EPTCS 386

Proceedings of the

16th International Conference on

Automata and Formal Languages

Eger, Hungary, September 5-7, 2023

Edited by: Zsolt Gazdag, Szabolcs Iván and Gergely Kovásznai

Published: 3rd September 2023
DOI: 10.4204/EPTCS.386
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

Zsolt Gazdag
University of Szeged, Hungary
gazdag@inf.u-szeged.hu

Szabolcs Iván
University of Szeged, Hungary
szabivan@inf.u-szeged.hu

Gergely Kovásznai
Eszterházy Károly Catholic University
kovasznai.gergely@uni-eszterhazy.hu

The 16th International Conference on Automata and Formal Languages (AFL 2023) was held in Eger, September 5-7, 2023. It was organized by the Eszterházy Károly Catholic University of Eger, Hungary, and the University of Szeged, Hungary. Topics of interest covered the theory and applications of automata and formal languages and related areas.

The scientific program consisted of invited lectures by

- Galina Jirásková (Slovak Academy of Sciences, Košice),
- Andreas Maletti (Universität Leipzig),
- Victor Mitrana (Polytechnic University of Madrid and National Institute of R&D for Biological Sciences, Bucharest),

and 18 contributed presentations.

This volume contains the texts of the invited lecturers and the 18 papers selected by the International Program Committee from a total of 23 submissions. We would like to thank everybody who submitted a paper to the conference. We are especially grateful to the invited speakers for presenting interesting new developments.

The members of the International Program Committee were

- Francine Blanchet-Sadri (UNC Greensboro),
- Henning Bordihn (Universität Potsdam),
- Miroslav Ćirić (University of Niš),
- Erzsébet Csuhaj-Varjú (Eötvös Loránd University, Budapest),
- Pál Dömösi (University of Nyíregyháza),
- Gabriele Fici (Università di Palermo),
- Zoltán Fülöp (University of Szeged),
- Zsolt Gazdag (University of Szeged, co-chair),
- Viliam Geffert (P. J. Šafárik University, Košice),
- Géza Horváth (University of Debrecen),
- Szabolcs Iván (University of Szeged, co-chair),
- Roland Király (Eszterházy Károly Catholic University, Eger),
- Gergely Kovásznai (Eszterházy Károly Catholic University, Eger),
- Martin Kutrib (Universität Gießen),
- Andreas Malcher (Universität Gießen),

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 1–2, doi:10.4204/EPTCS.386.0

- František Mráz (Charles University, Prague),
- Benedek Nagy (Eastern Mediterranean University, Famagusta; Eszterházy Károly Catholic University, Eger),
- Chrystopher L. Nehaniv (University of Waterloo),
- Giovanni Pighizzini (University of Milan),
- Agustín Riscos-Nunez (Universidad de Sevilla),
- Kai Salomaa (Queen's University, Kingston),
- Petr Sosík (Silesian University in Opava),
- Bianca Truthe (Universität Gießen),
- György Vaszil (University of Debrecen).

The members of the Steering Committee overseeing the AFL series are

- András Ádám (Budapest, honorary chair),
- István Babcsányi (Budapest),
- Erzsébet Csuhaj-Varjú (Budapest),
- Pál Dömösi (Nyíregyháza, chair),
- Zoltán Fülöp (Szeged),
- Zsolt Gazdag (Szeged),
- Géza Horváth (Debrecen),
- László Hunyadvári (Budapest),
- Szabolcs Iván (Szeged),
- László Kászonyi (Szombathely),
- Gergely Kovásznai (Eger),
- Attila Nagy (Budapest),
- György Vaszil (Debrecen).

We thank all members of the Program Committee and their subreferees for their excellent cooperation in the selection of the papers. We are grateful to the Faculty of Informatics of the Eszterházy Károly Catholic University of Eger and the Institute of Informatics of University of Szeged for the local organization and financial support of AFL 2023.

Eger, September 2023.

Zsolt Gazdag, Szabolcs Iván, and Gergely Kovásznai

Operations on Boolean and Alternating Finite Automata

Galina Jirásková*
Mathematical Institute, Slovak Academy of Sciences, Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk

We examine the complexity of basic regular operations on languages represented by Boolean and alternating finite automata. We get tight upper bounds $m+n$ and $m+n+1$ for union, intersection, and difference, 2^m+n and 2^m+n+1 for concatenation, 2^n+n and 2^n+n+1 for square, m and $m+1$ for left quotient, 2^m and 2^m+1 for right quotient. We also show that in both models, the complexity of complementation and symmetric difference is n and $m+n$, respectively, while the complexity of star and reversal is 2^n. All our witnesses are described over a unary or binary alphabets, and whenever we use a binary alphabet, it is always optimal.

1 Introduction

Boolean and alternating finite automata [1, 2, 6, 7, 10, 11, 12] are generalizations of nondeterministic finite automata. They recognize regular languages, however, they may be exponentially smaller, with respect to the number of states, than equivalent nondeterministic finite automata (NFAs). While in an NFA the transition function maps any pair of a state and input symbol to a set of states that can be viewed as a disjunction of the states, in a Boolean finite automaton (BFA) the result of the transition function is given by any Boolean function with variables in the state set.

Fellah et al. [3] examined alternating finite automata (AFAs), that is, Boolean automata in which the initial Boolean function is given by a projection. They proved that every n-state AFA can be simulated by a (2^n+1)-state nondeterministic finite automaton with a unique initial state, and left as an open problem the tightness of this upper bound. An answer to this problem was given in [7, Lemma 1, Theorem 1] by describing an n-state binary AFA whose equivalent NFA with a unique initial state has at least 2^n+1 states. Here we present a different example in which the reachability and co-reachability of all singleton sets immediately implies the result.

In [3] it was also shown that given an m-state and n-state AFAs for languages K and L, the languages L^c, $K \cup L$, $K \cap L$, KL, and L^* are recognized by AFAs of at most $n, m+n+1, m+n+1, 2^m+n+1$, and 2^n+1 states, respectively, and the tightness of these upper bounds was left open as well.

Here we present the results obtained in [5, 6, 7, 8, 11] that provide the exact complexity of basic regular operations on languages represented by Boolean and alternating finite automata. Table 1 summarizes these results. It also displays the sizes of alphabet used to describe witness languages.

2 Preliminaries

Let Σ be a non-empty *alphabet* of symbols. Then Σ^* denotes the set of all *strings* over the alphabet Σ including the empty string ε. A *language* over Σ is any subset of Σ^*.

*This research was supported by the Slovak Grant Agency for Science (VEGA) under contract 2/0096/23 "Automata and Formal Languages: Descriptional and Computational Complexity".

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 3–10, doi:10.4204/EPTCS.386.1

Table 1: The complexity of basic regular operations on Boolean and alternating finite automata.

operation	BFA	$\|\Sigma\|$	AFA	$\|\Sigma\|$	source
complementation	n	1	n	1	[6, Thm. 1]
union	$m+n$	1	$m+n+1$	1	[7, Thm. 2(1) and 3(1)], [11, Thm. 4.3 and 4.4]
intersection	$m+n$	1	$m+n+1$	1	[7, Thm. 2(2) and 3(2)], [11, Thm. 4.3 and 4.4]
difference	$m+n$	1	$m+n+1$	1	[6, Thm. 13(a) and 14(a)], [11, Thm. 4.3 and 4.4]
symm. difference	$m+n$	1	$m+n$	1	[6, Thm. 13(b) and 14(b)], [11, Thm. 4.3 and 4.4]
star	2^n	2	2^n	2	[6, Thm. 12]
reversal	2^n	2	2^n	2	[6, Thm. 13(c) and 14(c)]
right quotient	2^m	2	2^m+1	2	[6, Thm. 13(d) and 14(d)]
left quotient	m	1	$m+1$	1	[6, Thm. 13(e) and 14(e)]
concatenation	2^m+n	2	2^m+n+1	2	[7, Thm. 4 and 5],[5, Thm. 6.4]
square	2^n+n	2	2^n+n+1	2	[8, Thm. 13 and 14]

A *Boolean finite automaton* (BFA) is a quintuple $A=(Q,\Sigma,\cdot,g_s,F)$ where $Q=\{q_1,q_2,\ldots,q_n\}$ is a finite non-empty set of states, Σ is a finite input alphabet, $\cdot$ is a transition function that maps $Q\times\Sigma$ into the set $\mathscr{B}_n$ of Boolean functions with variables $\{q_1,q_2,\ldots,q_n\}$, $g_s\in\mathscr{B}_n$ is the initial Boolean function, and $F\subseteq Q$ is the set of final states. The transition function $\cdot$ is extended to the domain $\mathscr{B}_n\times\Sigma^*$ as follows: For each $g\in\mathscr{B}_n$, each $a\in\Sigma$, and each $w\in\Sigma^*$, we have

- $g\cdot\varepsilon=g$,
- if $g=g(q_1,q_2,\ldots,q_n)$, then $g\cdot a=g(q_1\cdot a,q_2\cdot a,\ldots,q_n\cdot a)$,
- $g\cdot(wa)=(q\cdot w)\cdot a$.

Let $f=(f_1,f_2,\ldots,f_n)$ be a Boolean vector (*finality vector*) such that $f_i=1$ if and only if $q_i\in F$. The *language accepted by* a BFA A is the set of strings $L(A)=\{w\in\Sigma^*\mid(g_s\cdot w)(f)=1\}$. We illustrate the above mentioned notions in the following example.

Example 1. Consider the 2-state binary Boolean finite automaton $A=(\{q_1,q_2\},\{a,b\},\cdot,q_1\wedge q_2,\{q_1\})$ where the transition function $\cdot$ is defined in Table 2.

Table 2: The transition function of the BFA A.

