcal{D}} \frac{1}{\alpha(S_{0,T})} \sum_{0 \leq i_0 < i_R \leq T} \beta(A_{i_0, i_R}) \sum_{i_0 < \cdots < i_R} \prod_{j=0}^{R-1} \alpha(\phi_{i_j, i_{j+1}}^j) \quad (5)$$

where p is the rule probability function, A is a nonterminal, ϕ^i is either a non-terminal or a terminal, and $\phi_{s,t}^i$ is that same (non-)terminal covering the span from s to t , R is the number of nonterminals and terminals on the right hand side of the rule, $w_{0..T} = w_0 w_1 \cdots w_{T-1}$ is a sentence in the data \mathcal{D} , α is the inside/forward probability, β is the backward/outside probability, and S is the dedicated start symbol of the grammar. Saers and Wu [11] and later Eisner [4] show that reverse values can be generalized to rules, replacing most of Eq. 5 with the rule's reverse value:

$$q(r) = p(r) \sum_{w_{0..T} \in \mathcal{D}} \frac{1}{\alpha(S_{0,T})} \beta(r) \quad (6)$$

2.2 Viterbi Training

Viterbi training consists of iteratively counting the number of times a rule occurs in the best path/tree of a sentence over the entire training data, and performing maximum likelihood estimation on those counts. This is guaranteed to approach a local optimum of the best path probability, at least for hidden Markov models [7]. Viterbi training is attractive, as it is often faster to derive only the best path/tree than to derive the entire lattice/forest. Describing it as closely as we can to EM, we have:

$$\text{counts: } c^{(t)} = \operatorname{argmax}_z \log P(X, Z = z | \theta^{(t)}) \quad (7)$$

$$\text{maximization: } \theta^{(t+1)} = \text{MLE} \left(c^{(t)} \right) \quad (8)$$

with the only difference that rather than integrating over the hidden data, we choose the hidden data configuration z that maximizes the likelihood.

Many NLP applications contain a best decoding component that provides the path or tree to collect Viterbi counts from, but there is an inherent problem hidden in this approach: It typically chooses a single path/tree—either randomly or worse: systematically—when there are multiple equiprobable paths/trees. For correct training, the rule counts have to be distributed equally to all equiprobable best paths/trees. It is possible to enumerate all such paths/trees, but the time complexity grows exponentially with the ambiguity of the model. In this paper, we show that it is possible to leverage the lattice/forest created during the forward/inside pass to efficiently and correctly solve this problem using automatic differentiation and the Viterbi semiring provided an unbiased subderivative of the max function is used.

Table 1. Common semirings.

Name	Domain	\oplus	\otimes	0	1
real	\mathbb{R}	+	\times	0	1
Boolean	{true, false}	\vee	\wedge	false	true
tropical	$\mathbb{R} \cup \infty$	min	+	∞	0
max-plus (“arctic”)	$\mathbb{R} \cup -\infty$	max	+	$-\infty$	0
probability	$[0, 1]$	+	\times	0	1
Viterbi	$[0, 1]$	max	\times	0	1
log probability	$[-\infty, 0]$	logadd	+	$-\infty$	0
negative log probability (cost)	$[0, \infty]$	logadd	+	∞	0
log Viterbi	$[-\infty, 0]$	max	+	$-\infty$	0
negative log Viterbi (min cost)	$[0, \infty]$	min	+	∞	0

2.3 Semiring Parsing and Deductive Systems

A deductive system [6, 9, 12] is a way to specify how a parser uses grammar rules to construct larger constituents from smaller constituents and the input sentence. A **conclusion** b may be reached iff all the I **conditions** a_0, a_1, \dots, a_{I-1} are met. This is written as an **inference rule**:

$$\frac{a_0, a_1, \dots, a_{I-1}}{b}$$

Conclusions are **items** (partial results such as labeled spans), and conditions are either other conclusions or **axioms** (grammar rules).

Semirings are best understood as generalizations of addition and multiplication. Formally they are tuples $(\mathcal{A}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, where \mathcal{A} is the domain, \oplus is a generalization of addition, and \otimes is a generalization of multiplication, with

identity elements **0** and **1** respectively. Table 1 shows several semirings with different usages: The real semiring is conventional math, the Boolean is symbolic logic. The tropical [13] and max-plus [1] semirings are closely related in that their domains can be thought of as the negative logarithms and logarithms of the real numbers respectively. The probability semiring is the real semiring over the domain of valid probabilities, and the Viterbi semiring—of interest to this paper—is the same but with addition replaced with the maxoperator. In practice, we rarely deal with probabilities in the real domain due to underflow problems, but instead with (negative) log probabilities that rely on the logadd operator: $\text{logadd}(x, y) \equiv \log_b(b^x + b^y)$. The corresponding (negative) log Viterbi semiring replaces the logadd-operator with the (min) max operator. The choice between raw logarithms and negated logarithms is mostly a matter of taste, but there is a nice interpretation of negative log probabilities as costs that are higher for unlikely events and lower for likely events.

Intuitively, deductive systems and semirings can be understood as a generalization of Boolean logic with arbitrary **values** instead of *true/false*. We can use the generalized semiring operators to compute the value α , corresponding to inside/forward probabilities of a conclusion, as:

$$\alpha(b) = \underbrace{\bigoplus_{\substack{a_0, \dots, a_{I-1} \\ b}}}_{\beta} \bigotimes_{i=0}^{I-1} \alpha(a_i) \quad (9)$$

One advantage with this generalization is that **reverse values**, β s, corresponding to backward/outside probabilities, can be derived from values:

$$\beta(a_i) = \underbrace{\bigoplus_{\substack{a_0, \dots, a_i, \dots, a_{I-1} \\ b}}}_{\beta} \beta(b) \otimes \bigotimes_{j=0}^{I-1} \begin{cases} \alpha(a_j) & \text{if } i \neq j \\ \mathbf{1} & \text{otherwise} \end{cases} \quad (10)$$

where the reverse value of the goal item is assumed to be **1**.

2.4 Viterbi Training with Equiprobable Derivations

Viterbi training relies on finding the best derivation of a deductive system, and counting each rule used in that derivation once. Finding only the best derivation is more efficient because more efficient search algorithms can be used, and because, when operating in log domain (which is typically necessary to avoid underflow problems), the operations themselves are faster: the logadd procedure requires calls to log and exp, both relatively heavy functions, whereas max and min are built-in CPU instructions. One problem with Viterbi training that is often glossed over is that grammars frequently contain ambiguity, and there is a real risk of having multiple equiprobable best derivations. In practice, implementations typically solve this by choosing one to be the best on an *ad-hoc* basis, although it is understood that the choice should be random in order to avoid biasing the training. With enough observations any true ambiguity will be preserved by the random selection. This sounds good until you realize that counts

are integers, and that observing a phenomenon an uneven number of times will drive likelihood of the unfavored interpretation to zero in the next iteration.

Imagine for example the Swedish noun *stegen*, which has two interpretations: *steg+en* ‘the steps’ and *stege+n* ‘the ladder’. Suppose that we have a CFG for noun phrases containing the following eight rules with their associated probabilities initialized to be an informative example:

$$\begin{aligned} \dots \\ p(\text{NP} \rightarrow \text{NN}^{\text{SG_DET}}) &= 0.05 \\ p(\text{NP} \rightarrow \text{NN}^{\text{PL_DET}}) &= 0.05 \\ p(\text{NP} \rightarrow \text{Art}^{\text{SG}} \text{NN}^{\text{SG_DET}}) &= 0.15 \\ p(\text{NP} \rightarrow \text{Art}^{\text{PL}} \text{NN}^{\text{PL_DET}}) &= 0.10 \\ p(\text{NN}^{\text{SG_DET}} \rightarrow \text{stegen}) &= 0.03 \\ p(\text{NN}^{\text{PL_DET}} \rightarrow \text{stegen}) &= 0.03 \\ p(\text{Art}^{\text{SG}} \rightarrow \text{den}) &= 0.50 \\ p(\text{Art}^{\text{PL}} \rightarrow \text{de}) &= 1.00 \\ \dots \end{aligned} \tag{11}$$

Suppose also that the grammar is ambiguous, so that $p(\text{NP} \rightarrow \text{NN}^{\text{SG_DET}}) = p(\text{NP} \rightarrow \text{NN}^{\text{PL_DET}})$ and $p(\text{NN}^{\text{SG_DET}} \rightarrow \text{stegen}) = p(\text{NN}^{\text{PL_DET}} \rightarrow \text{stegen})$. Our training data contains many examples of noun phrases with articles, but only one example of a lone noun: *stegen*. Even if the ties are broken randomly, we will only ever get counts, and thus nonzero value for either $\text{NP} \rightarrow \text{NN}^{\text{SG_DET}}$ and $\text{NN}^{\text{SG_DET}} \rightarrow \text{stegen}$, or $\text{NP} \rightarrow \text{NN}^{\text{PL_DET}}$ and $\text{NN}^{\text{PL_DET}} \rightarrow \text{stegen}$, not both; the information that *stegen* is ambiguous will be lost. Now suppose our test data contains the noun phrases *de stegen* ‘those steps’ and *den stegen* ‘that ladder’; we will only be able to assign a nonzero probability to one or the other, depending on whether we trained the grammar to treat *stegen* as a singular or plural noun. In this paper we will show that automatic differentiation under the Viterbi semiring preserves this type of ambiguity provided that an unbiased subderivative of max is used.

2.5 Automatic Differentiation

As pointed out in Li and Eisner [8] and further explored in Smith [14] it is possible to view the expectation step in expectation maximization as a case of automatic differentiation in the reverse mode [2], which makes it essentially identical to backpropagation [10], but over a lattice/forest rather than a neural network [4]. With this view, reverse values of rules are equivalent to partial derivatives of the goal value with respect to rule probabilities:

$$\beta(r) = \frac{\partial \alpha(S_{0,T})}{\partial \alpha(r)} \tag{12}$$

We can use the chain rule to aggregate the derivatives of all consequences that any one condition can lead to:

$$\frac{\partial \alpha(S_{0,T})}{\partial \alpha(a_i)} = \sum_{\substack{e = \frac{a_0, \dots, a_i, \dots, a_{I-1}}{b}}} \frac{\partial \alpha(S_{0,T})}{\partial \alpha(b)} \frac{\sum_{\substack{e' = \frac{a_0, \dots, a_{K-1}}{b}}} \alpha(e')}{\partial \alpha(e)} \frac{\partial \prod_{j=0}^{I-1} \alpha(a_j)}{\partial \alpha(a_i)} \quad (13)$$

which calculates the same quantity as traditional reverse values, which we get by substituting sum and product for their generalized place holders in Eq. 10:

$$\beta(a_i) = \underbrace{\sum_{\substack{a_0, \dots, a_i, \dots, a_{I-1}}}_{b}}_{\beta} \beta(b) \prod_{j=0}^{I-1} \begin{cases} \alpha(a_j) & \text{if } i \neq j \\ 1 & \text{otherwise} \end{cases} \quad (14)$$

The equality in Eq. 12 can be understood by inspecting the summation in Eq. 13 and noting that (a) the first factor is $\beta(b)$ provided that the equality holds (equal to the first factor in the summation in Eq. 14), (b) the second factor will always be 1 since it is the derivative of a sum with respect to one of its terms, and that (c) the third factor is the derivative of a product with respect to one of its factors, which is equivalent to the product of all the other factors (equal to the second factor in the summation in Eq. 14).

Relating this back of EM reestimation, we can reformulate Eq. 6 as:

$$q(r) = p(r) \sum_{w_{0..T} \in \mathcal{D}} \frac{1}{\alpha(S_{0,T})} \frac{\partial \alpha(S_{0,T})}{\partial \alpha(r)} \quad (15)$$

3 Reverse Values \neq Derivatives

As we saw in Sect. 2.5, both reverse values and automatic differentiation can be used in EM-training, as they are equivalent under the real semiring; in this section we show that this equivalence fails to hold for the Viterbi semiring. First, we observe that values under the Viterbi semiring correspond to the most likely path/tree leading to a particular conclusion, so the value calculated for the goal item is the probability of the best path/tree. Next, we construct an expression for the derivative of a condition under the Viterbi semiring. Following Eq. 13 we get:

$$\frac{\partial \alpha(S_{0,T})}{\partial \alpha(a_i)} = \sum_{\substack{e = \frac{a_0, \dots, a_i, \dots, a_{I-1}}{b}}} \frac{\partial \alpha(S_{0,T})}{\partial \alpha(b)} \frac{\max_{\substack{e' = \frac{a_0, \dots, a_{K-1}}{b}}} \alpha(e')}{\partial \alpha(e)} \frac{\partial \prod_{j=0}^{I-1} \alpha(a_j)}{\partial \alpha(a_i)} \quad (16)$$

Note that only the inner sum is replaced by a max, the outer sum is part of how derivatives are accumulated and has nothing to do with the semiring operators. Further note that we use max as an iterated binary operator like sum or product.

If we instead substitute the generalized operators in Eq. 10 with the Viterbi operators max and product we get:

$$\beta(a_i) = \underbrace{\max_{\substack{a_0, \dots, a_i, \dots, a_{I-1}}}_{b}}_{\beta} \beta(b) \prod_{j=0}^{I-1} \begin{cases} \alpha(a_j) & \text{if } i \neq j \\ 1 & \text{otherwise} \end{cases} \quad (17)$$

Equations 16 and 17 have very different forms, and there is no reason to believe that they would calculate the same quantity. The easiest way to see why they are different is to note that the second factor in the summation in Eq. 16 allows the entire term to become zero (whenever there is a competing way to arrive at b that has higher probability), whereas Eq. 17 is nonzero for all axioms that are used in the derivation.

4 Viterbi Training with Derivatives

Reverse values are unrelated to the rule counts needed for Viterbi training, but we show in this section that derivatives can be used to calculate counts for Viterbi training, and that the mechanism for doing so is the same as for how fractional counts but in a different semiring.