$\cdot$	a	b
q_1	$q_1\vee q_2$	q_1
q_2	q_2	$q_1\wedge\neg q_2$

Then the string ab is accepted by A since we have

$$g_s\cdot ab=(q_1\wedge q_2)\cdot ab=((q_1\vee q_2)\wedge q_2)\cdot b=(q_1\vee(q_1\wedge\neg q_2))\wedge(q_1\wedge\neg q_2),$$

and the resulting function evaluates to 1 in the finality vector $(1,0)$. □

A BFA A is called *alternating* (AFA) if its initial function is a projection $g_s(q_1,q_2,\ldots,q_n)=q_1$; cf. [2, 3, 15]. It is *nondeterministic with multiple initial states* (MNFA) if g_s and all $q_i\cdot a$ are of the form $q_{i_1}\vee q_{i_2}\vee\cdots\vee q_{i_\ell}$. If moreover $g_s=q_1$, then A is *nondeterministic* (with a unique initial state) (NFA). If moreover all $q_i\cdot a$ are of the form q_j, then A is *deterministic* (DFA).

3 Simulations of BFAs and AFAs by MNFAs, NFAs, and DFAs

In this section we recall the trade-offs between different models of finite automata. Let us start with the simulation of BFAs by MNFAs.

Proposition 2 ([3, Theorem 4.1], [7, Lemma 1])**.** *Let L be a language accepted by an n-state BFA. Then L is accepted by a 2^n-state MNFA whose reverse is a DFA.*

Proof Idea. Let $A=(Q,\Sigma,\cdot,g_s,F)$ be a BFA with $Q=\{q_1,q_2,\ldots,q_n\}$. Let $f=(f_1,f_2,\ldots,f_n)$ be the Boolean finality vector with $f_i=1$ iff $q_i\in F$. Construct a MNFA $A'=(Q',\Sigma,\circ,I,\{f\})$ where

- $Q'=\{0,1\}^n$,
- $I=\{u\in Q' \mid g_s(u)=1\}$,
- for each $u\in Q'$ and each $a\in\Sigma$, we set $u\circ a=\{u'\in Q' \mid (q_1\cdot a,q_2\cdot a,\ldots,q_n\cdot a)(u')=u\}$.

Then $L(A)=L(A')$. □

Since the reverse of the MNFA in the proof above is a DFA, we get the next result.

Corollary 3. *If L is accepted by an n-state BFA, then L^R is accepted by a 2^n-state DFA.* □

Notice that if A is an AFA, then the MNFA A' constructed in the proof of Proposition 2 has 2^{n-1} initial states, and we get the following observation.

Corollary 4. *If L is accepted by an n-state AFA, then L^R is accepted by a 2^n-state DFA of which 2^{n-1} are final.* □

Our next aim is to get the converses of the above corollaries.

Proposition 5 ([7, Lemma 2])**.** *Let L be accepted by a 2^n-state MNFA whose reverse is a DFA. Then L is accepted by an n-state BFA.*

Proof Idea. Let $A=(Q,\Sigma,\cdot,I,F)$ be a MNFA with $Q=\{0,1\}^n$. Since A^R is a DFA, the MNFA A has a unique final state $f\in Q$, and moreover, for each $u\in Q$ and each $a\in\Sigma$ there is a unique state u' with $u'\cdot a=u$; denote this state by au. Construct a BFA $A'=(Q',\Sigma,\circ,g_s,F')$ where

- $Q'=\{q_1,q_2,\ldots,q_n\}$,
- $g_s(u)=1$ iff $u\in I$,
- $F'=\{q_i \mid f_i=1\}$,
- $(q_1\circ a,q_2\circ a,\ldots,q_n\circ a)(u)=au$.

Then $L(A)=L(A')$. □

Corollary 6. *If L is accepted by a 2^n-state DFA, then L^R is accepted by an n-state BFA.* □

Corollary 7. *If L is accepted by a 2^n-state DFA which has 2^{n-1} final states, then L^R is accepted by an n-state AFA.* □

We continue with the simulation of BFAs by DFAs.

Proposition 8 ([10, Theorem 7], [1, Theorem 2], [2, Theorem 5.2], [12, Corollary 3])**.** *Let L be a language over an alphabet Σ accepted by an n-state BFA. Then L is accepted by a DFA of at most 2^{2^n} states, and this upper bound is tight if $|\Sigma| \geq 2$.*

Proof Idea. If L is accepted by an n-state BFA, then by Proposition 2, it is accepted by a 2^n-state MNFA, and, consequently, by a 2^{2^n}-state DFA. For tightness, let K be the binary 2^n-state DFA from [12, Proposition 2] whose reversal K^R requires 2^{2^n} deterministic states. By Corollary 6, the language K^R is accepted by an n-state BFA. □

Finally, we consider the simulation of BFAs by NFAs, and provide an answer to an open problem from [3].

Theorem 9 ([7, Theorem 1])**.** *Let L be accepted by an n-state BFA. Then L is accepted by an NFA of at most 2^n+1 states. This upper bound is tight, and it can be met by a binary n-state AFA.*

Proof Idea. By Proposition 2, the language L is accepted by a 2^n-state MNFA, and, consequently, by a (2^n+1)-state NFA. For tightness, let $n \geq 2$. Let L be the language accepted by the 2^n-state MNFA $A = (Q, \{a,b\}, \cdot, I, F)$ where

- $Q = \{0,1,\ldots,2^n-1\}$,
- $I = \{0,1,\ldots,2^{n-1}-1\}$,
- $F = \{2^n-1\}$,
- $i \cdot a = \{(i+1) \bmod 2^n\}$ for each $i \in Q$,
- $0 \cdot b = \{0\}$, $(2^n-1) \cdot b = Q \setminus \{0\}$, and $i \cdot b = \emptyset$ is $i \in Q \setminus \{0, 2^n-1\}$;

see Figure 1 for an illustration 1. The reverse A^R is a 2^n-state DFA which has 2^{n-1} final states. By Corollary 7, the language L is accepted by an n-state AFA. On the other hand, each singleton set is reachable and co-reachable in the MNFA A which means that every NFA accepting L has at least 2^n+1 states by [4, Lemma 9]. □

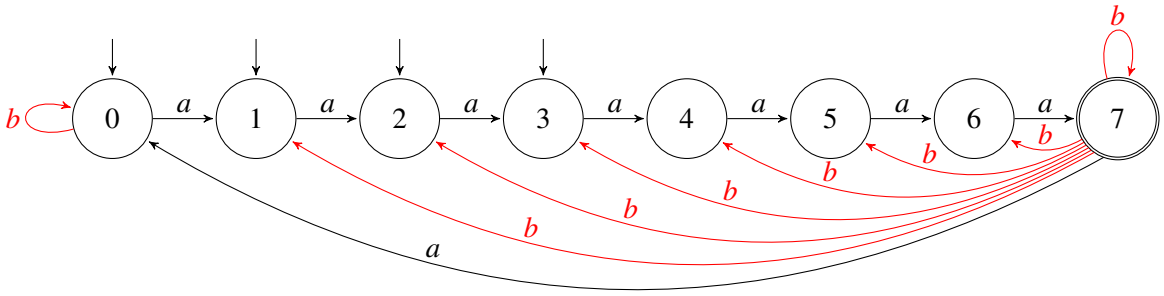


Figure 1: The MNFA A; $n = 3$.

4 Operational Complexity on Boolean and Alternating Finite Automata

In this section we use the four corollaries from the previous section to get the complexity of basic regular operations on languages represented by Boolean and alternating finite automata. The idea is as follows. Consider a binary operation and take languages K and L recognized by a 2^m-state and 2^n-state DFA, respectively, that are witnesses for the considered operation on DFAs. Then the languages K^R and L^R are accepted by an m-state and n-state BFA, respectively. Now it is enough to show that the language resulting from the operation applied to the languages K^R and L^R requires large enough BFA. In the case of AFAs, we start with DFAs with half of their states final that are hard for the considered operation on DFAs. We illustrate this idea for the concatenation operation.

Theorem 10 (Concatenation on BFAs). *Let K and L be languages over an alphabet Σ accepted by an m-state and n-state BFA, respectively. Then the language KL is accepted by a BFA of at most 2^m+n states, and this upper bound is tight if $|\Sigma| \geq 2$.*

Proof. To get an upper bound, let $A=(Q_A,\Sigma,\cdot_A,g_A,F_A)$ and $B=(Q_B,\Sigma,\cdot_B,g_B,F_B)$ be BFAs accepting the languages K and L, respectively. We first convert the BFA A to the 2^m-state MNFA $M=(Q_M,\Sigma,\cdot_M,g_M,F_M)$. Now we construct a BFA $C=(Q_M\cup Q_B,\Sigma,\cdot,g_M,F_B)$ with

$$q\cdot a=\begin{cases} q\cdot_M a, & \text{if } q\in Q_M\setminus F_M;\\ q\cdot_M a\vee g_B\cdot_B a, & \text{if } q\in F_M;\\ q\cdot_B a, & \text{if } q\in Q_B;\end{cases}$$

cf. [3, Theorem 9.2]. Then the BFA C has 2^m+n states and recognizes the language KL.

To get tightness, let K and L be Maslov's binary witnesses for concatenation on DFAs from [13], see Figure 2, accepted by a 2^n-state and 2^m-state DFA, respectively. Then every DFA accepting the language KL has at least $2^n2^{2^m}-2^{2^m-1}$ states. By Corollary 6, the languages L^R and K^R are accepted by m-state and n-state BFA, respectively. Next, we have $(L^RK^R)^R=KL$, so every DFA accepting the reverse of the concatenation L^RK^R has at least $2^n2^{2^m}-2^{2^m-1}$ states. By Corollary 3, it follows that every BFA accepting K^RL^R has at least $\lceil\log(2^n2^{2^m}-2^{2^m-1})\rceil=2^m+n$ states. □

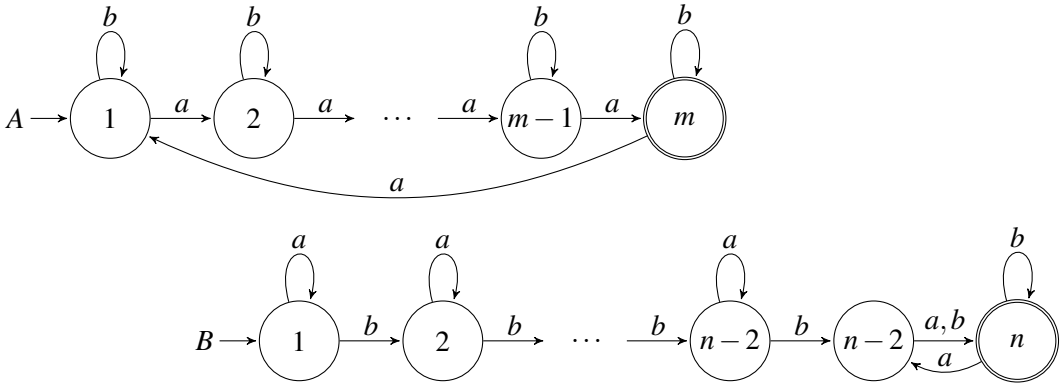


Figure 2: Maslov's witness DFAs for concatenation meeting the upper bound $m2^n-2^{n-1}$.

Theorem 11 (Concatenation on AFAs). *Let K and L be languages over an alphabet Σ accepted by an m-state and n-state AFA, respectively. Then the language KL is accepted by a AFA of at most 2^m+n+1 states, and this upper bound is tight if $|\Sigma|\geq 2$.*

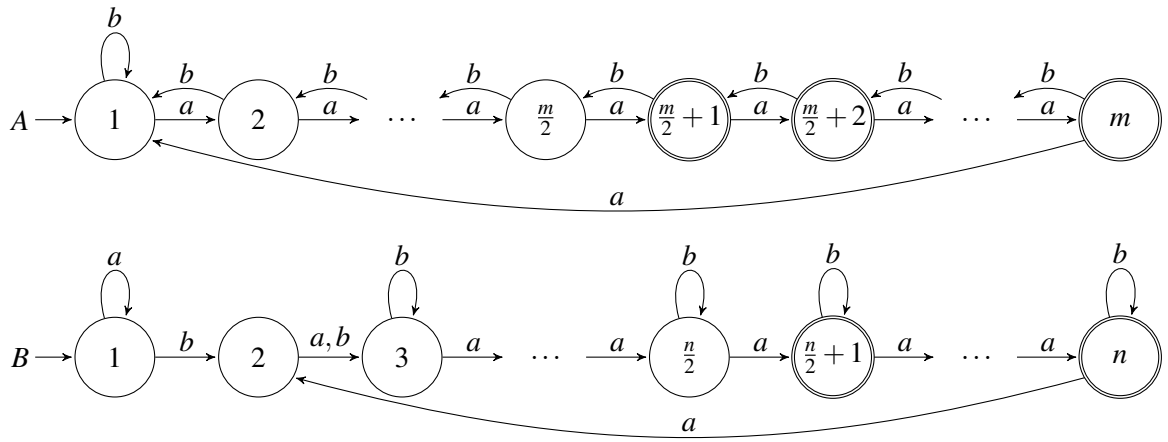


Figure 3: Witness DFAs for concatenation with half of states final meeting the upper bound $m2^n - \frac{m}{2}2^{n-1}$.

Proof. The upper bound follows from the previous theorem since one more state is enough to get an AFA equivalent to a given BFA. To get tightness, we use languages K and L accepted by 2^n-state and 2^m-state witness DFAs for concatenation with half of their states final from [5, Theorem 4.7], see Figure 3. Then the minimal DFA for KL has $2^n 2^{2^m} - 2^{n-1}2^{2^m-1}$ states, of which more that 2^{2^m+n-1} states are final [5, Lemma 6.4]. Then the languages L^R and K^R are accepted by an m-state and n-state AFA, respectively. Next, we have $(L^R K^R)^R = KL$, so every AFA for $L^R K^R$ has at least $\lceil \log(2^n 2^{2^m} - 2^{n-1}2^{2^m-1}) \rceil = 2^m + n$ states. If an AFA of $2^m + n$ states would accept $L^R K^R$, then the reverse of this language, that is, the language KL would be accepted by a DFA of 2^{2^m+n} states with 2^{2^m+n-1} final states. However, the minimal DFA for KL has more than 2^{2^m+n-1} final states, a contradiction. □

Hence, the upper bound $2^m + n + 1$ for concatenation on AFAs from [3, Theorem 9.3] is tight. This provides an answer to the second open problem from [3]. A similar idea as for concatenation also works for square, and left and right quotients. Our results for the star operation are covered by the next theorem.

Theorem 12 (Star on BFAs and AFAs). *If L is accepted by an n-state BFA, then L^* is accepted by a 2^n-state AFA. Moreover, there exists a binary language L accepted by an n-state AFA such that every BFA for L^* has at least 2^n states.*

Proof. If L is accepted by an n-state BFA, then L^R is accepted by a 2^n-state DFA by Corollary 3. Then $(L^R)^*$ is accepted by a 2^{2^n}-state DFA with half of its state final [6, Proposition 8]. Next, we have $(L^*)^R = (L^R)^*$. Hence L^* is accepted by an n-state AFA by Corollary 7.

To get tightness, let L be the Palmovský's witness DFA for star with 2^n states half of which are final [14, Theorem 4.4], see Figure 4. Then L^R is accepted by an n-state AFA by Corollary 7. Next, we have $((L^R)^*)^R = ((L^*)^R)^R = L^*$, and every DFA for L^* has at least $2^{2^n-1} + 2^{2^n-1+2^{n-1}}$ states. It follows that every BFA for $(L^R)^*$ has at least $\lceil \log(2^{2^n-1} + 2^{2^n-1+2^{n-1}}) \rceil = 2^n$ states by Corollary 3. □

Similar arguments work for reversal. If L is accepted by an n-state BFA, that L^R is accepted by a 2^n-state DFA, a special case of AFA. For tightness, we take the language L accepted by a 2^n-state Šebej's DFA from [9, Fig. 6] with half of its states final. Then L^R ia accepted by an n-state AFA, while every DFA for L^R has at least 2^{2^n} states. Hence, every BFA for $L = (L^R)^R$ has at least 2^n states by Corollary 6.

We conclude this section with Boolean operations. Denote by $\mathrm{bsc}(L)$ the number of states in a minimal, with respect to the number of states, BFA accepting L. Define $\mathrm{asc}(L)$ in an analogous way.

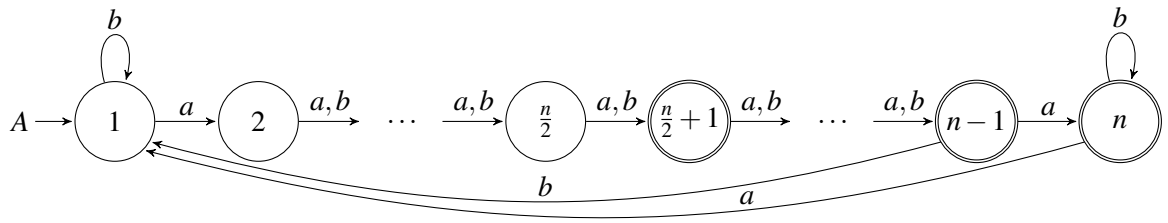


Figure 4: Witness DFA for star with half of states final meeting the upper bound $2^{n-1}+2^{n-1-\frac{n}{2}}$.

Proposition 13. *Let L be a regular language. Then $bsc(L) = bsc(L^c)$ and $asc(L) = asc(L^c)$.*

Proof. If L is accepted by a minimal n-state BFA, then L^R is accepted by a 2^n-state DFA by Corollary 3. It follows that $(L^R)^c = (L^c)^R$ is accepted by a 2^n state DFA, and therefore L^c is accepted by an n-state BFA by Corollary 6. Moreover, the language L^c cannot be accepted by a smaller BFA because otherwise the language $L = (L^c)^c$ would be accepted by a smaller BFA as well. In the case of AFAs, the DFAs for L^R and $(L^R)^c$ have 2^n states and 2^{n-1} final states, and we use Corollaries 4 and 7 to get the result. □

Theorem 14. *Let K and L be languages over Σ accepted by an m-state and n-state AFA, respectively. Then $K \cup L$ is accepted by an AFA of at most $m+n+1$ states, and this upper bound is tight if $|\Sigma| \geq 1$.*

Proof. The language $K \cup L$ can be accepted by a $(m+n)$-state BFA constructed from the two AFAs by setting the initial function to the disjunction of the corresponding initial states. The upper bound for AFAs follows. For tightness, let K be the language accepted by the unary 2^m-state DFA with state set $\{0,1,\ldots,2^m-1\}$, the initial state 0, the set of final states $\{2^{m-1},2^{m-1}+1,\ldots,2^m-1\}$, and transitions given by $i \cdot a = (i+1) \bmod 2^m$. Then $K^R = K$ is accepted by an m-state AFA. Next, let L be a language accepted by a (2^n-1)-state unary DFA with state set $\{0,1,\ldots,2^n-2\}$, the initial state 0, the set of final states $\{2^{n-1},2^{m-1}+1,\ldots,2^m-2\}$, and transitions given by $i \cdot a = (i+1) \bmod 2^m-1$. Then we can add an unreachable final state to this DFA to get an equivalent 2^n-state DFA with half of its states final. Hence $L^R = L$ is accepted by an n-state AFA. As shown in [11, Lemma 4.2, Theorem 4.4], the minimal DFA for $K \cup L$ has $2^m(2^n-1)$ states, of which more than 2^{m+n+1} are final. It follows that every AFA for $K \cup L$ has at least $m+n+1$ states. □

By Proposition 13 and De Morgan's laws, the complement of the languages described in the previous proof are witnesses for intersection. The case of difference is analogous. The same languages give a lower bound $m+n$ for symmetric difference on AFAs [11, Lemma 4.2] which is also an upper bound; notice that the symmetric difference of two DFAs with half of their states final is accepted by a DFA with half of its states final. Finally, exactly the same languages serve as witnesses for Boolean operations on BFAs [11, Theorem 4.3].