In the case when there is exactly one best derivation, its probability $\alpha(S_{0,T})$ is the product of the probability of all grammar rules used in that derivation. Designating the rules used in a derivation of sentence $w_{0..T}$ as r_0, r_1, \dots, r_k we have:

$$c_{w_{0..T}}(r_i) = p(r_i) \frac{1}{\alpha(S_{0,T})} \frac{\partial \alpha(S_{0,T})}{\partial p(r_i)} = \frac{p(r_i)}{\prod_{j=0}^{k-1} p(r_j)} \frac{\partial \prod_{j=0}^{k-1} p(r_j)}{\partial p(r_i)} = \frac{p(r_i)}{p(r_i)} = 1 \quad (18)$$

which is what we would expect: every rule in the best derivation of a single sentence gets a count of one. More generally, the rule counts have the exact same form as the fractional counts in EM, but under the Viterbi semiring rather than the probability semiring:

$$c(r) = p(r) \sum_{w_{0..T} \in \mathcal{D}} \frac{1}{\alpha(S_{0,T})} \frac{\partial \alpha(S_{0,T})}{\partial \alpha(r)} \quad (19)$$

Calculating the derivatives of the max operator does, however, pose a problem for sentences where there are multiple equiprobable (sub-)derivations because the derivative of max is undefined for equal arguments. We can get around this by using the *subderivative*, which allows us to distribute the results between the two maximizers. But with the subderivative we have to decide *how* to distribute the results between the two maximizers. We can characterize the distribution of the results with a parameter $0 \leq \lambda \leq 1$, and define our subderivative as:

$$\frac{\partial \max(x, y)}{\partial x} = \begin{cases} 1 & \text{if } x > y \\ \lambda & \text{if } x = y \\ 0 & \text{if } x < y \end{cases} \quad \text{and} \quad \frac{\partial \max(x, y)}{\partial y} = \begin{cases} 1 & \text{if } x > y \\ 1 - \lambda & \text{if } x = y \\ 0 & \text{if } x < y \end{cases} \quad (20)$$

5 Example

Only an unbiased subderivative for max will preserve genuine ambiguity found in the data, and to show the mechanism, we will return to our example CFG from 11. We have two derivations for the Swedish noun phrase *stegen*:

$$\frac{p(\text{NN}^{\text{SG-DET}} \rightarrow \text{stegen})}{\text{NN}_{0,1}^{\text{SG-DET}}}, p(\text{NP} \rightarrow \text{NN}^{\text{SG-DET}}) \quad (21)$$

$$\frac{p(\text{NP} \rightarrow \text{NN}^{\text{SG-DET}})}{\text{NP}_{0,1}}$$

and

$$\frac{p(\text{NN}^{\text{PL-DET}} \rightarrow \text{stegen})}{\text{NN}_{0,1}^{\text{PL-DET}}}, p(\text{NP} \rightarrow \text{NN}^{\text{PL-DET}}) \quad (22)$$

$$\frac{p(\text{NP} \rightarrow \text{NN}^{\text{PL-DET}})}{\text{NP}_{0,1}}$$

The value of the noun phrase $\text{NP}_{0,1}$ is:

$$\alpha(\text{NP}_{0,1}) = \max \left(\frac{p(\text{NP} \rightarrow \text{NN}^{\text{SG-DET}}) p(\text{NN}^{\text{SG-DET}} \rightarrow \text{stegen})}{p(\text{NP} \rightarrow \text{NN}^{\text{PL-DET}}) p(\text{NN}^{\text{PL-DET}} \rightarrow \text{stegen})}, \right) \quad (23)$$

where both arguments to max are equal, which is the property that we want to preserve, since the data lacks any evidence for preferring one over the other. We can work out the counts of a rule $r_1 = \text{NN}^{\text{SG-DET}} \rightarrow \text{stegen}$ with its companion rule $r_2 = \text{NP} \rightarrow \text{NN}^{\text{SG-DET}}$ as follows:

$$c(r_1) = p(r_1) \frac{1}{\alpha(\text{NP}_{0,1})} \frac{\partial \alpha(\text{NP}_{0,1})}{\partial p(r_1)} \quad (24)$$

$$= p(r_1) \frac{1}{\alpha(\text{NP}_{0,1})} \frac{\partial \alpha(\text{NP}_{0,1})}{\partial p(r_1) p(r_2)} \frac{\partial p(r_1) p(r_2)}{\partial p(r_1)} \quad (25)$$

$$= p(r_1) \frac{1}{p(r_1) p(r_2)} \lambda p(r_2) \quad (26)$$

$$= \lambda \quad (27)$$

conversely for the other rules we have:

$$c(\text{NP} \rightarrow \text{NN}^{\text{SG-DET}}) = \lambda, \quad (28)$$

$$c(\text{NP} \rightarrow \text{NN}^{\text{PL-DET}}) = 1 - \lambda, \quad (29)$$

$$c(\text{NN}^{\text{PL-DET}} \rightarrow \text{stegen}) = 1 - \lambda \quad (30)$$

as we can see, setting $\lambda = 0$ or $\lambda = 1$ recreates the behavior of choosing one or the other, whereas setting $\lambda = 0.5$ preserves the desired ambiguity. Since setting λ to anything but 0.5 will cause the winner to be sole maximizer in the next iteration of training, any value other than 0.5 has the same effect as setting it to 0 or 1. We call the subderivative that preserve the ambiguity **unbiased**.

After one iteration of training, the rules of interest would have their probabilities altered depending on λ such that:

$$\begin{array}{ll} \lambda = 1 & \lambda = 0.5 \\ p(\text{NP} \rightarrow \text{NN}^{\text{SG_DET}}) = 0.10 & p(\text{NP} \rightarrow \text{NN}^{\text{SG_DET}}) = 0.05 \\ p(\text{NP} \rightarrow \text{NN}^{\text{PL_DET}}) = 0.00 & p(\text{NP} \rightarrow \text{NN}^{\text{PL_DET}}) = 0.05 \\ p(\text{NN}^{\text{SG_DET}} \rightarrow \text{stegen}) = 0.06 & p(\text{NN}^{\text{SG_DET}} \rightarrow \text{stegen}) = 0.03 \\ p(\text{NN}^{\text{PL_DET}} \rightarrow \text{stegen}) = 0.00 & p(\text{NN}^{\text{PL_DET}} \rightarrow \text{stegen}) = 0.03 \end{array}$$

Looking at the derivations of our test sentences, the grammar trained with the unbiased subderivative give non-zero probabilities to both *de stegen* ‘those steps’:

$$\frac{p(\text{Art}^{\text{PL}} \rightarrow \text{de}), p(\text{NN}^{\text{PL_DET}} \rightarrow \text{stegen})}{\text{Art}_{0,1}^{\text{PL}}, \text{NN}_{1,2}^{\text{PL_DET}}}, p(\text{NP} \rightarrow \text{Art}^{\text{PL}} \text{NN}^{\text{PL_DET}}) \quad (31)$$

$$\frac{}{\text{NP}_{0,2}}$$

and *den stegen* ‘that ladder’:

$$\frac{p(\text{Art}^{\text{SG}} \rightarrow \text{den}), p(\text{NN}^{\text{SG_DET}} \rightarrow \text{stegen})}{\text{Art}_{0,1}^{\text{SG}}, \text{NN}_{1,2}^{\text{SG_DET}}}, p(\text{NP} \rightarrow \text{Art}^{\text{SG}} \text{NN}^{\text{SG_DET}}) \quad (32)$$

$$\frac{}{\text{NP}_{0,2}}$$

whereas the biased subderivative assigns zero probability to derivation Eq. 31.

The unbiased subderivative of max has a problem when being generalized to iterated binary operations. Consider $\max(a, b, c)$ where $a = b = c$. If we binarize it as $\max(a, \max(b, c))$,

$$\frac{\partial \max(a, \max(b, c))}{\partial a} = \lambda \neq \frac{\partial \max(a, \max(b, c))}{\partial b} = \lambda(1 - \lambda) \quad (33)$$

Luckily, we can generalize the unbiased subderivative to distribute the mass uniformly to one or more arguments:

$$\frac{\partial \max_{k=0}^{K-1} x_k}{\partial x_i} = \begin{cases} \frac{1}{\sum_{j=0}^{K-1} \left\{ \begin{array}{l} 1 \text{ if } x_j = \max_{k=0}^{K-1} x_k \\ 0 \text{ otherwise} \end{array} \right\}} & \text{if } x_i = \max_{k=0}^{K-1} x_k \\ 0 & \text{otherwise} \end{cases} \quad (34)$$

The time complexity for this derivative of max is still linear in K .

6 With Respect to the Entire Data Set

It turns out that the derivatives have another advantage, as we can generalize the calculation away from individual sentences and apply them to the entire data

set; specifically, the (fractional) counts are obtained when taking the partial derivative of rule probabilities with respect to the logarithm of the probability of the data:

$$p(r) \frac{\partial \log \prod_{w_{0..T} \in \mathcal{D}} \alpha(S_{0,T})}{\partial p(r)} = p(r) \frac{\partial \sum_{w_{0..T} \in \mathcal{D}} \log \alpha(S_{0,T})}{\partial p(r)} \quad (35)$$

$$= p(r) \sum_{w_{0..T} \in \mathcal{D}} \frac{\partial \log \alpha(S_{0,T})}{\partial p(r)} \quad (36)$$

$$= p(r) \sum_{w_{0..T} \in \mathcal{D}} \frac{\partial \log \alpha(S_{0,T})}{\partial \alpha(S_{0,T})} \frac{\partial \alpha(S_{0,T})}{\partial p(r)} \quad (37)$$

$$= p(r) \sum_{w_{0..T} \in \mathcal{D}} \frac{1}{\alpha(S_{0,T})} \frac{\partial \alpha(S_{0,T})}{\partial p(r)} \quad (38)$$

$$= c(r) \quad (39)$$

We get the counts needed for Viterbi training when differentiating under the Viterbi semiring, and the fractional counts or sufficient statistics needed for expectation maximization when differentiating under the probability semiring. This gives an interesting unified view of training as applying maximum likelihood estimation to “expected rule probability gradient”.

7 Conclusions

We have showed that the equality between a reverse pass and automatic differentiation, which exists for the probability semiring, fails to hold for the Viterbi semiring, and that automatic differentiation, in contrast to reverse values, gives the counts needed for Viterbi training. The differentiation approach is the same for both expectation maximization and Viterbi training except for the semiring used, opening up great opportunities for code reuse in implementations. We further highlighted a problem with the intuitive way of doing Viterbi training, in that ambiguous models, whose best paths/trees should be counted proportionally to not loose important information, are hard to count correctly and efficiently; again, just using automatic differentiation solves the problem, provided that an unbiased subderivative of max is used.

Acknowledgements. This material is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) under LORELEI contract HR0011-15-C-0114, BOLT contracts HR0011-12-C-0014 and HR0011-12-C-0016, and GALE contracts HR0011-06-C-0022 and HR0011-06-C-0023; by the European Union under the Horizon 2020 grant agreement 645452 (QT21) and FP7 grant agreement 287658; and by the Hong Kong Research Grants Council (RGC) research grants GRF16210714, GRF16214315, GRF620811 and GRF621008. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the EU, or RGC. The authors would also like to thank the anonymous reviewers for valuable feedback.

References

1. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and Linearity: An Algebra For Discrete Event Systems. Wiley Series in Probability and Mathematical Statistics. Wiley, Chichester (1992)
2. Corliss, G., Faure, C., Griewank, A., Hascoët, L., Naumann, U. (eds.): Automatic Differentiation of Algorithms: From Simulation to Optimization. Springer, New York (2002). <https://doi.org/10.1007/978-1-4613-0075-5>
3. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc.: Ser. B (Methodol.)* **39**(1), 1–38 (1977)
4. Eisner, J.: Inside-outside and forward-backward algorithms are just backprop. In: Proceedings of the EMNLP Workshop on Structured Prediction for NLP, Austin, Texas, November 2016
5. Eisner, J., Goldlust, E., Smith, N.A.: Compiling comp Ling: weighted dynamic programming and the Dyna language. In: Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP), Vancouver, Canada, pp. 281–290, October 2005
6. Goodman, J.: Semiring parsing. *Comput. Linguist.* **25**(4), 573–605 (1999)
7. Juang, B.H., Rabiner, L.R.: The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Trans. Acoust. Speech Signal Process.* **38**, 1639–1641 (1990)
8. Li, Z., Eisner, J.: First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In: 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009), Singapore, pp. 40–51, August 2009
9. Pereira, F.C.N., Warren, D.H.D.: Parsing as deduction. In: Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL 1983), Cambridge, Massachusetts, pp. 137–144, June 1983
10. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986)
11. Saers, M., Wu, D.: Reestimation of reified rules in semiring parsing and biparsing. In: Fifth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-5), Portland, Oregon, pp. 70–78, June 2011
12. Shieber, S.M., Schabes, Y., Pereira, F.C.: Principles and implementation of deductive parsing. *J. Logic Program.* **24**(1–2), 3–36 (1995)
13. Simon, I.: Recognizable sets with multiplicities in the tropical semiring. In: Chytil, M.P., Koubek, V., Janiga, L. (eds.) MFCS 1988. LNCS, vol. 324, pp. 107–120. Springer, Heidelberg (1988). <https://doi.org/10.1007/BFb0017135>
14. Smith, N.A.: Linguistic structure prediction. *Synth. Lect. Hum. Lang. Technol.* **4**(2), 1–274 (2011)



Over-Approximative Petri Net Synthesis for Restricted Subclasses of Nets

Uli Schlachter^(✉)

Department of Computing Science, Universität Oldenburg, Oldenburg, Germany
uli.schlachter@informatik.uni-oldenburg.de

Abstract. We show that, given a finite lts, there is a minimal bounded Petri net over-approximation according to a structural preorder and present an algorithm to compute this over-approximation. This result is extended to subclasses of nets, namely pure Petri nets, plain Petri nets, T-nets, and marked graphs, plus combinations of these properties.

Keywords: Petri net synthesis · Petri net properties · Region theory

1 Introduction

When building a system, the behaviour of the desired system is described in a specification. This specification is then used to actually produce, or *synthesise*, a system, preferably automatically. In Petri net synthesis, a specification can be a labelled transition system (lts) and the system should be a Petri net whose reachability graph is isomorphic to the given lts.

The basis for Petri net synthesis are *regions* of the lts, which were introduced in [8] for elementary nets and later extended to a more general setting of Petri nets [1]. Besides these general results, there are also specialised algorithms for the synthesis of certain subclasses of Petri nets, such as [4, 5, 10]. An overview of Petri net synthesis can be found in [2, 3].

Not all possible behaviours can be generated by Petri nets. For example, if two sequences of labels differ only in the order of events, then in any Petri net they must both reach the same marking by the well-known marking equation. Thus, a specification which requires an action a to be enabled after bc , but disabled after cb , cannot be solved by a Petri net. A possible way to deal with unsatisfiable specifications is over-approximation. Instead of solving the input exactly, some extra behaviour is added, preferably in a minimal way. In the example, we would also allow action a after cb .