In the unary case, the reverse of any language is the same language, and the right quotient is the same as the left quotient of the corresponding languages. Moreover, we can show that the complexity of star, concatenation, and square on unary BFAs is $2n, m+n$, and $n+1$, respectively. It follows that whenever we used a binary alphabet to describe witnesses for the corresponding operations on BFAs and AFAs, it was always optimal.

The exact complexity of star, concatenation, and square on unary AFAs remains open since the complexity of these operations on unary DFAs with half of their states final is not known. The complexity of less common regular operations like shuffle, cyclic shift, or square root, would be of interest as well.

References

[1] J.A. Brzozowski & E.L. Leiss (1980): *On equations for regular languages, finite automata, and sequential networks*. *Theor. Comput. Sci.* 10, pp. 19–35, doi:10.1016/0304-3975(80)90069-9.

[2] A.K. Chandra, D. Kozen & L.J. Stockmeyer (1981): *Alternation*. *J. ACM* 28(1), pp. 114–133, doi:10.1145/322234.322243.

[3] Abdelaziz Fellah, Helmut Jürgensen & Sheng Yu (1990): *Constructions for alternating finite automata*. *Int. J. Comput. Math.* 35(1-4), pp. 117–132, doi:10.1080/00207169008803893.

[4] M. Hospodár (2021): *Power, positive closure, and quotients on convex languages*. *Theor. Comput. Sci.* 870, pp. 53–74, doi:10.1016/j.tcs.2021.02.002.

[5] M. Hospodár & G. Jirásková (2018): *The complexity of concatenation on deterministic and alternating finite automata*. *RAIRO Theor. Informatics Appl.* 52(2-3-4), pp. 153–168, doi:10.1051/ita/2018011.

[6] M. Hospodár, G. Jirásková & I. Krajňáková (2018): *Operations on Boolean and alternating finite automata*. In F.V. Fomin & V.V. Podolskii, editors: *CSR 2018*, *LNCS, vol.* 10846, Springer, pp. 181–193, doi:10.1007/978-3-319-90530-3_16.

[7] G. Jirásková (2012): *Descriptional complexity of operations on alternating and Boolean automata*. In E.A. Hirsch, J. Karhumäki, A. Lepistö & M. Prilutskii, editors: *CSR 2012*, *LNCS, vol.* 7353, Springer, pp. 196–204, doi:10.1007/978-3-642-30642-6_19.

[8] G. Jirásková & I. Krajňáková (2019): *Square on deterministic, alternating, and Boolean finite automata*. *Int. J. Found. Comput. Sci.* 30(6-7), pp. 1117–1134, doi:10.1142/S0129054119400318.

[9] G. Jirásková & J. Šebej (2012): *Reversal of binary regular languages*. *Theor. Comput. Sci.* 449, pp. 85–92, doi:10.1016/j.tcs.2012.05.008.

[10] D. Kozen (1976): *On parallelism in Turing machines*. In: *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, IEEE Computer Society, pp. 89–97, doi:10.1109/SFCS.1976.20.

[11] I. Krajňáková (2020): *Finite Automata and Operational Complexity*. Ph.D. thesis, Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics. Available at `https://www.mat.savba.sk/musav/autoreferaty/Krajnakova-dizertacna_praca.pdf`.

[12] E.L. Leiss (1981): *Succint representation of regular languages by Boolean automata*. *Theor. Comput. Sci.* 13, pp. 323–330, doi:10.1016/S0304-3975(81)80005-9.

[13] A. N. Maslov (1970): *Estimates of the number of states of finite automata*. *Soviet Math. Doklady* 11(5), pp. 1373–1375. Available at `https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=dan&paperid=35742&option_lang=eng`.

[14] M. Palmovský (2016): *Kleene closure and state complexity*. *RAIRO Theor. Informatics Appl.* 50(3), pp. 251–261, doi:10.1051/ita/2016024.

[15] S. Yu (1997): *Regular Languages*. In G. Rozenberg & A. Salomaa, editors: *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*, Springer, pp. 41–110, doi:10.1007/978-3-642-59136-5_2.

Compositions of Weighted Extended Top-Down Tree Transducers

Andreas Maletti
Universität Leipzig, Germany
andreas.maletti@uni-leipzig.de

Weighted extended top-down tree transducers are a natural generalization of the classic top-down tree transducers, but their compositions pose additional problems due to the extension as well as the weights. We review the basic composition results of [1] of the unweighted case and the basic composition approach of [2] or the weighted case. Additionally we report on recent progress on the conjectures raised in the latter reference. In particular we recently were able to obtain results that mirror the classical composition results for top-down tree transducers.

References

[1] Joost Engelfriet, Zoltán Fülöp & Andreas Maletti (2017): *Composition closure of linear extended top-down tree transducers*. *Theory of Computing Systems* 60(2), pp. 129–171, doi:10.1007/s00224-015-9660-2.

[2] Aurélie Lagoutte & Andreas Maletti (2011): *Survey: Weighted Extended Top-Down Tree Transducers Part III – Composition*. In: *Algebraic Foundations in Computer Science, Lecture Notes in Computer Science* 7020, pp. 272–308, doi:10.1007/978-3-642-24897-9_13.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 11–11, doi:10.4204/EPTCS.386.2

On the Degree of Extension of Some Models Defining Non-Regular Languages*

Victor Mitrana
Department of Information Systems,
Universidad Politécnica de Madrid
Calle Alan Turing s/n (Carretera de Valencia Km 7),
28031 Madrid, Spain
and
National Institute of R&D for Biological Sciences,
296 Independenței Bd., 060031, Bucharest. Romania
victor.mitrana@upm.es

Mihaela Păun
National Institute for R&D for Biological Sciences
296 Independenței Bd., 060031, Bucharest, Romania
mihaela.paun@incdsb.ro

This work is a survey of the main results reported for the degree of extension of two models defining non-regular languages, namely the context-free grammar and the extended automaton over groups. More precisely, we recall the main results regarding the degree on non-regularity of a context-free grammar as well as the degree of extension of finite automata over groups. Finally, we consider a similar measure for the finite automata with translucent letters and present some preliminary results. This measure could be considered for many mechanisms that extend a less expressive one.

1 Introduction

Language defining models play a central role in formal language theory, and in theoretical computer science. There have been defined very many such models with various motivations depending on the specific problems to be solved. In this work, we restrict ourselves to the most well-known devices: Chomsky generative grammars and finite automata. Regular languages are classically represented by: regular or right-linear grammars, many variants of finite automata, regular expressions, logical or algebraic formalisms. Due to their limited expressiveness, some of these models have been extended to more complex models such that the old model is just a particular case of the extended one. For instance, context-free grammars are natural extensions of regular or right-linear grammars, finite automata with valences [15], jumping automata [29], automata with translucent letters [31] are extensions of finite automata able to accept non-regular languages, etc. In their turn, context-free languages are classically represented by: context-free grammars, pushdown automata, logical and algebraic formalisms. Mainly, by the same reason as above, there have been proposed various extensions like context-sensitive grammars, grammars with regulated rewriting [11], etc.

In this work, we recall a measure for evaluating the degree of extension of a two such models, namely the context-free grammar as an extension of regular grammar, and the extended finite automaton over groups as an extension of the finite automaton. A similar measure is also considered for finite automata with translucent letters. Roughly speaking, this measure is defined as follows:
(i) by counting the maximal number of non-regular rules used in a derivation [6],
(ii) by evaluating the group memory used by the extended automata over groups [1].
(iii) by counting the number of jumping moves used by a finite automata with translucent letters.

*This work was performed through the Core Program within the National Research, Development and Innovation Plan 2022-2027, carried out with the support of MRID, project no. 23020101(SIA-PRO), contract no 7N/2022, and project no. 23020301(SAFE-MAPS), contract no 7N/2022.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 12–24, doi:10.4204/EPTCS.386.3

As far as the first measure is concerned, it is worth noting that similar investigations have been reported from the time of introducing the classes of regular and context-free languages. Here are several results giving sufficient conditions for a context-free grammar and a context-sensitive grammar to generate a regular and context-free language, respectively:

- Each context-free grammar that is not self-embedding generates a regular language [8].
- An arbitrary grammar in which no terminal is used as context and every rule generates at least one terminal, generates a context-free language [19].
- An arbitrary grammar generates a context-free language if the left side of every rule contains only one nonterminal, with terminal words as the only context [5].
- If every rule of an arbitrary grammar has as left context a word of terminal symbols at least as long as the right context, then the language generated is context-free [5].
- A grammar which has a partial ordering on its symbols, such that in every rule of the grammar every symbol on the left side is "smaller" than some symbol on the right generates a context-free language [21].
- In a grammar, the sets of terminal words generated by "one-way" and "two-ways" derivations are context-free [26, 27, 28] and [13].
- An arbitrary grammar such that in each of its non-context-free rules, the right side contains a word of terminals longer than any terminal word appearing between two nonterminals in the left side, generates a context-free language [2].

As one can see, some of the above conditions can be immediately checked, namely by examining the rules. In many cases, the complexity of a device generating a language is a function with nonnegative integer values: rational index [4], initial index [16], index of a context-free grammar [7], height of derivational trees [10], etc. Similar approaches have been reported in [6] for context-free and context-sensitive grammars, and [1] for extended automata over groups. In the sequel, we survey the most important results of these papers.

2 Preliminaries

We assume the reader is familiar with the basic definitions and concepts in formal language and automata theory and combinatorial algebra such as monoids and groups, presentations and generating sets, etc. For further details, we refer to [33] (for formal languages and automata theory), and [24, 32] (for combinatorial algebra).