There is a lot of research on over-approximating languages via Petri nets [6, 7, 9, 12]. In this case, *minimality* can be understood w.r.t. language inclusion, i.e. for a given language L , a Petri net N with $L \subseteq L(N)$ is wanted, so that for all Petri nets N' with $L \subseteq L(N')$ also $L(N) \subseteq L(N')$ holds.

The author is supported by the German Research Foundation (DFG) project ARS (Algorithms for Reengineering and Synthesis), reference number Be 1267/15-1.

In the present paper, we investigate a structural form of minimal over-approximation originating from [11], namely two lts A and B satisfy $A \sqsubseteq B$ if there is an edge-preserving homomorphism from the states of A to the states of B . Additionally, not only general Petri nets are considered, but also some restricted subclasses. These subclasses are combinations of the properties pure, plain, T-net and marked graph, which will be defined in Sect. 5.

Petri net synthesis solves so-called *separation problems*. Our algorithm is based on this. For a given lts, all unsolvable separation problems are dealt with by modifying the lts. Since these modifications can cause new unsolvable problems, they are repeated until no unsolvable separation problems remain.

The investigated problem of structural minimal over-approximation of lts with Petri net came from our recent result on realising modal transition systems with Petri nets [11]. There, a brute-force approach was used that can only be employed for k -bounded Petri nets. The present paper complements this work with a more efficient approach for over-approximation that is also more general.

Our algorithm can also be used to minimally over-approximate a given language with respect to language inclusion by using a so-called *limited unfolding* [2] as a pre-processing step and then using our new algorithm on the resulting lts. While this specific problem is already solved [7,9], this shows that our algorithm solves a more general problem, using a finer preorder than language inclusion.

The present paper is structured as follows: Sect. 2 introduces the basic concepts and Sect. 3 recapitulates region theory. In Sect. 4, an algorithm to compute a minimal over-approximation of an lts is presented. Section 5 extends this result to subclasses of Petri nets and Sect. 6 concludes the paper.

2 Labelled Transition Systems and Petri Nets

In this section we recall some definitions, first for labelled transition systems and then for Petri nets. Examples for the following definitions are given in Fig. 1.

An *lts* (*labelled transition system with initial state*) is a quadruple $A = (S, T, \delta, s_0)$, where S is a set of *states* with $s_0 \in S$ an *initial state*, T is the set of *labels* with $S \cap T = \emptyset$, and $\delta \subseteq S \times T \times S$ is the *edge relation*. For states $s, s' \in S$ and a sequence $w = t_0 t_1 \dots t_n \in T^*$, we write $s[w]s'$ if there are states $s_1, s_2, \dots, s_{n+1} \in S$ with $s_1 = s$, $s_{n+1} = s'$, and $(s_i, t_i, s_{i+1}) \in \delta$ for all $1 \leq i \leq n$. Note that $s[\varepsilon]s$ holds for all $s \in S$. When $s[w]s'$ holds, we write $s[w]$ to express that a suitable s' exists.

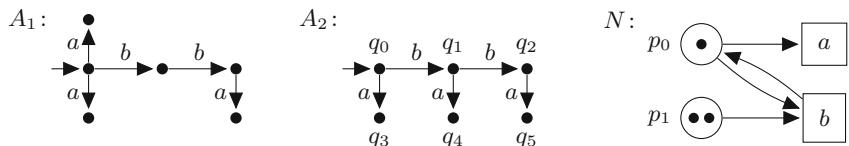


Fig. 1. The lts A_1 is reachable and finite, while the lts A_2 is additionally deterministic. The relation $A_1 \sqsubseteq A_2$ holds. The Petri net N solves A_2 .

The *language* of an lts is $L(A) = \{w \in T^* \mid s_0[w]\}$. A state $s \in S$ is *reachable* if a sequence $w \in T^*$ exists so that $s_0[w]s$. An lts is *reachable* if all of its states are reachable. An lts is *finite* if S and T (and hence also δ) are finite sets. It is *deterministic* if $s[a]s'$ and $s[a]s''$ implies $s' = s''$ for all $a \in T$ and $s, s', s'' \in S$.

For two lts $A = (S_A, T_A, \delta_A, s_{0,A})$ and $B = (S_B, T_B, \delta_B, s_{0,B})$, an *lts homomorphism from A to B* is a function $f: S_A \rightarrow S_B$ so that $f(s_{0,A}) = s_{0,B}$ and for all $(s, t, s') \in \delta_A$ also $(f(s), t, f(s')) \in \delta_B$. If such an lts homomorphism f exists, we write $A \sqsubseteq B$ (via f). \sqsubseteq is reflexive (with the identity homomorphism id) and transitive (as homomorphisms are closed under composition), so \sqsubseteq is a preorder. A bijection f is an *lts isomorphism* if both f and f^{-1} are lts homomorphisms. In this case we call A and B *isomorphic*.

Lemma 1. *Let $A = (S_A, T, \delta_A, s_{0,A})$ and $B = (S_B, T, \delta_B, s_{0,B})$ be lts so that $A \sqsubseteq B$ via f and $A \sqsubseteq B$ via f' . If A is reachable and B is deterministic, then $f = f'$.*

Proof. We proof by induction on the length of words $w \in L(A)$ that if $s_{0,A}[w]s$, then $f(s) = f'(s)$. Since A is reachable, we reach all states of A in this way, showing that $f(s) = f'(s)$ for all $s \in S$. The induction basis follows from the definition of \sqsubseteq : $f(s_{0,A}) = s_{0,B} = f'(s_{0,A})$.

For the induction step, assume that $s_{0,A}[w]s$ with $f(s) = f'(s)$ and consider any edge $s[a]s'$. Since $A \sqsubseteq B$ via f and f' , we have $f(s)[a]f(s')$ and $f'(s)[a]f'(s')$. Because B is deterministic, there cannot be two different states that are reached from $f(s) = f'(s)$ via label a . We conclude $f(s') = f'(s')$. \square

We already argued that \sqsubseteq is a preorder. The next lemma shows that this is a partial order for reachable and deterministic lts.

Lemma 2. *Let A and B be reachable and deterministic lts so that $A \sqsubseteq B$ via f_A and $B \sqsubseteq A$ via f_B . Then A and B are isomorphic.*

Proof. Consider $A \sqsubseteq A$ (via id) and $A \sqsubseteq B \sqsubseteq A$ via $f_B \circ f_A$. Since Lemma 1 is applicable, we conclude $f_B \circ f_A = \text{id}$, making f_A injective and f_B surjective. With an analogous argument for $f_A \circ f_B = \text{id}$ we see that f_A and f_B are both bijective homomorphisms with $f_A^{-1} = f_B$, i.e. they are isomorphisms. \square

A (finite, initially marked, place-transition, arc-weighted, unlabelled) Petri net is a tuple (P, T, F, M_0) such that P is a finite set of *places*, T is a finite set of *transitions*, with $P \cap T = \emptyset$, F is a *flow function* $F: ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}$, and M_0 is the *initial marking*, where a *marking* is a mapping $M: P \rightarrow \mathbb{N}$, indicating the number of *tokens* in each place. $F(p, t) = w > 0$ (resp. $F(t, p) = w > 0$) means that there is an *arc* from p to t (resp. from t to p) with *arc weight* w . A transition $t \in T$ is *enabled* by a marking M , denoted by $M[t]$, if for all places $p \in P$, $M(p) \geq F(p, t)$. If t is enabled at M , then t can *occur* (or *fire*) in M , leading to the marking M' defined by $\forall p \in P: M'(p) = M(p) - F(p, t) + F(t, p)$ (notation: $M[t]M'$). The set $\mathcal{E}(N)$ of *reachable markings* is recursively defined as the least set containing M_0 so that when $M[t]M'$ with $M \in \mathcal{E}(N)$, then also $M' \in \mathcal{E}(N)$. The *reachability graph RG(N)* of N , with initial marking M_0 , is

the labelled transition system with the set of states $\mathcal{E}(N)$ and the set of edges $\{(M, t, M') \mid M, M' \in \mathcal{E}(N) \wedge M[t]M'\}$. If $\text{RG}(N)$ is finite, N is called *bounded*. If an lts A is isomorphic to the reachability graph of a Petri net N , then we will also say that N *solves* A .

3 Region Theory

According to [2], a *region* of an lts $A = (S, T, \delta, s_0)$ is a triple $r = (r_S, r_B, r_F)$ where $r_S: S \rightarrow \mathbb{N}$ assigns each state s a number of tokens $r_S(s)$ and r_B (resp. r_F) are functions $T \rightarrow \mathbb{N}$ assigning each label a *backward* (*resp. forward*) *weight*. Additionally, for all $(s, t, s') \in \delta$ both $r_S(s) \geq r_B(t)$ and $r_S(s') = r_S(s) - r_B(t) + r_F(t)$ must hold. For any region r we will canonically call its components r_S, r_B , resp. r_F . A set R of regions of the lts $A = (S, T, \delta, s_0)$ corresponds to the Petri net $N(R) = (R, T, F, M_0)$ where for each $r = (r_S, r_B, r_F) \in R$ and $t \in T$ we have $F(r, t) = r_B(t)$, $F(t, r) = r_F(t)$, and $M_0(r) = r_S(s_0)$ by definition. Similarly, for a Petri net $N = (P, T, F, M_0)$, for each place $p \in P$, the *extension* of p is a region $[[p]] = (r_S, r_B, r_F)$ on $\text{RG}(N)$ defined by $r_S(M) = M(p)$, $r_B(t) = F(p, t)$, and $r_F(t) = F(t, p)$.

Thus, a region of A is a potential place of a Petri net N solving A . The definition of a region ensures that any region of A can be added to such a Petri net N without preventing an existing edge of A . The two places of the Petri net N from Fig. 1 correspond to two different regions of the lts A_2 in the same figure. Specifically, when writing a region (r_S, r_B, r_F) as a tuple of vectors $((r_S(q_0), \dots, r_S(q_5)), (r_B(a), r_B(b)), (r_F(a), r_F(b)))$ then p_0 corresponds to the region $((1, 1, 1, 0, 0, 0), (1, 1), (0, 1))$ and p_1 corresponds to the region $((2, 1, 0, 2, 1, 0), (0, 1), (0, 0))$.

This idea of potential places is formalised in the next lemma.

Lemma 3. *Let R be a set of regions of $A = (S, T, \delta, s_0)$. Then $A \sqsubseteq \text{RG}(N(R))$.*

Proof. We define a homomorphism f via $f(s)(r) = r_S(s)$ for each region $r \in R$. This means that the state s of A is mapped to the marking M of $N(R)$ where each place r contains $r_S(s)$ tokens. We have $f(s_0) = M_0$ by definition of M_0 .

For each edge $(s, t, s') \in \delta$ we have for all $r = (r_S, r_B, r_F) \in R$ that $r_S(s)(r) \geq r_B(t)$, which means that no place prevents t in $f(s)$ and we have $f(s)[t]$. Let M be the marking defined by $f(s)[t]M$. For each place $r = (r_S, r_B, r_F) \in R$ of $N(R)$ we have $M(r) = f(s)(r) - F(p, t) + F(t, p) = r_S(s) - r_B(t) + r_F(t) = r_S(s')$ by the Petri net firing rule and by definition of a region, which means that $M = f(s')$ and concludes the proof that f is an lts homomorphism. \square

Next we show that $r_S(s_0)$ is enough to fully determine r_S in a region.

Lemma 4. *Let r and r' be two regions of a reachable lts A . If $r_B = r'_B$, $r_F = r'_F$ and $r_S(s_0) = r'_S(s_0)$, where s_0 is the initial state of A , then $r = r'$.*

Proof. We will show by induction on the length of w that $s_0[w]s$ implies $r_S(s) = r'_S(s)$. The base case with $w = \varepsilon$ and $s = s_0$ holds by assumption.

Assuming $s_0[w']s'[t]s$ with $t \in T$ and $r_S(s') = r'_S(s')$, we want to show that $r_S(s) = r'_S(s)$. This holds by $r_S(s) = r_S(s') - r_B(t) + r_F(t) = r'_S(s)$. \square

Remember that the goal of Petri net synthesis is to find a Petri net N solving a given lts A . So far, we defined regions which describe potential places of N . However, we do not know which regions are needed to construct N .

For an lts $A = (S, \rightarrow, T, s_0)$, a *state separation problem (SSP)* is a set $\{s, s'\} \subseteq S$ with $s \neq s'$. An *event/state separation problem (ESSP)* is a tuple $(s, t) \subseteq S \times T$ so that $\neg s[t]$. A region $r = (r_S, r_B, r_F)$ solves a state separation problem $\{s, s'\}$ if $r_S(s) \neq r_S(s')$, i.e. it will make s and s' correspond to different markings in $N(\{r\})$. It solves an event/state separation problem (s, t) if $r_S(s) < r_B(t)$, i.e. it will disable transition t in the marking corresponding to s in $N(\{r\})$.

Each separation problem needs a solution to solve an lts:

Proposition 5 [2]. *For a set of regions R , $N(R)$ solves A if, and only if, each SSP and ESSP instance of A is solved by a region in R and A is reachable.*

The following two properties of regions will be useful later.

Lemma 6. *Let A and B be finite lts with $A \sqsubseteq B$ via f . If $r = (r_S, r_B, r_F)$ is a region of B , then $r \circ f := (r_S \circ f, r_B, r_F)$ is a region of A .*

Proof. For an edge $s[t]s'$ of A , $f(s)[t]f(s')$ is an edge of B . Since r is a region of B , we now have $r_S(f(s)) \geq r_B(t)$. By the same argument we have $r_S(f(s')) = r_S(f(s)) - r_B(t) + r_F(t)$. Thus, $(r_S \circ f, r_B, r_F)$ is a region of A . \square

Lemma 7. *For a region $r = (r_S, r_B, r_F)$ of a finite lts $A = (S, T, \delta, s_0)$, there is a complementary region $\bar{r} = (\bar{r}_S, \bar{r}_B, \bar{r}_F)$ and a number $k \in \mathbb{N}$ with $\bar{r}_B = r_F$ and $\bar{r}_F = r_B$, so that for all $s \in S$ we have $\bar{r}_S(s) = k - r_S(s)$.*

Proof. Most of the region property for \bar{r} is inherited from r . We just need to find a suitable k so that \bar{r}_S does not produce negative numbers and for each edge $(s, t, s') \in \delta$ we have $r_S(s) \geq r_B(t)$. Since A is finite, we can choose k to be the minimum value so that \bar{r} is a region. \square

4 Computing Minimal Over-Approximations

We want to show that any finite lts has a minimal Petri net over-approximation, and want to calculate this over-approximation. We begin with the existence:

Theorem 8. *For a finite lts A there is a set of regions R so that $N(R)$ is bounded and for all sets of regions \hat{R} also $\text{RG}(N(R)) \sqsubseteq \text{RG}(N(\hat{R}))$ holds.*

Proof. Let R' be the set of all regions of A . This might be an infinite set, so $N(R')$ might not be a (finite) Petri net, but for the moment we lift the restriction that a Petri net has only finitely many places. $\text{RG}(N(R'))$ is finite, because by Lemma 7, for each region r , there is a complement region \bar{r} so that $r_S(s) + \bar{r}_S(s)$ is constant. The same also holds for the corresponding places: The sum of their numbers of tokens stays constant when firing transitions. Thus, for each place of $N(R')$, there are only finitely many reachable numbers of tokens, which means that $N(R')$ is bounded and $\text{RG}(N(R'))$ is finite.

$\text{RG}(N(R'))$ is solvable by a finite Petri net: Because the lts is finite, it has only finitely many separation problems. Each separation problem is solvable by construction: If two markings are different, then some region/place differs in its number of tokens and thus the corresponding SSP instance is solvable. If a transition is disabled in a given marking, there is some region/place that prevents the transition from firing and which thus solves the corresponding ESSP instance. Since the number of separation problems of a finite lts is finite, there is a finite set of regions R so that $N(R)$ solves $\text{RG}(N(R'))$ by Proposition 5.

Finally, for any set of regions \widehat{R} of A also $\text{RG}(N(R)) \sqsubseteq \text{RG}(N(\widehat{R}))$ holds: Since R' contains all regions, we have $\widehat{R} \subseteq R'$. Define f to be the function that restricts markings of $N(R')$ to those regions/places present in $N(\widehat{R})$. f is an lts homomorphism showing $\text{RG}(N(R')) \sqsubseteq \text{RG}(N(\widehat{R}))$. Since $\text{RG}(N(R'))$ and $\text{RG}(N(R))$ are isomorphic, it follows that $\text{RG}(N(R)) \sqsubseteq \text{RG}(N(\widehat{R}))$. \square

To compute this minimal over-approximation, we modify the lts according to its unsolvable separation problems. If a state separation problem $\{s, s'\}$ is unsolvable, then the states s and s' must be the same in a Petri net solution. Thus, we identify these two states. If an event/state separation problem (s, t) is unsolvable, this means that in state s transition t cannot be prevented. Thus, we add an edge with label t going from s to a new state.

To formalise this idea, let $A = (S, T, \delta, s_0)$ be an lts and define $\text{SSP}_{\text{unsol}}(A)$ and $\text{ESSP}_{\text{unsol}}(A)$ to be its set of unsolvable SSP instances, respectively unsolvable ESSP instances. We define $s \equiv_A s'$ if, and only if, $s = s'$ or $\{s, s'\} \in \text{SSP}_{\text{unsol}}(A)$. This is an equivalence relation.

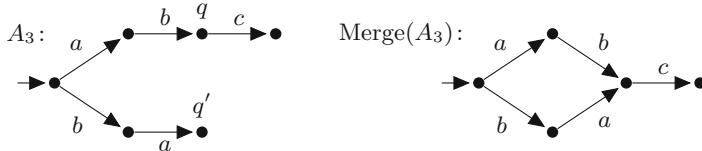


Fig. 2. On the left: An lts A_3 with $\text{SSP}_{\text{unsol}}(A_3) = \{\{q, q'\}\}$ and $\text{ESSP}_{\text{unsol}}(A_3) = \{(q', c)\}$. On the right: The lts $\text{Merge}(A_3)$ where states q and q' are identified.

This equivalence relation is used to define the state-merged lts $\text{Merge}(A) = (S/\equiv_A, T, \delta/\equiv_A, [s_0])$ as follows: A state s is replaced with its equivalence class $[s] = \{s' \mid s \equiv_A s'\}$ and an edge $(s, t, s') \in \delta$ is replaced with $([s], t, [s']) \in \delta/\equiv_A$. An example for this construction is shown in Fig. 2.

The lts $\text{Merge}(A)$ handles unsolvable state separation problems. Next, we handle unsolvable event/state separation problems. For this, let $\text{Merge}(A) = (S', T, \delta', [s_0])$ and define the expansion $\text{Expand}(A)$ of A to be the lts $\text{Expand}(A) = (S' \cup S'', T, \delta' \cup \delta'', [s_0])$, i.e. some states and edges are added to $\text{Merge}(A)$. Let the set of remaining¹ ESSP instances be $\Lambda = \{([s], t) \in (S/\equiv_A) \times T \mid (s, t) \in$

¹ For example, in Fig. 2 the ESSP instance (q', c) no longer applies to $\text{Merge}(A)$.

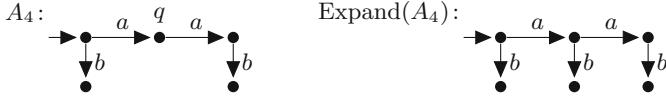


Fig. 3. On the left: The lts A_4 with $\text{SSP}_{\text{unsol}}(A_4) = \emptyset$ and $\text{ESSP}_{\text{unsol}}(A_4) = \{(q, b)\}$. Because there are no unsolvable state separation problems, A_4 and $\text{Merge}(A_4)$ are isomorphic. On the right: The lts $\text{Expand}(A_4)$ where a new edge with $[q][b]$ was added.

$\text{ESSP}_{\text{unsol}}(A) \wedge \forall s' \in [s]: \neg s'[t]\}$. For each missing edge we add a new target state and an edge, $S'' = \{s_{([s], t)} \mid ([s], t) \in A\}$ and $\delta'' = \{([s], t, s_{([s], t)}) \mid ([s], t) \in A\}$, where we assume w.l.o.g. that $S' \cap S'' = \emptyset$. Figure 3 shows an example. A variation of this example can be found in Fig. 1 where $\text{Expand}(A_1) = A_2$.

In the remainder of this section, we want to show that repeated application of the Expand -function calculates the minimal over-approximation of a given lts.

Lemma 9. *Let $A = (S, T, \delta, s_0)$ be an lts, then $A \sqsubseteq \text{Expand}(A)$.*

Proof. Let $\text{Expand}(A) = (S', T, \delta', [s_0])$. The canonical homomorphism $f: S \rightarrow S'$ defined via $f(s) = [s]$ is an lts homomorphism: By definition we have $f(s_0) = [s_0]$ and for each edge $(s, t, s') \in \delta$, we have $([s], t, [s']) \in \delta'$ by definition of δ' . \square

Lemma 10. *For a reachable lts $A = (S, T, \delta, s_0)$, the expansion $\text{Expand}(A)$ is deterministic and reachable.*

Proof. First, we show that $\text{Merge}(A)$ is deterministic and reachable. Reachability is easily inherited from A : Any path $s_0[w]s$ in A can inductively be translated onto $[s_0][w][s]$ in $\text{Merge}(A)$.

For determinism, assume that $[s][a][s']$ and $[s][a][s'']$ in $\text{Merge}(A)$. We want to show that $[s'] = [s'']$, i.e. $s' \equiv_A s''$. Since we have the edges $[s][a][s']$ and $[s][a][s'']$ in $\text{Merge}(A)$, there must be edges $s[a]s'$ and $\widehat{s}[a]s''$ in A for some state \widehat{s} with $s \equiv_A \widehat{s}$. Let $r = (r_S, r_B, r_F)$ be an arbitrary region. By $s \equiv_A \widehat{s}$ we have $r_S(s) = r_S(\widehat{s})$. Thus, by the definition of a region it follows that $r_S(s') = r_S(s) - r_B(a) + r_F(a) = r_S(\widehat{s}) - r_B(a) + r_F(a) = r_S(s'')$. This means that these two states cannot be separated and we have $s' \equiv_A s''$.

For $\text{Expand}(A)$, reachability is obviously inherited from $\text{Merge}(A)$ since every new state is reachable from an already-reachable state of $\text{Merge}(A)$. Also, the edges that are added to $\text{Merge}(A)$ to construct $\text{Expand}(A)$ are constructed such that no non-determinism is introduced. \square

Lemma 11. *Let $A = (S, T, \delta, s_0)$ be an lts and $N = (P, T, F, M_0)$ a Petri net. If $A \sqsubseteq \text{RG}(N)$, then also $\text{Expand}(A) \sqsubseteq \text{RG}(N)$.*

Proof. Let f be the homomorphism witnessing $A \sqsubseteq \text{RG}(N)$. We want to define a homomorphism f' witnessing $\text{Expand}(A) \sqsubseteq \text{RG}(N)$. For this, we first show that if $(s, t) \in \text{ESSP}_{\text{unsol}}(A)$, then $f(s)[t]$ in $\text{RG}(N)$, which we do by contraposition.

Assume that there is a place p of N with $f(s)(p) < F(p, t)$, i.e. p disables transition t in the marking $f(s)$. We use the extension $[[p]]$ of p and invoke Lemma 6 to turn this into a region $[[p]] \circ f$ of A . Let $[[p]] \circ f = (r_S, r_B, r_F)$. By assumption this region satisfies $r_S(s) < r_B(t)$, i.e. it solves the ESSP instance (s, t) . Thus, $(s, t) \in \text{ESSP}_{\text{unsol}}(A)$ implies $f(s)[t]$.

We now define the homomorphism f' witnessing $\text{Expand}(A) \sqsubseteq \text{RG}(N)$ via a case analysis on its argument as $f'([s]) = f(s)$ and $f'(s_{([s], t)}) = M_{(s, t)}$, where $M_{(s, t)}$ is defined by $f(s)[t]M_{(s, t)}$. Here, $M_{(s, t)}$ must exist since a state $s_{([s], t)}$ is only created if $(s, t) \in \text{ESSP}_{\text{unsol}}(A)$ and we have just shown that then $f(s)[t]$.

We have to show three things so that f' is a homomorphism: (1) f' is well-defined and does not depend on the choice of s from the equivalence class $[s]$; (2) $f'([s_0]) = M_0$; and (3) $s[t]s'$ in $\text{Expand}(A)$ implies $f'(s)[t]f'(s')$ in $\text{RG}(N)$.

For (1), we use contraposition: We assume two states s and s' with $f(s) \neq f(s')$ and show that $s \not\equiv_A s'$. Since $f(s) \neq f(s')$, there is a place p of N with $f(s)(p) \neq f(s')(p)$. We can now translate its extension $[[p]]$ into a region $[[p]] \circ f$ of A that separates the states s and s' . This shows that $s \not\equiv_A s'$.

For (2), observe that by $f(s_0) = M_0$ also $f'([s_0]) = M_0$.

For (3), there are again two cases. Assuming an edge $([s], t, s_{(s, t)})$, we have shown above that a marking $M_{(s, t)}$ exists with $f(s)[t]M_{(s, t)}$. By $f'([s]) = f(s)$ and $f'(s_{([s], t)}) = M_{(s, t)}$ we obtain an edge $(f'([s]), t, f'(s_{([s], t)}))$ in $\text{RG}(N)$. The other possible case is an edge $([s], t, [s'])$. Here we have directly $f'([s]) = f(s)$ and $f'([s']) = f(s')$, (s, t, s') is an edge in A and since f is an homomorphism, there is an edge $(f(s), t, f(s')) = (f'([s]), t, f'([s']))$ in $\text{RG}(N)$. \square

Lemma 6 allows us to turn regions of B into regions of A if $A \sqsubseteq B$ holds, i.e. ‘transplant’ them to a smaller lts. The opposite direction, ‘transplanting’ a region to a larger lts, is not possible in general. For example, in Fig. 1, place p_0 of the Petri net N limits transition a to fire only once. Thus, the corresponding region of A_2 is not valid for an lts B with $A_2 \sqsubseteq B$ where a can occur twice.

The next lemma shows that ‘transplanting’ a region to a larger lts is possible in the special case of $B = \text{Expand}(A)$.

Lemma 12. *Given a reachable lts A and one of its regions r , there is a region r' of $\text{Expand}(A)$ so that $r = r' \circ f$ where f is the canonical homomorphism $f(s) = [s]$.*

Proof. By Lemma 4, a region (r_S, r_B, r_F) of A is fully determined by $r_S(s_0)$, r_B and r_F . Thus, we can define the region $r' = (r'_S, r'_B, r'_F)$ of $\text{Expand}(A)$ based on $r = (r_S, r_B, r_F)$ by $r'_S([s_0]) = r_S(s_0)$, $r'_B = r_B$ and $r'_F = r_F$. We have to show that this is indeed a region, i.e. for all edges $s[t]s'$ in $\text{Expand}(A)$ both (1) $r'_S(s) \geq r'_B(t)$ and (2) $r'_S(s') = r'_S(s) - r'_B(t) + r'_F(t)$ hold.

Let $s[t]s'$ be an edge of $\text{Expand}(A)$. By definition of $\text{Expand}(A)$, there is a state s'' of A so that $s = [s'']$. There are now two possibilities: Either there is a state s''' of A with $s' = [s''']$, or $s' = s_{(s, t)}$, i.e. s' already exists in $\text{Merge}(A)$ or was added in $\text{Expand}(A)$.

In the first case we have $r_S(s'') \geq r_B(t)$ in A and by $r'_S(s) = r_S(s'')$ and $r'_B(t) = r_B(t)$ thus also $r'_S(s) \geq r'_B(t)$. Condition (2) follows similarly.

In the second case, $s[t]s'$ with $s = [s'']$ and $s' = s_{(\hat{s}, t)}$, (\hat{s}, t) is an unsolvable ESSP instance of A . This means by definition that no region of A satisfies $r_S(\hat{s}) < r_B(t)$, from which condition (1) follows. In this case, the state s' of $\text{Expand}(A)$ is not in the image of f where f witnesses $A \sqsubseteq \text{Expand}(A)$. Thus, the number of tokens on s' is defined purely by (2) and this equation holds automatically. \square

These lemmas can now be used to compute the minimal over-approximation of an lts A via iterated application of the Expand -function.