We denote by $\mathbb{N}$ the set of nonnegative integers. An alphabet is a finite set of letters or symbols. For a set A we denote by $card(A)$ the cardinality of A. For a finite set V, called alphabet, we denote by $(V^*, \cdot, \varepsilon)$ the free monoid generated by V under the operation of concatenation with the neutral element ε. The elements of V^* are called words and ε is the empty word. For a word $x \in V^*$ we denote by $alph(x)$ the smallest subset of V such that $x \in alph^*(x)$. Given a set A, we denote by $\mathscr{P}_f(A)$ the family of all finite subsets of A. The free semigroup generated by V with concatenation is denoted by V^+. The length of $x \in V^*$ is denoted by $|x|$, $|x|_a$ is the number of occurrences of a in x, whereas $|x|_B$ is the number of occurrences of symbols $B \subseteq V$ in x. For a word $w = a_1 a_2 \ldots a_n$, $n \geq 1$, $a_i \in V$ for all $1 \leq i \leq n$, we write $\widetilde{w} = a_n \ldots a_2 a_1$.

By regular grammar we mean a grammar that is right-linear, hence a regular rule should be understood as a right-linear rule of one of the forms $A \to wB$, and $A \to w$, with A, B being nonterminals and w

being a word of terminals, possibly the empty word. In what follows we also use the regular expressions for defining regular languages. A context-free grammar $G = (N,T,S,P)$ is a *reduced* grammar if for any $X \in N$ we have the derivations $S \Longrightarrow^* \alpha X \beta$, for some $\alpha, \beta \in (N \cup T)^*$ (X is said to be *accessible*), and $X \Longrightarrow^* u$, with $u \in T^*$ (X is said to be *co-accessible*). A context-free grammar is *proper* if it has no λ-production (i.e. $X \to \lambda$, $X \in N$) and no chain-production (i.e., $X \to Y$, $X,Y \in N$). It is known that for every context-free grammar (which does not generate λ) there exists an equivalent proper and reduced context-free grammar.

For an arbitrary grammar $G = (N,T,S,P)$ we denote by

- $G(A) = (N,T,A,P)$, $A \in N$ the grammar in which the axiom S was replaced by another nonterminal, A.
- $G_{reg} = (N,T,S,P_{reg})$ the grammar obtained from G by considering the set $P_{reg} \subseteq P$ of regular productions of P only.
- $G_{cf} = (N,T,S,P_{cf})$ the grammar obtained from G by considering the set $P_{cf} \subseteq P$ of context-free productions of P only.

We denote by REG and CF the class of regular and context-free languages, respectively.

A finite *multiset* over a finite set A is a mapping $\sigma : A \longrightarrow \mathbb{N}$; $\sigma(a)$ expresses the number of copies of $a \in A$ in the multiset σ. In what follows, a multiset containing the elements $b_1, b_2, \ldots, b_r$, any element possibly being repeated one or more times in the sequence, will be denoted by $\langle b_1, b_2, \ldots, b_r \rangle$. Each multiset σ over a set A of cardinality n may also be viewed as an array of size n with non-negative entries.

For two functions $f,g : \mathbb{N} \longrightarrow \mathbb{N}$ we say that $f(n) \in \mathscr{O}(g(n))$ iff there is a constant $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$. Equivalently, $f(n) \in \mathscr{O}(g(n))$ iff $\limsup\limits_{n\to\infty} \dfrac{f(n)}{g(n)} < \infty$. Following [3], we say that $f(n) \in \Omega(g(n))$ iff $\limsup\limits_{n\to\infty} \dfrac{f(n)}{g(n)} > 0$. Furthermore, we say that $f(n) \in o(g(n))$ iff $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$.

3 The degree of non-regularity

Given a context-free grammar $G = (N,T,S,P)$ a derivation step in G by using the rule $r \in P$ is denoted by $\Rightarrow_r$. For a derivation in G

$$D = (S \Rightarrow_{r_1} w_1 \Rightarrow_{r_2} w_2 \cdots \Rightarrow_{r_m} w_m = w),$$

where $w \in T^*$ and $r_i \in P$ for $1 \leq i \leq m$, we define the degree of non-regularity of w with respect to D by

$$dnreg_G(w,D) = \text{card}\{i \mid r_i \notin P_{reg}, 1 \leq i \leq m\}.$$

Less formally, $dnreg_G(w,D)$ is the number of non-regular rules applied in the derivation D of w in the grammar G. The degree of non-regularity of a terminal word w with respect to the grammar G is

$$dnreg_G(w) = \begin{cases} min\{dnreg_G(w,D) \mid D \text{ is a derivation of } w \text{ in } G\}, \\ 0, \text{ if } w \notin L(G). \end{cases}$$

In other words, the degree of non-regularity of a word with respect to a grammar is computed by taking into consideration the "least non-regular derivation" if there is one.

The degree of non-regularity of a context-free grammar G as above is a mapping from $\mathbb{N}$ to $\mathbb{N}$ defined by

$$dnreg_G(n) = max\{dnreg_G(w) \mid |w| = n, w \in T^+\}.$$

As one can see, the most "non-regular" word of each length is considered.

For a function $f : \mathbb{N} \to \mathbb{N}$ we now define the complexity class

$$\begin{aligned} DNREG(f(n)) &= \{L \mid L = L(G) \text{ for some context-free grammar } G \text{ and} \\ &\quad dnreg_G(n) \in \mathscr{O}(f(n))\}. \end{aligned}$$

Otherwise stated, a language has the degree of non-regularity $f(n)$ if and only if it belongs to $DNREG(f(n))$.

In the sequel we recall the main results about the degree on non-regularity. A simple remark turns out to be useful. If G is an arbitrary context-free grammar and G_1 is the reduced grammar obtained from G, $dnreg_G(n) = dnreg_{G_1}(n)$ holds for all n, because none of the removed productions contributes in any derivation of a terminal word in G. By several considerations, a similar situation holds if G is not proper and G_1 is the proper grammar obtained from G. Therefore, the context-free grammars considered in the sequel are reduced and proper.

A context-free grammar is said to be in *quasi normal form* if all its rules of are of the following forms:

(i) $A \to a$, where a is a terminal,

(ii) $A \to aB$, where a is a terminal and B is a nonterminal,

(iii) $A \to \alpha$, where α is a word of nonterminals of length at least 2.

Proposition 1 *For every context-free grammar G there exists an equivalent context-free grammar G' in quasi normal form such that $dnreg_G(n) = dnreg_{G'}(n)$ for all n.*

If the length of α is exactly 2 in every rule $A \to \alpha$ of a grammar in quasi normal form, we say that the grammar is in *quasi Chomsky normal form*. If we have a grammar in quasi normal form, each rule $A \to \alpha$, with $|\alpha| \geq 3$ can be replaced by a sequence of rules with the right-hand side of length 2. Hence, each grammar in quasi normal can be replaced by an equivalent grammar in quasi Chomsky normal form at a price of a constant number of times higher degree of non-regularity.

Let G be a context-free grammar and c be a positive integer; we define the language

$$L(G, \leq c) = \{w \in L(G) \mid dnreg_G(w) \leq c\}.$$

Clearly, if $dnreg_G(n) \leq c$ for any $n \geq 1$, then $L(G, \leq c) = L(G)$ holds.

Theorem 1 *$DNREG(1) = REG$. A language generated by a context-free grammar is finitely-non-regular if and only if it is regular.*

Theorem 2 *For any given context-free grammar G and a positive integer c, one can algorithmically decide whether $dnreg_G(n) \leq c$.*

It is worth mentioning that $dnreg_G(n) \leq c$, for a context-free grammar G and a positive integer c, implies that $L(G)$ is regular. However, if $dnreg_G(n) > c$, then nothing can be said about the regularity of $L(G)$. Even more, if $L(G)$ is regular, it does not generally follow that $dnreg_G(n) \in \mathscr{O}(1)$.

Theorem 3 *Given an unambiguous context-free grammar G, one can algorithmically decide whether* $dnreg_G(n) \in \mathscr{O}(1)$.

The problem turns out to be undecidable even for arbitrary linear context-free grammars. It is worth mentioning that this problem is not equivalent to the problem of whether or not a given context-free grammar generates a regular language, which is known to be undecidable.

Theorem 4 *Given a linear context-free grammar G, it is undecidable whether* $dnreg_G(n) \in \mathscr{O}(1)$.

If L is a language generated by a context-free grammar such that every derivation of each word $w \in L$ of length n needs a number of non-regular productions at most linear in n, then the language is said to be "at most linearly non-regular".

Theorem 5 $CF \subseteq DNREG(n)$. *Every context-free language is at most linearly-non-regular.*

The next result gives an evaluation of the degree of non-regularity of unambiguous context-free grammar generating a non-regular language.

Theorem 6 *Let G be an unambiguous context-free grammar generating a non-regular language. Then* $dnreg_G(n) \in \Omega(n)$.

In [6] one defines a complexity measure on pushdown automata which is related, to some extent, to the pushdown space complexity of languages introduced in [17].

Let $\Gamma = (Q, V, U, \delta, q_0, Z_0, F)$ be a pushdown automaton with the set of states Q, the input alphabet V, the stack alphabet U, the transition mapping δ, the initial state q_0, the initial stack symbol Z_0 and the set of accepting states F. We say that a transition $(s, \alpha) \in \delta(q, a, A)$, with $q, s \in Q$, $a \in V \cup \{\lambda\}$, $A \in U$, $\alpha \in U^*$, is a *push move*, if $|\alpha| \geq 2$, it is a *pop move* if $\alpha = \lambda$, and it is a *neutral move* if $\alpha \in U$. In [6] one defines the *push complexity* of a language as the number of push moves needed by a pushdown automaton to accept that language. Let $w \in V^*$ and

$$C : (q_o, w, Z_0) \vdash^* (a, \lambda, \lambda)$$

be a computation in Γ accepting the input word w with empty stack, see, e.g., Chapter 6 in [25]. Then, the number of push moves in the computation C, is denoted by $push_\Gamma(w, C)$. Furthermore, for every word $w \in V^*$ we define

$$push_\Gamma(w) = \begin{cases} min\{push_\Gamma(w, C) \mid \text{C is a computation accepting } w\}, \\ \text{if } w \text{ is accepted by } \Gamma, \\ 0, \text{ if } w \text{ is not accepted by } \Gamma. \end{cases}$$

We now define the function $push_\Gamma : \mathbb{N} \longrightarrow \mathbb{N}$ by

$$push_\Gamma(n) = max\{push_\Gamma(w) \mid |w| = n\}.$$

This function is called the push complexity of Γ. Note that if a pushdown automaton has stack space complexity $f(n)$, its push complexity is a function $g(n)$ such that $f(n) \in \mathscr{O}(g(n))$. As for the degree of non-regularity we set

$$PUSH_\lambda(f(n)) = \{L \mid L = L(\Gamma) \text{ for some pushdown automaton } \Gamma \text{ accepting with empty stack and } push_\Gamma(n) \in \mathscr{O}(f(n))\}.$$

Analogously, we define

$$PUSH_f(f(n)) = \{L \mid L = L(\Gamma) \text{ for some pushdown automaton } \Gamma \text{ accepting with final states and } push_\Gamma(n) \in \mathscr{O}(f(n))\}.$$

It is known how a pushdown automaton accepting with final states can be transformed into an equivalent one accepting with empty stack (Theorem 5.1 in [22]), and conversely (Theorem 5.2 in [22]). By these constructions the equality $PUSH_\lambda(f(n)) = PUSH_f(f(n))$.

The *push* measure will turn out to be very useful for our further investigation. Indeed, we claim that the two classes of languages $PUSH_\lambda(f(n))$ and $DNREG(f(n))$ are identical.

Theorem 7 *Let L be a deterministic context-free language that is not regular. If $L \in DNREG(f(n))$, then $f(n) \in \Omega(n)$.*

Theorem 8 *Both families $DNREG(\sqrt{n})$ and $DNREG(\log n)$ contain non-regular languages.*

A very natural problem arises: Are there other sublinear functions f such that $DNREG(f)$ does contain non-regular languages? The problem of finding other sublinear functions f such that $DNREG(f)$ contains non-regular languages is of interest from a computational point of view as well. By the next theorem, functions like $\log_p(n)$, for some $p \geq 2$, are of a special interest.

Theorem 9 *Let G be a context-free grammar in quasi Chomsky normal form generating a non-regular language such that $dnreg_G(n) \leq f(n)$. Then $L(G)$ is recognizable in $\mathscr{O}(n \cdot p^{f(n)})$ time, where p is the number of nonterminals of G.*

4 The degree of extension of finite automata over groups

Let $(M, \cdot, 1)$ be a group under an operation denoted by $\cdot$ with the neutral element denoted by 1. An extended finite automaton (EFA shortly) A over the group $(M, \cdot, 1)$ is defined formally as follows. $A = (Q, V, M, f, q_0, F)$, where Q, V, q_0, F have the same meaning as in a usual finite automaton, namely the set of states, the input alphabet, the initial state and the set of final states, respectively, and $f : Q \times V \longrightarrow \mathscr{P}_f(Q \times M)$. This is actually the extension of finite automata with additive or multiplicative valences to an arbitrary group, see [15] and the references therein.

This type of automaton can be viewed as a finite automaton having a register in which any element of M can be stored, let us call it "group memory". The relation $(q, m) \in f(s, a),\ q, s \in Q,\ a \in V,\ m \in M$ means that the automaton A changes its current state s into q, by reading the symbol a on the input tape, and stores $x \cdot m$ in the register, where x is the former content of the register. The initial value stored in the register is 1.

We shall use the notation

$$(q, aw, m) \models_A (s, w, mr) \text{ iff } (s, r) \in f(q, a)$$

for all $s, q \in Q\ a \in V,\ m, r \in M$. The reflexive and transitive closure of the relation $\models_A$ is denoted by $\models_A^*$. Sometimes, the subscript identifying the automaton will be omitted when it is self-understood.

The word $x \in V^*$ is accepted by the automaton A if, and only if, there is a final state q such that $(q_0, x, 1) \models^* (q, \varepsilon, 1)$. In other words, a string is accepted if the automaton completely reads the string and reaches a final state with the content of the register being the neutral element of M. The language accepted by an EFA A over a group as above is denoted by $L(A)$.

The following simple observation will be useful in what follows. If L is a language accepted by an EFA over some group M, there exists a finitely generated subgroup N of M such that L is accepted by an EFA over N. Indeed, since the EFA over some group has finitely many transitions, only finitely many elements of the group can be associated with these transitions. Consequently, the register can only ever hold values in the subgroup of the initial group generated by these elements, so it suffices to view the automaton as an EFA over this subgroup.

It is clear that some words in the language accepted by an EFA over a group can be accepted by computations containing "non-regular transitions", that is transitions that change the contents of the group memory. The use of these transitions can make EFA more powerful than

nite automata. A very simple example is a finite automaton that accepts the language $\{a^n b^m \mid n,m \geq 1\}$. If we extend this automaton such that each transition reading an a add the value 1 to its register and each transition reading a b subtracts 1 from the register, the new automaton is an EFA over the additive group of integers that accepts the non-regular language $\{a^n b^n \mid n \geq 1\}$. Consequently, EFA over groups are able to accept non-regular languages or even not context-free languages, see, e.g.,[12]. In the remainder of the present work we study "how much" group memory, defined as the number of non-regular transitions, needs an EFA for accepting a non-regular language.

Given an EFA $A = (Q,V,M,f,q_0,F)$ over a group $(M,\cdot,1)$, $w \in L(A)$, and a computation

$$C_A(w) : (q_0,w_0,m_0) \models_A (q_1,w_1,m_1) \models_A (q_2,w_2,m_2) \models_A \ldots \models_A (q_s,\varepsilon,m_s),$$

for some $s \geq 1$, where $w_0 = w$, $m_0 = m_s = 1$, we define the multiset $E(C_A(w)) = \langle m_i^{-1} m_{i+1} \mid 0 \leq i \leq s-1 \rangle$. In words, $E(C_A(w))$ contains all the elements of M used in the computation $C_A(w)$, each element appearing in exactly the same number of copies as that of times that element was used during the computation. Further on, let $N(C_A(w))$ be the integer defined by

$$N(C_A(w)) = \sum_{x \in M, x \neq 1} E(C_A(w))(x).$$

We now define the *group memory complexity* of the computation of A on the word w by

$$gmc_A(w) = \begin{cases} \min\{N(C_A(w)) \mid C_A(w) \text{ is a computation of } A \text{ on } w\} \\ 0, \text{ if } w \notin L(A). \end{cases}$$

In other words, the group memory complexity of a word with respect to an EFA over M is computed by taking into consideration the "least non-regular computation" if there is one. The group memory complexity of an EFA as above is a mapping from $\mathbb{N}$ to $\mathbb{N}$ defined by

$$gmc_A(n) = \max\{gmc_A(w) \mid |w| = n, w \in V^*\}.$$

As one can see, the most "non-regular" word of each length is considered.

Let A be an arbitrary EFA over some group and c be a positive integer; we define the language $L(A,\leq c) = \{w \in L(A) \mid gmc_A(w) \leq c\}$. Clearly, if $gmc_A(n) \leq c$ for any $n \geq 1$, then $L(A,\leq c) = L(A)$ holds. A natural question arises: Are there EFA accepting non-regular languages with a constant group memory complexity? We give a negative answer to the question through the following result:

Theorem 10 *Given an EFA A and a positive integer c, the language $L(A,\leq c)$ is regular.*

Theorem 11 *Let M be a group such that all its finitely generated subgroups are finite. Then the language accepted by any EFA over M is regular.*

It is worth mentioning that the proof of Theorem 10 is effective, that is a finite automaton recognizing $L(A, \leq c)$ can effectively be constructed. On the other hand, it is known that a pushdown automaton may be seen as an EFA over a free group [12, 9] or an EFA over a polycyclic monoid [8, 18]. Starting from these results we prove the next result.

Theorem 12 *For every EFA A over a free group or a polycyclic monoid and a positive integer c, the problem of whether or not $gmc_A(n) \leq c$ is decidable.*

Are there other classes of groups for which the question in the previous statement is decidable? Yes, actually this happens for every finitely generated abelian group. The reason is a fundamental result in the group theory.

Theorem 13 *Every finitely generated abelian group is the direct product of a finite number of cyclic groups.*

Consequently, the language accepted by an EFA over a finitely generated abelian group is either regular or is a language accepted by an EFA over a group $\mathbb{Z}^k \times H$, where k is a positive integer and H is a finite abelian group. Moreover, $\mathbb{Z}^k$ is the additive group of vectors of size k with integer entries. We now make use of the next result (Theorem 7 in [30]):

Theorem 14 *The language accepted by an EFA over an abelian group can be: (1) regular, (2) accepted by an EFA over by $\mathbb{Z}^k$, (3) accepted by an EFA over the multiplicative group of rationals.*

It follows that if a language L accepted by an EFA over a finitely abelian group is not regular, then there exists a positive integer k such that L is accepted by an EFA over $\mathbb{Z}^k$. We can now state

Theorem 15 *For every EFA A over a finitely generated abelian group and a positive integer c, the problem of whether or not $gmc_A(n) \leq c$ is algorithmically decidable.*

We now provide an EFA over an abelian group that accept non-regular languages and has a sublinear group memory complexity, namely a function in $\mathcal{O}(\sqrt{n})$.

Lemma 1 *There exists an EFA A over $\mathbb{Z} \times \mathbb{Z}_2$ such that $L(A)$ is not regular and $gmc_A(n) \in \mathcal{O}(\sqrt{n})$.*

Inspired by the Goldstine language:

$$G = \{a^{n_1} b a^{n_2} b \ldots a^{n_p} b \mid p \geq 1, n_i \geq 0, \text{ and } n_j \neq j \text{ for some } j, 1 \leq j \leq p\},$$

we define the non-regular language

$$\begin{aligned} L \quad = \quad & \{ba^{i_1} b a^{i_2} b \ldots a^{i_k} b c^m \mid k \geq 1, i_1, i_2, \ldots, i_k > 0, \text{ and} \\ & \text{there exists } 1 \leq j \leq k \text{ such that } i_j \neq j \text{ and } m = \left\{ \begin{array}{l} j - i_j, \text{ if } j > i_j, \\ 1, \text{ if } j < i_j \end{array} \right. \}. \end{aligned}$$

It can be routinely proved that L is not regular.