Theorem 13. *Given a finite and reachable lts A , the chain defined by $A_0 = A$ and $A_{i+1} = \text{Expand}(A_i)$ reaches a fixed-point A^* (up to isomorphism), there is a Petri net N solving A^* , and N is the least Petri net over-approximation of A , i.e. for all Petri nets N' with $A \sqsubseteq \text{RG}(N')$, also $A^* \sqsubseteq \text{RG}(N')$.*

Proof. First we show that the fixed point always exists and then that it is the minimal Petri net over-approximation.

To show that the fixed point exists, consider an arbitrary lts A_i in the chain $(A_i)_{i \in \mathbb{N}}$. We want to show that there are only finitely many possibilities for A_i and thus eventually the chain must reach a fixed-point.

We begin by showing an upper bound on the number of states of A_i . By Theorem 8, there is a minimal Petri net over-approximation $N(R')$ of A where $N(R')$ is bounded. Let $m \in \mathbb{N}$ be the number of reachable markings in $N(R')$. We have $A \sqsubseteq A_i \sqsubseteq \text{RG}(N(R'))$ by iterative application of Lemma 9, respectively Lemma 11. Let $n \in \mathbb{N}$ be the number of states of $\text{Merge}(A_i)$. Since each state-separation problem in $\text{Merge}(A_i)$ and $\text{RG}(N(R'))$ is solvable by definition and since each region of $\text{Merge}(A_i)$ can be transferred to $\text{RG}(N(R'))$ via Lemma 12, we have $n \leq m$, i.e. $\text{Merge}(A_i)$ cannot have more states than $\text{RG}(N(R'))$. This also provides an upper bound on the size of $\text{Expand}(A_i)$: It has at most $m \cdot (1 + |T|)$ states, since at most one state is added per state and label.

By Lemma 10, each A_i is deterministic and reachable. Thus, by Lemma 2 the preorder \sqsubseteq is in fact a partial order in this setting. If some element appears twice in a partially ordered sequence, it must also appear twice consecutively and thus is a fixed-point of the underlying function. Since there are only finitely many different lts with an upper bound on the number of states and a fixed alphabet, at least one lts A' must appear infinitely often in the chain $(A_i)_{i \in \mathbb{N}}$. Thus $A' = A^*$ is a fixed-point of the Expand -function.

Next we want to show that A^* can be solved by a Petri net and that it is the least over-approximation of A . Since $\text{Expand}(A^*) = A^*$ holds, we have $\text{SSP}_{\text{unsol}}(A) = \emptyset = \text{ESSP}_{\text{unsol}}(A)$ by definition of the Expand -function. By Proposition 5, A^* can thus be solved by a Petri net $N(R)$ for a suitable set R of regions that solve all separation problems. By iterated application of Lemma 11, we have that for all Petri nets N' with $A \sqsubseteq \text{RG}(N')$ also $A^* \sqsubseteq \text{RG}(N')$. \square

5 Subclasses of Nets

In this section we will show that the algorithm from the previous section also works on some subclasses of nets, where a subclass of Petri nets is a restriction on the Petri nets to consider.

We begin by introducing some well-known subclasses, for which the following definitions are needed: For a place p of a Petri net $N = (P, T, F, M_0)$, let $\bullet p = \{t \in T \mid F(t, p) > 0\}$ be its *preset*, and $p^\bullet = \{t \in T \mid F(p, t) > 0\}$ its *postset*. A Petri net is *plain* if no weighted arcs exist, i.e. each place $p \in P$ of $N = (P, T, F, M_0)$ satisfies $F(t, p) \leq 1 \geq F(p, t)$ for all $t \in T$. A Petri net is *pure* if there are no side-conditions, i.e. for each $p \in P$ and $t \in T$ we have $F(t, p) = 0 \vee F(p, t) = 0$. A *T-net* is a Petri net where each place has at most one transition in its preset and postset, i.e. $\forall p \in P : |\bullet p| \leq 1 \geq |p^\bullet|$. A *marked graph* is a Petri net where each place has exactly one transition its preset and postset, i.e. $\forall p \in P : |\bullet p| = 1 = |p^\bullet|$. A Petri net is *k-bounded* if no place has ever more than $k \in \mathbb{N}$ tokens, i.e. each reachable marking $M \in \mathcal{E}(N)$ satisfies $\forall p \in P : M(p) \leq k$. These subclasses can also be combined, e.g. the subclass of plain and pure Petri nets is the intersection of the sets of all plain and all pure nets.

The goal of this section is to show that the algorithm that was introduced in the previous section also works for these subclasses. For this, we recapitulate which lemmas were used as the basis of the proof. For any subclass where these lemmas hold, Theorem 13 is true, i.e. iteratively applying the Expand-function results in a minimal Petri net over-approximation of an lts.

- If r is a region of B and $A \sqsubseteq B$ via f , then $r \circ f$ is a region of A (Lemma 6).
- For each region r there is a complement region \bar{r} with $\bar{r}_B = r_F$ and $\bar{r}_F = r_B$ (Lemma 7).
- A region r of A can be translated into a region r' of $\text{Expand}(A)$ (Lemma 12).

Except for *k*-boundedness, the properties that were previously defined only consider the arcs between transitions and places and not the number of tokens on a place. For a region (r_S, r_B, r_F) this means that only the backward weight r_B and the forward weight r_F are needed to decide membership of a subclass. Since the constructions of Lemmas 6 and 12 do not change these functions, membership of a subclass is preserved by the constructions from these lemmas. Also, these subclasses are symmetric in the sense that the same restrictions are placed on r_B and r_F . Thus, Lemma 7, which swaps these two functions, also preserves membership of these subclasses.

This argument also applies to combinations of these properties.

For *k*-boundedness, a different argumentation is needed. When two reachable and deterministic lts satisfy $A \sqsubseteq B$, then any *k*-bounded region r of B becomes a *k*-bounded region $r \circ f$ of A since $(r \circ f)_S$, which is the function that assigns tokens to states, has the same codomain as r_S (Lemma 6 holds). Complement regions (Lemma 7) become easier, since we can directly define $\bar{r}_S(s) = k - r_S(s)$ and do not have to compute a suitable k as in Lemma 7. This construction also helps in translating regions r of A into regions r' of $\text{Expand}(A)$ (Lemma 12): If a new edge $s[a]s'$ appears in $\text{Expand}(A)$, this is due to an unsolvable ESSP instance. If the number of tokens assigned to s' by r' would exceed the value k , then the complement region \bar{r} of r would solve the ESSP instance (s, t) which was assumed to be unsolvable. Thus, the existing construction to produce r' preserves *k*-boundedness.

We can also combine k -boundedness with the other subclasses mentioned before since the individual argumentations do not interfere with each other: k -boundedness only considers r_S while the arguments for the remaining subclasses only talk about r_B and r_F . This shows that all constructions of Sect. 4 preserve classes and thus all the results also hold for all the subclasses defined above.

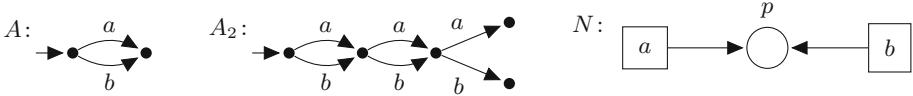


Fig. 4. An lts A , an intermediate result $A_2 = \text{Expand}(\text{Expand}(A))$ in the computation of A^* , and the minimal Petri net over-approximation N of A .

An example for a subclass of nets in which the presented construction does not work are *place-output-nonbranching* Petri nets. A Petri net N is *place-output-nonbranching* if for each place $p \in P$ its postset has at most size one: $|p^\bullet| \leq 1$. This definition is not symmetric in the sense that no restriction is applied to the preset of p . Thus, no complementary regions \bar{r} in the sense of Lemma 7 can be constructed. This is illustrated by the place p of the Petri net N in Fig. 4, whose extension $[[p]]$ cannot be complemented. Also, Theorem 8, which showed that the minimal Petri net over-approximation of a finite lts is a bounded Petri net, does not hold for this subclass. This is shown by the lts A in Fig. 4. Its minimal Petri net over-approximation N is an unbounded Petri net. Thus, the chain $A_{i+1} = \text{Expand}(A_i)$ does not reach a fixed-point.

6 Conclusion

In this paper we introduced a fixed-point algorithm to compute minimal Petri net over-approximations of given labelled transition systems. Minimality is here understood according to an lts homomorphism preorder, which is an edge-preserving function between states of different lts. This algorithm is also shown to work for plain, pure, k -bounded Petri nets, T-nets, and marked graphs, and combinations like plain and pure Petri nets. Existing algorithms solve separation problems, and fail if one of them is unsolvable. The presented algorithm uses the set of unsolvable separation problems to modify the lts suitably for the next iteration. The algorithm was implemented in the APT toolbox².

An open question regarding this algorithm is its complexity. The approach from [2] for a similar problem, but only considering general and pure Petri nets, is based on so-called *extremal regions* and may need exponential time, since the number of extremal regions can be exponential in the size of an lts [2]. While our approach can be used with polynomial algorithms for Petri net synthesis, e.g. [1], this does not entail that our approach is polynomial, because so far no upper bound on the number of iterations is known.

² Available in the `overapproximate_synthetize`-module at <https://github.com/CvO-Theory/apt>.

However, we conjecture that a polynomial number of iterations suffices and that the size of the minimal over-approximation is polynomial in the size of the input. For an acyclic lts over a fixed set of labels T , let n be the length of its longest path. Each event can be limited to occur at most n times, so that at most $(n + 1)^{|T|}$ markings are reachable in the minimal over-approximation, which is computed in at most n iterations. Future work will have to refine this argument and extend it to lts containing cycles.

Acknowledgements. The author would like to thank the anonymous reviewers, Eike Best, and Harro Wimmel for their very useful comments.

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Nielsen, M., Schwartzbach, M.I. (eds.) CAAP 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59293-8_207
2. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. TTCS. Springer, Heidelberg (2015). <https://doi.org/10.1007/978-3-662-47967-4>
3. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_22
4. Best, E., Devillers, R.: Characterisation of the state spaces of live and bounded marked graph Petri nets. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 161–172. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04921-2_13
5. Best, E., Devillers, R.: State space axioms for T-systems. Acta Inf. **52**(2–3), 133–152 (2015). <https://doi.org/10.1007/s00236-015-0219-0>
6. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering Petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_26
7. Darondeau, P.: Deriving unbounded Petri nets from formal languages. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 533–548. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0055646>
8. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Part I: basic notions and the representation problem and Part II: state spaces of concurrent systems. Acta Inf. **27**(4), 315–368 (1990). <https://doi.org/10.1007/BF00264611>
9. Lorenz, R., Mauser, S., Juhás, G.: How to synthesize nets from languages: a survey. In: WSC, pp. 637–647 (2007). <https://doi.org/10.1109/WSC.2007.4419657>
10. Schlachter, U.: Petri net synthesis for restricted classes of nets. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 79–97. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_6
11. Schlachter, U., Wimmel, H.: k-bounded Petri net synthesis from MTS. In: Meyer, R., Nestmann, U. (eds.) CONCUR 2017. LIPIcs, vol. 85, pp. 6:1–6:15. Schloss Dagstuhl (2017). <https://doi.org/10.4230/LIPIcs.CONCUR.2017.6>
12. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 368–387. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68746-7_24



Efficient Translation with Linear Bimorphisms

Christoph Teichmann^{ID}, Antoine Venant^(✉), and Alexander Koller^{ID}

Department of Language Science and Technology, Saarland University,
Campus C7.2, Saarbrücken, Germany
{cteichmann,venant,koller}@coli.uni-saarland.de

Abstract. We show that the image of a regular tree language under a linear bimorphism over binary signatures can be computed in linear time in the size of the input automaton. We do this by transformation into a novel normal form. Our result applies to the translation and parsing complexity of a wide range of grammar formalisms used in computational linguistics, which can now be shown in a uniform way.

Keywords: Parsing · Tree languages and tree automata
Bimorphisms

1 Introduction

Languages defined as the image of a regular tree language (RTL) under one or several tree-homomorphisms play an important role in formal language theory and computational linguistics. For instance, the translation performed by any bottom-up finite tree transducer (FTT) can be represented as a bimorphism [1, 13], i.e., an RTL and two tree homomorphisms. By interpreting the symbols in the homomorphic images of a bimorphism as operations over some algebra [5], one obtains Interpreted Regular Tree Grammars (IRTGs) [9], which can be used to define languages of and relations between objects from arbitrary algebras. IRTGs have been used to capture a wide variety of expressive grammar formalisms for strings, graphs [7], and other objects.

This paper presents a generic method for determining the asymptotic complexity of parsing and translating with IRTGs. IRTG parsers compute the translation of an RTL – represented as a finite tree automaton (FTA) or regular tree grammar – under a bimorphism. The size of the regular tree automaton to be translated depends not only on the input, but also on the specific algebra used and is therefore hard to analyze generically. But we also have to answer the question when and how the translation problem can be solved efficiently. Computing the translation requires the computation of an inverse homomorphism. While it is

We thank Frank Drewes, Meaghan Fowlie, Jonas Groschwitz, Andreas Maletti, and Heiko Vogler for discussions about the paper and the anonymous reviewers for their feedback. This work was supported by DFG grant KO 2916/2-1.

known that RTLs are closed under inverse homomorphism, an FTA representing an inverse image might be much larger than the input FTA if the homomorphism is complex. This issue is especially pressing in computational linguistics, where certain applications require computing the inverse homomorphism of very large FTAs with hundreds of millions of rules [7]. For specific classes such as linear symbol-to-symbol homomorphisms, it is known that inverse homomorphisms can be computed in linear time and space [3]. However, IRTGs used in computational linguistics typically use homomorphisms which are linear but not symbol-to-symbol [6, 11], because they express the complex way in which a word combines with its arguments in terms of primitive operations of the algebra. It is therefore an open question whether the bimorphisms used in computational linguistics allow for efficient computation of inverse images.

In this paper, we show that for any bimorphism using linear homomorphisms over binary signatures, the most relevant subclass of bimorphisms for computational linguistics, we can construct an equivalent bimorphism in which each homomorphic image contains at most one symbol (plus variables). We call this the Truncated Normal Form (TNF) of the bimorphism. We show that inverse homomorphisms can be calculated in linear time for bimorphisms in TNF. Thus, we obtain a general result on the translation complexity of bimorphisms. Because IRTGs build on bimorphisms, we can show that, as long as only linear homomorphisms over binary signatures are used, the computational complexity of IRTGs can be analyzed purely in terms of the input automata and the homomorphisms add no further asymptotic complexity. This yields a uniform proof of translation and parsing complexity for grammar formalisms expressible as IRTGs, including context-free, tree-adjoining [8] and hyperedge replacement grammars [10].

2 Background

2.1 Notations

Let \mathbb{N} denote the set of natural numbers. For $i, j \in \mathbb{N}$, $[i, j]$ denotes the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$. A signature (or ranked alphabet) is a pair $\Sigma = \langle X, \text{ar} \rangle$ where X is a countable set of symbols and $\text{ar} : X \mapsto \mathbb{N}$ assigns each symbol a rank. $l \in \Sigma$ has the meaning of $l \in X$. We say a signature is binary if $\max_{l \in \Sigma} \text{ar}(l) \leq 2$. We assume the reader is familiar with the concepts of terms and regular tree languages. Refer to [4] for a formal presentation. We use the words ‘terms’ and ‘trees’ interchangeably. We let $\mathcal{T}(\Sigma, \mathcal{V})$ denote the set of terms over some signature Σ and a set of variable symbols \mathcal{V} s.t. $\Sigma \cap \mathcal{V} = \emptyset$. For $x \in \mathcal{V}$, $t, t' \in \mathcal{T}(\Sigma, \mathcal{V})$, $t[t'/x]$ denotes the term obtained by replacing all occurrences of x in t with t' . We shorten $\mathcal{T}(\Sigma, \emptyset)$ to $\mathcal{T}(\Sigma)$ and call a member of such a set a *ground term*. We assume (\mathcal{V}, \prec) to be a fixed countably infinite linear order $x_1 \prec x_2 \cdots \prec x_n \prec \dots$ and write \mathcal{V}_n to denote $\{x_1, \dots, x_n\}$ the n first elements of \mathcal{V} .

A (top down) FTA is a tuple $A = \langle \Sigma, Q, R, S \rangle$ with Σ the FTA’s signature, Q the finite set of *states*, R a finite set of *rules* and $S \in Q$ the single *start state*. A rule in R is of the form $\mathbf{r} = X \rightarrow r(Y_1, \dots, Y_{\text{ar}(r)})$, meaning

that it allows rewriting the left hand side $lhs(\mathbf{r}) = X$ into the right hand side $rhs(\mathbf{r}) = r(Y_1, \dots, Y_{\text{ar}(r)})$. r is the *label* of \mathbf{r} . We let $\Sigma \cup N$ denote the signature obtained by augmenting Σ with all the symbols of N and assigning these rank 0. For two ground terms $t, t' \in T(\Sigma \cup N)$, we let $t \rightarrow_{\mathbf{r}} t'$ hold whenever replacing one instance of $lhs(\mathbf{r})$ in t with $rhs(\mathbf{r})$ produces t' . $t \rightarrow_{(r_1; \dots; r_n)} t'$ means $t (\rightarrow_{r_1} \circ \dots \circ \rightarrow_{r_n}) t'$. We let $t \rightarrow_A t'$ hold iff $\exists \mathbf{r} \in R$ such that $t \rightarrow_{\mathbf{r}} t'$. \rightarrow_A^* denotes the reflexive transitive closure of \rightarrow_A . The language of A is $L(A) = \{t \in T(\Sigma) \mid S \rightarrow_A^* t\}$. The size of an FTA $|A|$ is $|R|$, the cardinal of its set of rules.

An algebra is a tuple $\mathcal{A} = \langle \Sigma_{\mathcal{A}}, D_{\mathcal{A}}, \llbracket \cdot \rrbracket_{\mathcal{A}} \rangle$ with $\Sigma_{\mathcal{A}}$ the algebra's signature, $D_{\mathcal{A}}$ its domain and $\llbracket \cdot \rrbracket_{\mathcal{A}}$ the evaluation mapping assigning each symbol σ of rank n a function from $D_{\mathcal{A}}^n$ into $D_{\mathcal{A}}$. We naturally extend $\llbracket \cdot \rrbracket_{\mathcal{A}}$'s domain to inductively evaluate whole terms over $\Sigma_{\mathcal{A}}$ into $D_{\mathcal{A}}$.

2.2 Bimorphisms

Bimorphisms combine tree homomorphisms and FTAs in order to define a language of tuples of trees. Let us first state the definition of a tree homomorphism:

Definition 1 (Tree-homomorphism). *Let Σ and Λ be two signatures. A tree-homomorphism (henceforth, homomorphism) h is a mapping $\Sigma \mapsto T(\Lambda, V)$ such that a symbol $l \in \Sigma$ of rank n is mapped to a term in $T(\Lambda, V_n)$. A homomorphism h induces a mapping $T(\Sigma) \mapsto T(\Delta)$ recursively defined by*

$$h(l(t_1, \dots, t_n)) = h(l)[h(t_1)/x_1][h(t_2)/x_2] \dots [h(t_n)/x_n].$$

We focus on linear homomorphisms over binary signatures, which we call LB homomorphisms. A homomorphism is linear iff the image of any symbol l is a term with at most one occurrence of each variable symbol in $V_{\text{ar}(l)}$. $h : \Sigma \mapsto T(\Lambda, V)$ is a homomorphism over binary signatures if both Σ and Λ are binary.

LB homomorphisms are used in grammars and transducers in computational linguistics because non-binary signatures lead to more complex parsing problems. Note that if the domain of h , Σ , is a binary signature, then only the variables x_1 and x_2 will ever be in the image of any $l \in \Sigma$.

Definition 2 (Bimorphisms).

$H = \langle A, h_1, \dots, h_n \rangle$ is a generalized bimorphism if A is a finite tree automaton and h_1, \dots, h_n are homomorphisms from $T(\Sigma)$ into $T(\Lambda_1), \dots, T(\Lambda_n)$. H defines a language $L(H)$ as:

$$\{\langle t_1, \dots, t_n \rangle \in T(\Lambda_1) \times \dots \times T(\Lambda_n) \mid \exists t \in L(A) : t_1 = h_1(t), \dots, t_n = h_n(t)\}$$

We will call $t \in L(A)$ a derivation tree of H . Traditionally ‘bimorphism’ refers to the case $n = 2$, but we shorten ‘generalized bimorphism’ to just bimorphism. A bimorphism $H = \langle A, h_1, \dots, h_n \rangle$, where each h_1, \dots, h_n is an LB homomorphism, is called an LB bimorphism.

The extension of bimorphisms with algebras, called Interpreted Regular Tree Grammars (IRTGs) [9], defines languages of tuples of objects other than trees.

Definition 3 (IRTG). An IRTG is a tuple $\mathbb{G} = \langle H, \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ where $H = \langle A, h_1, \dots, h_n \rangle$ is a bimorphism and the \mathcal{A}_i are algebras with domains D_1, \dots, D_n . \mathbb{G} defines a language $L(\mathbb{G})$ as:

$$\{\langle [\![t_1]\!]_{\mathcal{A}_1}, \dots, [\![t_n]\!]_{\mathcal{A}_n} \rangle \in D_1 \times \dots \times D_n \mid \langle t_1, \dots, t_n \rangle \in L(H)\}$$

We call two bimorphisms or IRTGs which define the same language *equivalent*.

Figure 1(a) provides an example bimorphism, which can be extended to an IRTG by making \mathcal{A}_1 a string algebra, evaluating leaves to themselves and $*$ nodes as concatenation and by making \mathcal{A}_2 a tree algebra which evaluates every tree to itself (Fig. 2). Rules in Fig. 1(a) have unique labels, which we use to refer to rules.

Rule	h_1	h_2	Rule	\hat{h}_1	\hat{h}_2
$S \rightarrow r_1(B, B)$	$*(x_1, *(c, x_2))$	$I(I(x_2, c), x_1)$	$S \rightarrow r_1^{(1)}(S^{(2)}, S^{(3)})$	$*(x_1, *(c, x_2))$	$I(x_2, x_1)$
$B \rightarrow r_2$	$*(a, b)$	$B(a)$	$S^{(2)} \rightarrow r_1^{(2)}(B)$	x_1	x_1
$B \rightarrow r_3(B, B)$	$*(c, *(x_1, x_2))$	$I(I(x_2, x_1), c)$	$S^{(3)} \rightarrow r_1^{(3)}(B)$	x_1	$I(x_1, c)$
			$B \rightarrow r_2^{(1)}(B^{(2)}, B^{(3)})$	x_1	$B(x_1)$
			$B^{(2)} \rightarrow r_2^{(2)}$	$*(a, b)$	a
			$B^{(3)} \rightarrow r_2^{(3)}$	\top	\top
			$B \rightarrow r_3^{(1)}(B^{(4)}, B^{(5)})$	x_1	$I(x_1, x_2)$
			$B^{(4)} \rightarrow r_3^{(4)}(B, B)$	$*(c, *(x_1, x_2))$	$I(x_2, x_1)$
			$B^{(5)} \rightarrow r_3^{(5)}$	\top	c

Fig. 1. (a) Example bimorphism and (b) truncated bimorphism.

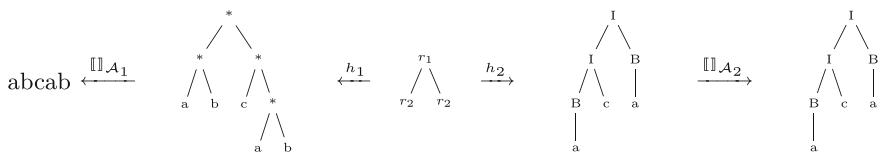


Fig. 2. Derivation tree and mappings.

IRTGs generalize grammar formalisms from computational linguistics such as context-free grammars, tree substitution grammars, tree adjoining grammars,

hyperedge replacement grammars and their synchronous versions. Hence, understanding how efficiently IRTGs and their underlying bimorphisms can be used in parsing and translation problems is very relevant for computational linguistics. We now introduce these problems and sketch our contribution.

2.3 The Inverse Homomorphism Problem

We want to know how efficiently the *translation problem* for an IRTG $\mathbb{G} = \langle H, \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ can be solved: given an input object $o \in D_{\mathcal{A}_i}$ compute $\mathbb{G}(o) = \{\langle o_1, \dots, o_n \rangle \in L(\mathbb{G}) \mid o_i = o\}$. This problem is usually solved by reducing it to the translation problem for the underlying bimorphism $H = \langle A, h_1, \dots, h_n \rangle$ and an FTA F , which consists of computing the image of $L(F)$ through $L(H)$, i.e., the set $H(F) = \{\langle t_1, \dots, t_n \rangle \in L(H) \mid t_i \in L(F)\}$.

Solving the translation problem for a bimorphism $H = \langle A, h_1, \dots, h_n \rangle$ on an input FTA F can be done by computing a *parse chart*, i.e., an FTA $Chart_F$ recognizing the language of derivation trees $t \in L(A)$ such that $h_i(t) \in L(F)$ (then, $\langle t_1, \dots, t_n \rangle \in H(F)$ iff there is some $t \in L(Chart_F)$ such that for each $i \in [1, n]$, $t_i = h_i(t)$). This can be achieved by computing the FTA

$$Chart_F = h_i^{-1}(F) \cap A$$

which requires $h_i^{-1}(F)$, the *invhom automaton* for F . The translation problem for an IRTG $\mathbb{G} = \langle H, \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ on an input o is therefore solvable when o is *regularly decomposable* in \mathcal{A}_i , meaning that we can find an FTA D_o , the *decomposition automaton* for o , recognizing the language $\llbracket o \rrbracket_{\mathcal{A}_i}^{-1} = \{t \in T(\Sigma_{\mathcal{A}_i}) \mid \llbracket t \rrbracket_{\mathcal{A}_i} = o\}$ of terms evaluating to o [9]. Then translation of o through \mathbb{G} reduces to the translation problem for H on D_o . Note that this requires computing the invhom automaton for D_o .

Often we are also interested in the *parsing problem for IRTGs*: with \mathbb{G} , o and H as above, find the set of derivation trees $\{t \in L(A) \mid \llbracket h_i(t) \rrbracket = o\}$. This problem reduces to the the translation problem for the irtg \mathbb{G}' obtained from \mathbb{G} by adding a new identity homomorphism id mapping derivation trees to themselves and interpreting them in the initial algebra of trees.

For many grammar formalisms, it has been shown that parsing/translation for a regularly decomposable input o can done in time $O(|D_o|)$, but there exists no generalisation of this fact to arbitrary formalisms encodable as IRTGs. The problem preventing such a generalization, lies in the fact that arbitrary linear homomorphisms complicate the analysis of the complexity of computing the invhom automaton. The invhom automaton can be quadratic in the size of the input automaton: consider a signature $\Sigma_{\{a\}}$ with $*$ of rank 2 and a as a constant and an FTA F over $\Sigma_{\{a\}}$ including the following rules:

$$\begin{aligned} X_i &\rightarrow * (Y, T), Y \rightarrow * (Z_i, T), Z_i \rightarrow a & (\text{three such rules for each } i \in [1, n]) \\ T &\rightarrow a. \end{aligned}$$

Consider the one symbol signature $\Sigma = \{\langle \sigma, 2 \rangle\}$, and homomorphism h defined by $h(\sigma) = *(*x_1, a), x_2$. The standard construction for $h^{-1}(F)$ will produce a

rule $P \rightarrow r(Q_1, \dots, Q_k)$ whenever $P \rightarrow_F^* h(r)[Q_1/x_1] \dots [Q_k/x_k]$. In our example, we have $X_i \rightarrow_F *Y, T) \rightarrow_F *(*Z_i, T), T) \rightarrow_F *(*Z_i, a), T)$ and therefore add a total of n^2 rules $X_i \rightarrow \sigma(Z_j, T)$, $i, j \in [1, n]$ to $h^{-1}(F)$, when F has $3n + 1$ rules.

Fortunately, the translation problem for a class \mathcal{C} of bimorphisms can be reduced to the translation problem for a smaller class \mathcal{C}' of bimorphisms, if any bimorphism in \mathcal{C} has an equivalent in \mathcal{C}' . This is interesting if bimorphisms in \mathcal{C}' have an “easy” invhom problem, hence translation problem. We will apply this strategy to the class \mathcal{C} of all LB bimorphisms, and take \mathcal{C}' to be the class of LB bimorphisms in a normal form which we define next. The translation problem for bimorphisms in the normal form will be shown to be linear, from which follows that the translation problem for an IRTG based on any LB bimorphism is solvable in time linear in the size of the input’s decomposition automaton.

3 Truncated Normal Form

To better understand bimorphisms and their translation problem, we introduce *Truncated Normal Form* (TNF) which limits the *height* of the homomorphisms used in a bimorphism to 1. We then show that any LB bimorphisms has an equivalent TNF.