As it suffices to use Theorem 7 from [30] to simply replace the group $\mathbb{Z} \times \mathbb{Z}_2$ by $\mathbb{Z}$ in the statement of previous lemma, we can state:

Theorem 16 *Let M be a group having at least one infinite cyclic subgroup. There exists an EFA A over M such that $L(A)$ is not regular and $gmc_A(n) \in \mathcal{O}(\sqrt{n})$.*

By using a similar idea to that used in the proof of Lemma 1 we prove the next result, where $\mathbb{F}_2$ is the free group of rank 2.

Lemma 2 *There exists an EFA A over the group $\mathbb{F}_2 \times \mathbb{Z}_2$ such that $L(A)$ is not regular and $gmc_A(n) \in \mathcal{O}(\log n)$.*

As $\mathbb{Z}_2$ is a finite group, we state:

Theorem 17 *There exists an EFA A over the group $\mathbb{F}_2$ such that $L(A)$ is not regular and $gmc_A(n) \in \mathcal{O}(\log n)$.*

We now give an example of a non-regular language such that any EFA over some group that accepts this language has a group memory complexity in $\Omega(n)$.

Theorem 18 *If $L(A) = \{a^n b^n \mid n \geq 1\}$, where A is an EFA over some group, then $gmc_A(n) \in \Omega(n)$.*

Along these lines, two problems remain open here:

1. Are there other abelian or non-abelian groups for which the aforementioned problem is decidable?
2. Give a class of groups $\mathscr{M}$ such that for any group $M \in \mathscr{M}$ and an EFA A over M the problem of whether or not the group memory complexity of A is finite is decidable/undecidable.

We have provided examples of EFA over some groups that accept non-regular languages and have a sublinear group memory complexity, namely a function in $\mathcal{O}(\sqrt{n})$ or $\mathcal{O}(\log n)$. Is it true that for any sublinear integer-valued function f, there is an EFA A over some group M such that $L(A)$ is not regular and $gmc_A(n) \in \mathcal{O}(f(n))$?

Theorem 18 provides a non-regular language such that any EFA over some group that accepts it has a linear group memory complexity.

It is worth mentioning that we have not considered here the deterministic variants of EFA over groups which will be investigated in another work.

5 Jumping complexity of finite automata with translucent letters

A *noneterministic finite automaton with translucent letters* (FATL) is a NFA M as above, such that the transition relation is defined in the following way. First, we define the partial relation $\circlearrowleft$ on the set of all configurations of M: $(s,xay) \circlearrowleft (p,xy)$ iff $p \in \delta(s,a)$, and $\delta(s,b)$ is not defined for any $b \in alph(x)$, $s,p \in Q$, $a,b \in V$, $x \in V^+$, $y \in V^*$. We now write

$$(p,x) \models_M (q,y), \text{ if either } (p,x) \to (q,y) \text{ or } (p,x) \circlearrowleft (q,y).$$

The subscript M is omitted when it is understood from the context.

The language accepted by M is defined by

$$L(M) = \{x \in V^* \mid (q_0,x) \models^* (f,\varepsilon), f \in F\}.$$

We want to stress that the automaton has been introduced in [31], with a slightly different definition. Actually, our definition is an FATL in the normal form in [31] without a marker for the end of the input word. This automaton is also related to the *one way jumping automaton* introduced in [29] with the difference that after each jump it returns to its previous position and does not make shift of the jumped part to the end of the word.

Let M be an FATL; we consider $w \in L(M)$, and the accepting computation in M on the input w:

$$C_M(w) : (q_0,w) \models (q_1,w_1) \models (q_2,w_2) \models \ldots \models (q_m,\varepsilon),$$

with $q_i \in Q$, $1 \leq i \leq m$, and $q_m \in F$. We define

$$jc(C_M(w)) = \{i \geq 1 \mid (q_{i-1}, w_{i-1}) \circlearrowleft (q_i, w_i)\}.$$

In words, $jc(C_M(w))$ contains all the jumping steps in the computation $C_M(w)$.

We now define the *jumping complexity* of the computation of M on the word w by

$$jc_M(w) = \begin{cases} \min\{card(jc(C_M(w))) \mid C_M(w) \text{ is a computation of } M \text{ on } w\} \\ 0, \text{ if } w \notin L(M). \end{cases}$$

In other words, the jumping complexity of a word with respect to M is computed by taking into consideration the "least non-regular computation" if there is one. Equivalently, the jumping complexity of a word with respect to M is the number of jumping steps of a computation with the minimal number of jumping steps.

The jumping complexity of an automaton M as above is a mapping from $\mathbb{N}$ to $\mathbb{N}$ defined by

$$jc_M(n) = \max\{jc_M(w) \mid |w| = n, w \in V^*\}.$$

As one can see, the most "non-regular" word of each length is considered. It is clear that $jc_M(n) \leq n$ for every jumping automaton M as one letter is consumed in every step of a computation.

Let f be a function from $\mathbb{N}$ to $\mathbb{N}$; we define the family of languages

$$LJC((f(n)) = \{L \mid \exists \text{ FATL } M \text{ such that } L = L(M) \text{ and } jc_M(n) \in \mathcal{O}(f(n))\}.$$

Let M be an arbitrary jumping automaton and c be a positive integer; we define the language $L(M, \leq c) = \{w \in L(M) \mid jc_M(w) \leq c\}$. Clearly, if $jc_M(n) \leq c$ for any $n \geq 1$, then $L(M, \leq c) = L(M)$ holds. A natural question arises: Are there FATL accepting non-regular languages with a constant jumping complexity? We give a negative answer to the question through the following result:

Lemma 3 *Given an FATL M and a positive integer c, the language $L(M, \leq c)$ is regular.*

Consequently, we have

Theorem 19 *$JCL(1)$ equals the class of regular languages.*

By Lemma 3, a sufficient condition for an FATL to accept a regular language is to be of constant jumping complexity. Is this condition necessary as well? Were this the case, the problem of deciding whether or not the jumping complexity of a given FATL is constant would be undecidable. Indeed, this decidability problem would be equivalent to decide whether or not the language accepted by an FATL is a regular language, which is not decidable, see Proposition 8 in [31]. As it was expected, the condition is not necessary. It suffices to consider the FATL M defined by the transition mapping:

$$\delta(q_0, b) = q_1, \delta(q_1, b) = q_1, \delta(q_1, c) = q_2, \delta(q_2, a) = q_3,$$

with the final state q_3. The language accepted by this automaton is $L = \{b^n a b^m c \mid n + m \geq 1\} \cup \{b^n c a \mid n \geq 1\}$, which is regular. On the other hand, $jc_M(ab^n c) = n$, for any $n \geq 1$.

However, the decidability status of the following related problem can be partially settled: *Given an FATL M and a positive integer c, is it decidable whether or not $jc_M(n) \leq c$ for all $n \geq 1$?* By modifying the construction in the proof of Lemma 3 we may infer:

Lemma 4 *Given a deterministic FATL M, there exists a deterministic FATL M′ such that* $L(M') = \{w \in L(M) \mid jc_M(w) \geq 1\}$.

This lemma is crucial for the next result.

Theorem 20 *Given a deterministic FATL M and a positive integer c, it is algorithmically decidable whether or not the jumping complexity of M is bounded by c.*

It is worth mentioning that even with this result, the decidability status of the problem "Is the jumping complexity of a deterministic FATL finite?" is still open.

Obviously, each FATL has a jumping complexity which is situated between the constant function and the identity function. In other words, $JCL(n)$ equals the class of all languages accepted by FATL. It remains to investigate what happens between these two extremes. First, we show that there are languages which require a jumping complexity in $\Omega(n)$.

Theorem 21 *If* $L(M) = \{w \mid |w|_a = |w|_b = n \geq 1\}$, *where M is an FATL, then* $jc_M(n) \in \Omega(n)$.

Corollary 1 $JCL(n) \setminus JCL(1) \neq \emptyset$.

6 Final remarks

There are still some attractive problems, in our opinion, that remained unsolved here. One of the most intriguing is the existence of a lower bound for the degree of non-regularity for context-free languages which are not regular. As we have seen, there are context-free languages which are not regular having a sublinear degree of non-regularity. We strongly suspect a more general result: $REG \subset DNREG(f)$, *strict inclusion, for any function* f *that is not a constant*. We mention a few other problems excepting the problem discussed before Theorem 9. Given a context-free grammar G, is it decidable whether or not G has the least degree of non-regularity among all grammars generating $L(G)$?

As far as the degree of extension of finite automata over groups is concerned, we have proved that given an EFA A over a free group, a polycyclic monoid, or a finitely generated abelian group and a constant c, one can algorithmically decide whether or not the group memory complexity of A is bounded by c. Along these lines, two problems remain open here:

1. Are there other abelian or non-abelian groups for which the aforementioned problem is decidable?
2. Give a class of groups $\mathscr{M}$ such that for any group $M \in \mathscr{M}$ and an EFA A over M the problem of whether or not the group memory complexity of A is finite is decidable/undecidable.

We have provided examples of EFA over some groups that accept non-regular languages and have a sublinear group memory complexity, namely a function in $\mathscr{O}(\sqrt{n})$ or $\mathscr{O}(\log n)$. Is it true that for any sublinear integer-valued function f, there is an EFA A over some group M such that $L(A)$ is not regular and $gmc_A(n) \in \mathscr{O}(f(n))$?

Theorem 18 provides a non-regular language such that any EFA over some group that accepts it has a linear group memory complexity. It is worth mentioning that [1] has not considered the deterministic variants of EFA over groups which is to be further investigated.

Some of the above problems remained open as well as regards the jumping complexity of finite automata with translucent letters.