The height of $t \in \mathcal{T}(\Sigma, \mathcal{V})$ written $height(t)$ is defined as the maximum number of non-variable nodes on any path from the root to any leaf. For example $height(*x_1, x_2) = 1$ and $height(*x_1, a) = 2$. This is extended to homomorphisms $h : A \rightarrow \mathcal{T}(\Sigma, \mathcal{V})$ by $height(h) = \max_{l \in A} height(h(l))$.

Definition 4 (Truncated Normal Form). A bimorphism $\langle A, h_1, \dots, h_n \rangle$ is in Truncated Normal Form (TNF) iff $\max_{i \in [1, n]} height(h_i) \leq 1$.

Note that TNF differs from requiring that all the homomorphisms be symbol-to-symbol [3], as it allows for height 0 homomorphic images, like $h(l) = x_1$.

The bimorphism in Fig. 1 (a) is not in TNF, since, e.g., r_2 has height 2 for the tree interpretation (h_2). However, r_2 could be replaced with $B \rightarrow r_4(C, D)$, $C \rightarrow r_5$, and $D \rightarrow r_6$ with $h_1(r_4) = *x_1, x_2)$, $h_1(r_5) = a$, $h_1(r_6) = b$, $h_2(r_4) = B(x_1)$, $h_2(r_5) = a$ and $h_2(r_6)$ evaluating to some random constant, e.g., a . These replacement rules would conform to TNF and it is possible to use similar replacements to bring the whole bimorphism into TNF.

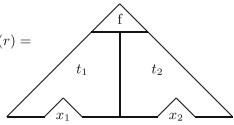
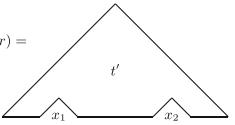
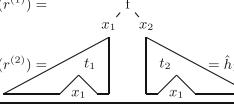
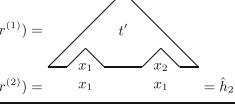
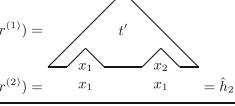
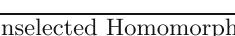
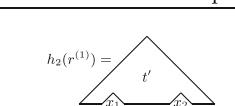
Theorem 5 (Truncated Normal Form). Let $H = \langle A, h_1, \dots, h_n \rangle$ be an LB bimorphism, then there exists an LB bimorphism $H' = \langle A', \langle h'_i \rangle_{i \in [1, n]} \rangle$ in TNF with $L(H) = L(H')$.

To prove Theorem 5 we will describe transformation operations which break apart the homomorphic images in a bimorphism, while keeping its language unchanged. We then give a proof of Theorem 5 at the end of Sect. 3.1.

3.1 Transforming Bimorphisms into TNF

Intuition. We will describe *truncation*, a procedure to transform an LB bimorphism H into an equivalent bimorphism in TNF. Truncation is done in steps where each takes a single rule r in the FTA of the bimorphism and one of the homomorphisms h – the *selected homomorphism* – and creates a new bimorphism \hat{H} . r is replaced with three new rules with labels that have homomorphic images of lower height under a newly constructed homomorphism \hat{h} . This operation will ensure that \hat{H} is equivalent to H .

Before we give the details of the construction, let us build the intuition behind it. Assume that the rule to be replaced is $X \rightarrow r(Y, Z)$ and the homomorphism to be truncated has $h(r) = f(t_1, t_2)$. For a concrete example see Fig. 1(b), which shows the result of applying a single truncation step to the corresponding rule from Fig. 1(a) and the second homomorphism. If t_1 and t_2 each have a variable as leaf, both labeled with distinct variable symbols, as in r_1 in Fig. 1(a), we apply the *2-case*, illustrated in Fig. 3 (top). In our illustration we want to truncate h_1 for a rule of the form $X \rightarrow r(Y, Z)$. We replace r with $r^{(1)}, r^{(2)}, r^{(3)}$ so that our new FTA will derive $r^{(1)}(r^{(2)}(Y), r^{(3)}(Z))$ wherever the old FTA derived $r(Y, Z)$. We split the homomorphic image $h_1(r) = f(t_1, t_2)$ of r at its root symbol and distribute t_1 and t_2 to $r^{(2)}$ and $r^{(3)}$ as shown in Fig. 3 (top). To ensure that the homomorphic image for $\hat{h}_2(r^{(1)}(r^{(2)}(Y), r^{(3)}(Z)))$ is the same as before, we let $r^{(2)}$ and $r^{(3)}$ map to x_1 and transfer the image of r to $r^{(1)}$.

	Selected Homomorphism	Unselected Homomorphism
Before	$h_1(r) =$ 	$h_2(r) =$ 
After	$\hat{h}_1(r^{(1)}) =$  $\hat{h}_1(r^{(2)}) =$  $\hat{h}_1(r^{(3)}) =$ 	$h_2(r^{(1)}) =$  $\hat{h}_2(r^{(2)}) =$  $\hat{h}_2(r^{(3)}) =$ 

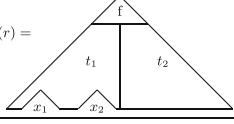
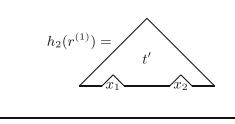
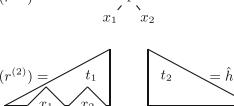
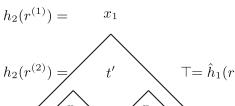
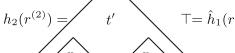
	Selected Homomorphism	Unselected Homomorphisms
Before	$h_1(r) =$ 	$h_2(r^{(1)}) =$ 
After	$\hat{h}_1(r^{(1)}) =$  $\hat{h}_1(r^{(2)}) =$  $\hat{h}_1(r^{(3)}) =$ 	$h_2(r^{(1)}) =$  $h_2(r^{(2)}) =$  $\top = \hat{h}_1(r^{(3)})$

Fig. 3. (Top) Truncation in 2-case and (bottom) truncation in the 1-case.

We cannot apply the 2-case for every rule. Consider r_3 in Fig. 1(a). If we split this rule as in the 2-case, then the x_2 reference is moved into the same subtree as the x_1 reference and we can no longer coordinate at the first rule. If only one or neither of t_1 and t_2 have a variable as a leaf, we apply the 1-case. Now we will have $r^{(1)}(r^{(2)}(Y, Z), r^{(3)})$ whenever the old grammar derived $r(Y, Z)$. We split the homomorphic image $h_1(r) = f(t_1, t_2)$ of r at its root symbol and distribute t_1 and t_2 to $r^{(2)}$ and $r^{(3)}$, as shown in Fig. 3 (bottom). To ensure that the homomorphic image for \hat{h}_2 is the same as for h_2 we simply assign $r^{(1)}$ an height 0 image under \hat{h}_2 , transfer the whole image of r to $r^{(2)}$ and then assign $r^{(3)}$ some placeholder symbol \top . \top is arbitrary, as $r^{(3)}$ will be deleted by \hat{h}_2 .

If $h(r) = f(t_1)$ as for r_2 in Fig. 1(a) then the ideas of the 1-case apply. If $h(r) = f$ or $h(r) = x$ for some constant f or variable x then there is no need for a truncation step. We now move to define the truncation steps in full generality.

Truncation Step. To ease notational clutter and reduce the number of cases to distinguish, we let \mathbf{Y} denote a sequence of 2 or less state symbols, and use this notation to denote rules $\mathbf{r} = X \rightarrow r(\mathbf{Y})$ of rank 0, 1 or 2. We refer to \mathbf{Y}_1 or \mathbf{Y}_2 only when the context guarantees that the rule has appropriate rank. We further let for $j \in \{1, 2\}$, $\neg j = 3 - j$.

A truncation step takes as arguments an LB bimorphism $H = \langle A, h_1, \dots, h_n \rangle$, with $A = \langle \Sigma, Q, R, S \rangle$, a rule $\mathbf{r} = X \rightarrow r(\mathbf{Y}) \in R$, and $i_0 \in [1, n]$ indicating the selected homomorphism. It creates a new bimorphism $\text{trunc}(H, \mathbf{r}, i_0) = \langle \hat{A}, \hat{h}_0, \dots, \hat{h}_n \rangle$ where \mathbf{r} has been replaced with three new truncated rules, $\mathbf{r}^{(1)}$, $\mathbf{r}^{(2)}$ and $\mathbf{r}^{(3)}$. The truncation step is only defined if $\text{height}(h_{i_0}(r)) > 1$.

Definition 6 (Truncation Step). With H, \mathbf{r}, i_0 as above, let $\text{trunc}(H, \mathbf{r}, i_0) = \langle \hat{A}, \hat{h}_0, \dots, \hat{h}_n \rangle$, where $\hat{A} = \langle \hat{\Sigma}, \hat{Q}, \hat{R}, S \rangle$ and the following hold:

$$\begin{aligned}\hat{\Sigma} &= \Sigma \cup \{(r^{(1)}, 2), (r^{(2)}, k_2), (r^{(3)}, k_3)\} \\ \hat{Q} &= Q \cup \{X^{(2)}, X^{(3)}\} \\ \hat{R} &= (R \setminus \{\mathbf{r}\}) \cup \{\mathbf{r}^{(1)}, \mathbf{r}^{(2)}, \mathbf{r}^{(3)}\}\end{aligned}$$

where $X^{(2)}, X^{(3)} \notin Q$, $r^{(1)}, r^{(2)}, r^{(3)} \notin \Sigma$, $\mathbf{r}^{(1)} = X \rightarrow r^{(1)}(X^{(2)}, X^{(3)})$, $\forall i \in [1, n] \forall r' \neq r \in R$, $\hat{h}_i(r') = h_i(r')$ and

2-case: If $h_{i_0}(r) = f(t_1, t_2)$ for some binary f and $\exists j \in \{1, 2\}$ s.t. x_j is a leaf in t_1 and $x_{\neg j}$ is one in t_2 , then $\mathbf{r}^{(2)} = X^{(2)} \rightarrow r^{(2)}(\mathbf{Y}_1)$, $\mathbf{r}^{(3)} = X^{(3)} \rightarrow r^{(3)}(\mathbf{Y}_2)$, $k_2 = k_3 = 1$ and

$$\begin{aligned}\hat{h}_{i_0}(r^{(1)}) &= f(x_j, x_{\neg j}) & \hat{h}_i(r^{(1)}) &= h_i(r) \\ \hat{h}_{i_0}(r^{(2)}) &= t_j[x_1/x_2] & \hat{h}_i(r^{(2)}) &= x_1 \\ \hat{h}_{i_0}(r^{(3)}) &= t_{\neg j}[x_1/x_2] & \hat{h}_i(r^{(3)}) &= x_1\end{aligned}$$

$\forall i \in [1, n], i \neq i_0$.

1- and 0-cases: If $h_{i_0}(r) = f(t_1, t_2)$ for some binary f where t_2 has no variable symbol as a leaf, i.e., $t_2 \in T_\Gamma$ with $\Gamma \cap \mathcal{V} = \emptyset$, then $r^{(2)} = X^{(2)} \rightarrow r^{(2)}(\mathbf{Y})$, $r^{(3)} = X^{(3)} \rightarrow r^{(3)}$, $k_2 = \text{ar}(r)$, $k_3 = 0$ and

$$\begin{array}{ll} \hat{h}_{i_0}(r^{(1)}) = f(x_1, x_2) & \hat{h}_i(r^{(1)}) = x_1 \\ \hat{h}_{i_0}(r^{(2)}) = t_1 & \hat{h}_i(r^{(2)}) = h_i(r) \\ \hat{h}_{i_0}(r^{(3)}) = t_2 & \hat{h}_i(r^{(3)}) = \top. \end{array}$$

$\forall i \in [1, n]$, $i \neq i_0$. If $h_{i_0}(r) = f(t_1, t_2)$ and t_1 has no variable as a leaf but t_2 has a variable x_j as a leaf, the above holds exchanging k_3 , t_2 , $\mathbf{r}^{(3)}$, $r^{(3)}$, $X^{(3)}$ with, resp., k_2 , t_1 , $\mathbf{r}^{(2)}$, $r^{(2)}$, $X^{(2)}$ everywhere. If $h_{i_0}(r) = f(t_1)$ for some unary f , then we can use the construction with: $\hat{h}_{i_0}(r^{(1)}) = f(x_1)$, $\hat{h}_{i_0}(r^{(3)}) = \top$.

Truncation Equivalence. We now prove that a truncation step leaves the language unchanged by showing that for every derivation tree in the original bimorphism there is one derivation tree in the new bimorphism which evaluates to the same values and vice versa.

Let $H = \langle A, h_1, \dots, h_n \rangle$ be an LB bimorphism with $A = \langle \Sigma, Q, R, S \rangle$, $\mathbf{r} = X \rightarrow r(\mathbf{Y}) \in R$, $i_0 \in [1, n]$ and $\hat{H} = \langle \hat{A}, h_1, \dots, h_n \rangle = \text{trunc}(H, i_0, \mathbf{r})$ with $\hat{A} = \langle \hat{\Sigma}, \hat{Q}, \hat{R}, S \rangle$. For ground terms t_1, \dots, t_n we write $t[t_1, \dots, t_n]$ as a shorthand for $t[t_1/x_1] \dots [t_n/x_n]$. For $t \in \mathcal{T}(\Sigma)$ and $t' \in \mathcal{T}(\hat{\Sigma})$ we let $t \equiv t'$ hold iff $\forall i \in [1, n], h_i(t) = \hat{h}_i(t')$.

Lemma 7 (Soundness). There exists exactly one term $c_r \in \mathcal{T}(\hat{\Sigma}, \mathcal{V}_{\text{ar}(r)})$ s.t. $X \rightarrow_{(\mathbf{r}^{(1)}; \mathbf{r}^{(2)}; \mathbf{r}^{(3)})} c_r[\mathbf{Y}]$ and for any sequence¹ of equivalent ground terms $\langle t_{Y_k} \equiv \hat{t}_{Y_k} \rangle_{k \in [1, \text{ar}(r)]}$ it holds that $r(t_{Y_1}, \dots, t_{Y_{\text{ar}(r)}}) \equiv c_r[\hat{t}_{Y_1}, \dots, \hat{t}_{Y_{\text{ar}(r)}}]$.

Proof. This lemma follows from the construction of $r^{(1)}$, $r^{(2)}$ and $r^{(3)}$. For space reasons, we develop the argument only for the 2 case with x_1 occurring in the left subtree. Assume that r is binary, i.e., $\mathbf{r} = X \rightarrow r(Y, Z)$, and that $h_{i_0}(r) = f(t_1, t_2)$ for some f , with x_1 occurring in t_1 and x_2 occurring in t_2 . It suffices to take $c_r = r^{(1)}(r^{(2)}(x_1), r^{(3)}(x_2))$. The uniqueness of c_r follows because we chose fresh state symbols. The equivalence follows from our construction and by computing the homomorphic images of $r^{(1)}(r^{(2)}(Y), r^{(3)}(Z))$ and $r(Y, Z)$.

Other cases are analogous. For instance, if $\mathbf{r} = X \rightarrow r(Y)$ and $h_2(r) = f(t_1)$ for some f (instance of 1-case), one must let $c_r = r^{(1)}(r^{(2)}(x_1), r^{(3)})$. \square

Theorem 8. For bimorphisms H and \hat{H} if $\hat{H} = \text{trunc}(H, \mathbf{r}, i_0)$, for some rule \mathbf{r} of the FTA of H and h_{i_0} some homomorphism of H , then $L(H) = L(\hat{H})$.

Proof. To show this we use Lemma 7 to show equivalence back and forth:

- (Forth) $\forall P \in Q, t \in \mathcal{T}(\Sigma)$ s.t. $P \rightarrow_A^* t$, there exists $\hat{t} \in \mathcal{T}(\hat{\Sigma})$ s.t. $t \equiv \hat{t}$ and $P \rightarrow_{\hat{A}}^* \hat{t}$.

¹ Recall that $\text{ar}(r_0) \in \{0, 1, 2\}$: we use sequences to avoid distinguishing between cases.

- (Back) $\forall P \in Q, \hat{t} \in \mathcal{T}(\hat{\Sigma})$ s.t. $P \xrightarrow{A}^* \hat{t}$, there exists $t \in \mathcal{T}(\Sigma)$ s.t. $t \equiv \hat{t}$ and $P \xrightarrow{A}^* t$.

Note that we only consider trees that can be derived from the states in the original state set Q , i.e., we do not consider trees that have $r^{(2)}$ or $r^{(3)}$ at their root. Both directions can be shown by using Lemma 7 in an induction over the number of rewrite steps involved in $\xrightarrow{A}^*/\xrightarrow{\hat{A}}^*$. \square

Using Truncation to Achieve TNF. Let $H = \langle A, h_1, \dots, h_n \rangle$ be an LB bimorphism. It remains to show that applying successive truncation steps to H eventually yields TNF. To prove this, we define a quantity $\mathcal{Q}(H)$ and show that it strictly decreases with each truncation step.

Definition 9 (\mathcal{Q}). Let Σ be the signature of A . For $k \in \mathbb{N}$, let $\#_H(k) = |\{(f, i) \mid \text{height}(h_i(f)) = k\}|$ denote the number of homomorphic images of height k in H . We define $\mathcal{Q}(H)$ as the infinite sequence $\langle \#_H(k) \rangle_{k \in \mathbb{N}}$, and we let \triangleleft denote the well founded partial order on $\mathbb{N}^\mathbb{N}$ defined as $\mathbf{s} \triangleleft \mathbf{s}'$ iff there exists $k_0 \in \mathbb{N}$ such that $s_{k_0} < s'_{k_0}$ and $\forall k \in \mathbb{N}, k > k_0 \rightarrow s_k = s'_k$.

Lemma 10 (Saturation). If $\hat{H} = \text{trunc}(H, \mathbf{r}, i_0)$ is defined, $\mathcal{Q}(\hat{H}) \triangleleft \mathcal{Q}(H)$.

Proof. Let $\mathbf{r} = X \rightarrow r(\mathbf{Y})$ and assume that \hat{H} is defined. This implies that $\text{height}(h_{i_0}(r)) > 1$. $\#_{\hat{H}}(\text{height}(h_{i_0}(r))) = \#_H(\text{height}(h_{i_0}(r))) - 1$ since $h_i(r)$ has been truncated, and for $k \neq \text{height}(h_{i_0}(r))$ we can have $\#_{\hat{H}}(k) \neq \#_H(k)$ only if $k < \text{height}(h_{i_0}(r))$ since all $h_{i \neq i_0}(r)$ have been transferred to exactly one of the new labels, and the only “new” images created are either of height $h_{i_0}(r) - 1$, 0, or 1. Hence $\mathcal{Q}(\hat{H}) \triangleleft \mathcal{Q}(H)$. \square

We can now prove Theorem 5:

Proof (Proof of Theorem 5). Since \triangleleft is well founded, Lemma 10 immediately entails that only finitely many truncation steps can be applied to a bimorphism. A truncation step is possible iff H is not in TNF. Since by Theorem 8 the language is preserved after each step, we have proved Theorem 5. \square

It can furthermore be seen from the truncation operations that the truncation procedure applied to a LB bimorphism $H = \langle A, h_1, \dots, h_n \rangle$ produces a bimorphism $H' = \langle A', \langle h'_i \rangle_{i \in [1, n]} \rangle$ in TNF s.t. $|A'| = O(|A| \times (\sum_{i \in [1, n]} \text{height}(h_i)))$. This is linear in $|A|$, hence the reduction of a LB bimorphism to TNF can be computed in linear time.

Bimorphisms can be extended to weighted bimorphisms which define a function from tuples of trees to weights [9] by associating each rule of the underlying FTA with weights. The arguments we used to show that TNF can be achieved for any LB bimorphism are almost identical to those that would be used for the weighted case. We state the following corollary without a proof.

Corollary 11 (Weighted TNF). For a weighted LB bimorphism there is a weighted LB bimorphism in TNF over the same algebras that defines the same function from tuples to weights.

4 Complexity of the Translation Problem

Now that we know how to bring any LB bimorphism into TNF, we turn back to the program of Sect. 2.3 and use TNF to upper bound the time and space complexity of the translation problem for bimorphisms, and thereby, for IRTGs. The key insight is the following lemma:

Lemma 12. *The size of the invhom automaton $h^{-1}(F)$ of an FTA F is linear in $|F|$ and can be computed in time linear in $|F|$, if the homomorphism h has $\text{height}(h) \leq 1$.*

Proof. We only show the construction of $h^{-1}(F)$. Let Σ be the domain of h and U be a fresh state. We first add rules to $h^{-1}(F)$ allowing us to expand U into any term over Σ , which requires $|\Sigma|$ distinct rules with only U as parent and child-states. For any f with $h(f) = r(x^{(1)}, \dots, x^{(n)})$ in Σ , and rule $X \rightarrow r(X_1, \dots, X_k)$ we add a rule $X \rightarrow f(X^{(1)}, \dots, X^{(\text{ar}(f))})$ to $h^{-1}(F)$, where $X^{(i)} = X_j$ if $x^{(j)} = x_i$ for some $j \in [1, \text{ar}(\sigma)]$ and $X^{(i)} = U$ otherwise. For every f with $h(f) = x_i$ and state X of F we add $X \rightarrow f(X^{(1)}, \dots, X^{(\text{ar}(f))})$ to $h^{-1}(F)$ with $X^{(i)} = X$ and all the other $X^{(j)} = U$.

An immediate corollary bounds the translation complexity for bimorphisms:

Corollary 13. *Let H be a bimorphism in TNF. From the translation algorithm given in Sect. 2.3 and Lemma 12 follows that the time complexity of the translation problem for H on F is linear in $|F|$.*

Based on Theorem 5, Corollary 13 can be extended to any LB bimorphism:

Theorem 14. *Let H be an LB bimorphism. The translation problem for H has a time complexity linear in the size of the input FTA.*

Finally, from the discussion of Sect. 2.3.

Corollary 15. *Let H be an LB bimorphism and $\mathbb{G} = \langle H, \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ an IRTG based on H . The translation and parsing problems for \mathbb{G} on an input o are solvable in time $O(|\mathcal{D}_o|)$, where \mathcal{D}_o is a decomposition automaton for o .*

Corollary 15 is of special importance in computational linguistics, where one uses algebras $\langle A_i \rangle_{i \in [1, n]}$ for which links between the sizes of the decomposition automata and the sizes of the input objects are well known. Corollary 15 thus provides a very general and immediate way to bound the complexity of the translation problem for any formalism encodable as an IRTG with LB bimorphisms.

Many formalisms used in computational linguistics can be encoded as IRTGs using linear homomorphisms. However, this may require an underlying grammar over a non-binary signature. [2] provides a generic binarization procedure which finds an equivalent IRTG over a binary signature whenever a rule-by-rule binarization is actually possible. Corollary 15 therefore extends to the wider class of IRTGs on which the procedure of [2] succeeds.

All of the above results can be directly generalized to translation problems taking multiple FTAs F_1, \dots, F_n (for bimorphisms) or objects $o_{i_1} \in D_{A_{i_1}} \dots o_{i_n} \in D_{A_{i_n}}$ as inputs, as well as weighted bimorphisms and IRTGs.

5 Conclusion

We have described how to transform LB bimorphisms into truncated normal form and shown that computing the translation of an RTL under an LB bimorphism is linear in the size of the input automaton. As a corollary, we obtained a general result on the translation complexity of any binarizable IRTG: it is linear in the size of the decomposition automaton. This yields immediate translation complexity results for any grammar formalism encodable as an IRTG – a class encompassing a wide variety of formalisms in practical use in computational linguistics. Bounding complexity in terms of the decomposition automaton is natural as they are the only algebra specific component of the translation algorithm.

In future research, we will investigate further uses of the TNF. One immediate application is in grammar induction for IRTGs, i.e. the problem of learning a weighted IRTG from only observations of input/output tuples (e.g. for mapping strings to graphs [12]). One reason why this problem is hard in practice is because the space of IRTGs that need to be considered is huge. The results of this paper imply that we can restrict this search to IRTGs in TNF in order to effectively cover all LB grammars. We will also look into new efficient parsing algorithms for IRTGs which exploit TNF.

References

1. Arnold, A., Dauchet, M.: Morphismes et bimorphismes d’arbres. *Theoret. Comput. Sci.* **20**(1), 33–93 (1982). [https://doi.org/10.1016/0304-3975\(82\)90098-6](https://doi.org/10.1016/0304-3975(82)90098-6)
2. Büchse, M., Koller, A., Vogler, H.: Generic binarization for parsing and translation. In: Proceedings of 51st ACL, pp. 145–154 (2013)
3. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M., Löding, C.: Tree automata techniques and applications (2007). <http://tata.gforge.inria.fr/>
4. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages*, vol. 3. Springer, Berlin (1997). https://doi.org/10.1007/978-3-642-59126-6_1
5. Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Initial algebra semantics and continuous algebras. *J. ACM* **24**(1), 68–95 (1977). <https://doi.org/10.1145/321992.321997>
6. Groschwitz, J., Koller, A., Johnson, M.: Efficient techniques for parsing with tree automata. In: Proceedings of 54th ACL, pp. 7–12 (2016). <https://doi.org/10.18653/v1/P16-1192>
7. Groschwitz, J., Koller, A., Teichmann, C.: Graph parsing with S-graph grammars. In: Proceedings of 53rd ACL and 7th ICJNLP, pp. 1481–1490 (2015). <https://doi.org/10.3115/v1/P15-1143>
8. Joshi, A.K., Schabes, Y.: Tree-adjoining grammars. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages*, vol. 3. Springer, Berlin (1997). https://doi.org/10.1007/978-3-642-59126-6_2
9. Koller, A., Kuhlmann, M.: A generalized view on parsing and translation. In: Proceedings of 12th IWPT, pp. 2–13 (2011)

10. Lautemann, C.: The complexity of graph languages generated by hyperedge replacement. *Acta Informatica* **27**, 399–421 (1990). <https://doi.org/10.1007/BF00289017>
11. Maletti, A., Graehl, J., Hopkins, M., Knight, K.: The power of extended top-down tree transducers. *SIAM J. Comput.* **39**, 410–430 (2009). <https://doi.org/10.1137/070699160>
12. Peng, X., Song, L., Gildea, D.: A synchronous hyperedge replacement grammar based approach for AMR parsing. In: Proceedings of 19th CONLL, pp. 32–41 (2015). <https://doi.org/10.18653/v1/K15-1004>
13. Shieber, S.: Bimorphisms and synchronous grammars. *J. Lang. Model.* **2**(1), 51–104 (2014)

Author Index

- Andres Montoya, J. 93
Asinowski, Andrei 195
Bacci, Giorgio 271
Bacher, Axel 195
Bakinova, Ekaterina 68
Banderier, Cyril 195
Basharin, Artem 68
Batmanov, Igor 68
Berdinsky, Dmitry 245
Bozzelli, Laura 80
Bulatov, Andrei A. 1
Chatain, Thomas 258
Cicalese, Ferdinando 207
Comlan, Maurice 258
Delfieu, David 258
Elder, Murray 220
Fichte, Johannes K. 130
Fijalkow, Nathanaël 271
Fiterău-Broștean, Paul 182
Ganardi, Moses 26
Gerdjikov, Stefan 143
Gittenberger, Bernhard 195
Goh, Yoong Kuan 220
Güler, Demen 156
Hecher, Markus 130
Hucke, Danny 26
Jain, Sanjay 169
Jezequel, Loïg 258
Kociumaka, Tomasz 232
Koller, Alexander 308
Krebs, Andreas 156
Kuek, Shao Ning 169
Lange, Klaus-Jörn 156
Larsen, Kim G. 271
Lipták, Zsuzsanna 207
Lohrey, Markus 26
Lyubort, Konstantin 68
Mardare, Radu 271
Martin, Eric 169
Murano, Aniello 80
Nolasco, Christian 93
Okhotin, Alexander 36, 68
Pedersen, Mathias R. 271
Peron, Adriano 80
Radoszewski, Jakub 232
Rossi, Massimiliano 207
Roux, Olivier H. 258
Rytter, Wojciech 232
Saers, Markus 284
Sazhneva, Elizaveta 68
Schindler, Irina 130
Schlachter, Uli 296
Shamir, Eli 60
Shumsky, Alexey 118
Smetsers, Rick 182
Stephan, Frank 169
Teichmann, Christoph 308
Trakuldit, Phongpitak 245
Vaandrager, Frits 182
Valdats, Māris 105
Venant, Antoine 308
Waleń, Tomasz 232
Wolf, Petra 156
Wu, Dekai 284
Zhirabok, Alexey 118