Generally, this could be a measure for investigating the degree of extension of many mechanisms that extend a less expressive one like context-free grammars with regulated rewriting, extended various types of finite automata and tree automata over groups, jumping automata, automata with translucent letters, etc. A first step in this direction has already been done in [14].

References

[1] F. Arroyo, V. Mitrana, A. Păun, M. Păun & J.R. Sánchez Couso (2020): *On the group memory complexity of extended finite automata over groups*. *J. Log. Algebraic Methods Program.* 117, p. 100605, doi:10.1016/j.jlamp.2020.100605.

[2] B.S. Baker (1974): *Non-context-free grammars generating context-free languages*. *Information and Control* 24, pp. 231 – 246, doi:10.1016/S0019-9958(74)80038-0.

[3] J.L. Balcazar, J. Diaz & J. Gabarró (1995): *Structural Complexity*. Springer-Verlag, Berlin, doi:10.1007/978-3-642-79235-9.

[4] L. Boasson, B. Courcelle & M. Nivat (1981): *The rational index, a complexity measure for languages*. *SIAM J. Computing* 10, pp. 284 – 296, doi:10.1137/0210020.

[5] R.V. Book (1972): *Terminal context in context-sensitive grammars*. *SIAM J. Computing* 1, pp. 20 – 30, doi:10.1137/0201003.

[6] H. Bordihn & V. Mitrana (2020): *On the degrees of non-regularity and non-context-freeness*. *J. Comput. Syst. Sci.* 108, pp. 104 – 117, doi:10.1016/j.jcss.2019.09.003.

[7] B. Brainerd (1968): *An analog of a theorem about context-free languages*. *Information and Control* 11, pp. 561 – 567, doi:10.1016/S0019-9958(67)90771-1.

[8] N. Chomsky & M. P. Schützenberger (1963): *The algebraic theory of context-free languages*. In: *Studies in Logic and the Foundations of Mathematics*, North-Holland, Amsterdam, pp. 118–161, doi:10.1016/S0049-237X(08)72023-8.

[9] J. M. Corson (2005): *Extended finite automata and word problems*. *J. Algebra Comput.* 15, pp. 455 – 466, doi:10.1142/S0218196705002360.

[10] K. Culik & H. Maurer (1978): *On the Derivation of Trees*. TR Vol, 18, Institut für Informationsverarbeitung (Graz).

[11] J. Dassow & G. Păun (1989): *Regulated Rewriting in Formal Language Theory*. Springer Berlin, Heidelberg, doi:10.1007/978-3-642-74932-2.

[12] J. Dassow & V. Mitrana (2000): *Finite automata over free groups*. *J. Algebra Comput.* 10, pp. 725 – 737, doi:10.1142/S0218196700000315.

[13] R. Evey (1963): *The Theory and Application of Pushdown Store Machines*. Doctoral Dissertation, Harvard University.

[14] S. Z. Fazekas, R. Mercaş & O. Wu (2022): *Complexities for jumps and sweeps*. *Journal of Automata, Languages and Combinatorics* 27, pp. 131–149, doi:10.25596/jalc-2022-131.

[15] H. Fernau & R. Stiebe (2002): *Sequential grammars and automata with valences*. *Theoret. Comput. Sci.* 276, pp. 377 – 405, doi:10.1016/S0304-3975(01)00282-1.

[16] J. Gabarro (1983): *Initial index:a new complexity function for languages*. In: *International Colloquium on Automata, Languages, and Programming ICALP*, LNCS 154 Springer Verlag, pp. 226–236, doi:10.1007/BFb0036911.

[17] J. Gabarro (1984): *Pushdown space complexity and related full-A.F.L.s*. In: *Annual Symposium on Theoretical Aspects of Computer Science STACS*, LNCS 166 Springer Verlag, pp. 250–259, doi:10.1007/3-540-12920-0_23.

[18] R. H. Gilman & M. Shapiro (1998): *On groups whose word problem is solved by a nested stack automaton*. *arXiv:math.GR/9812028*, doi:10.48550/arXiv.math/9812028.

[19] S. Ginsburg & S. Greibach (1966): *Mappings which preserve context-sensitive languages*. *Information and Control* 9, pp. 563 – 582, doi:10.1016/S0019-9958(66)80016-5.

[20] J. Goldstine (1972): *Substitution and bounded languages*. *J. Comput. Syst. Sci.* 6, pp. 9 – 29, doi:10.1016/S0022-0000(72)80038-2.

[21] T. Hibbard (1966): *Scan Limited Automata and Context Limited Grammars*. Doctoral Dissertation, University of California at Los Angeles.

[22] J.E. Hopcroft & J.D. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass.

[23] M. Kambites (2006): *Word problems recognisable by deterministic blind monoid automata*. Theoret. Comput. Sci. 362, pp. 232 – 237, doi:10.1016/j.tcs.2006.06.026.

[24] R. C. Lyndon & P. E. Schupp (1977): *Combinatorial Group Theory*. Springer-Verlag, Berlin, doi:10.1007/978-3-642-61896-3.

[25] C. Martín-Vide, V. Mitrana & Gh. Păun (2003): *Formal Languages and Applications*. Springer Verlag, doi:10.1007/978-3-540-39886-8.

[26] G. Matthews (1963): *Discontinuity and asymmetry in phrase structure grammars*. Information and Control 6, pp. 137–146, doi:10.1016/S0019-9958(63)90179-7.

[27] G. Matthews (1964): *A note on asymmetry in phrase structure grammars*. Information and Control 7, pp. 360–365, doi:10.1016/S0019-9958(64)90406-1.

[28] G. Matthews (1967): *Two-way languages*. Information and Control 10, pp. 111–119, doi:10.1016/S0019-9958(67)80001-9.

[29] A. Meduna & P. Zemek (2012): *Jumping finite automata*. Int. J. Found. Comput. Sci. 23, pp. 1555–1578, doi:10.1142/S0129054112500244.

[30] V. Mitrana & R. Stiebe (2001): *Extended finite automata over groups*. Discrete Appl. Math. 108, pp. 287 – 300, doi:10.1016/S0166-218X(00)00200-6.

[31] B. Nagy & L. Kovács (2014): *Finite automata with translucent letters applied in natural and formal language theory*. In: Transactions on Computational Collective Intelligence, LNCS 8790 Springer Verlag, pp. 107–127, doi:10.1007/978-3-662-44994-3_6.

[32] J. J. Rotman (1995): *An Introduction to the Theory of Groups*. Springer-Verlag, Berlin, doi:10.1007/978-1-4612-4176-8.

[33] G. Rozenberg & A. Salomaa (1997): *Handbook of Formal Languages*. Springer Verlag, doi:10.1007/978-3-642-59136-5.

A Construction for Variable Dimension Strong Non-Overlapping Matrices

Elena Barcucci
University of Florence
Italy
elena.barcucci@unifi.it

Antonio Bernini
University of Florence
Italy
antonio.bernini@unifi.it

Stefano Bilotta
University of Florence
Italy
stefano.bilotta@unifi.it

Renzo Pinzani
University of Florence
Italy
renzo.pinzani@unifi.it

We propose a method for the construction of sets of variable dimension strong non-overlapping matrices basing on any strong non-overlapping set of strings.

1 Introduction

Intuitively, two matrices do not overlap if it is not possible to move one over the other in a way such that the corresponding entries match. In some recent works ([2],[3],[4]) the matrices are constructed by imposing some constraints on their rows which must avoid some particular consecutive patterns or must have some fixed entries in particular positions. The matrices of the sets there defined have the same fixed dimension.

In the present paper, we deal with matrices having different dimensions and we construct them by means a different approach: we move from any strong non-overlapping set W of strings, defined over a finite alphabet, and, in a very few words, the strings of W becomes the rows of our matrices. The method is general and once the cardinality of the strings of W with a same length is known, the cardinality of the set of matrices is straightforward.

This work could fit in the theory of bidimensional codes, as well as non overlapping sets of strings do in the theory of codes. Moreover, if the latter have been used in telecommunication systems both theory and engineering [1, 13], the matrices of our sets could be useful in the field of digital image processing, and a possible (future) application of this kind of sets is in the template matching which is a technique to discover if small parts of an image match a template image.

2 Preliminaries

Let $\mathscr{M}_{m\times n}$ be the set of all the matrices with m rows and n columns. Given a matrix $A \in \mathscr{M}_{m\times n}$, we consider a block partition

$$A = (A_{i,j}) = \begin{bmatrix} A_{11} & \dots & A_{1k} \\ \vdots & \dots & \vdots \\ A_{h1} & \dots & A_{hk} \end{bmatrix} . \tag{1}$$

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 25–34, doi:10.4204/EPTCS.386.4

Let us define $fr(A_{ij})$ the *frame* of a block A_{ij} of A. Intuitively, it is a set tracking the borders of the block which lie on the top (t), left (l), right (r) and bottom (b) border of the matrix A. More precisely, the set $fr(A_{i,j})$ is a subset of $\{t,b,l,r\}$ defined as follows:

Definition 1.

$$fr(A_{i,j}) \supseteq \begin{cases} t, & \text{if } i = 1 \\ b, & \text{if } i = h \\ l, & \text{if } j = 1 \\ r, & \text{if } j = k \end{cases} .$$

For example, if $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \end{bmatrix}$ ($h = 1$ and $k = 3$) then $fr(A_{11}) = \{t,b,l\}$, $fr(A_{12}) = \{t,b\}$, and $fr(A_{13}) = \{t,b,r\}$ since $i = h = 1$. But if

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

then $fr(A_{11}) = \{t,l\}$, $fr(A_{12}) = \{t\}$, $fr(A_{13}) = \{t,r\}$ and similarly for the other blocks. Note that in this case $fr(A_{22}) = \emptyset$.

Definition 2. *Given two matrices $A \in \mathscr{M}_{m\times n}$ and $B \in \mathscr{M}_{m'\times n'}$, they are said* overlapping *if there exist two suitable block partitions $A = (A_{ij})$, $B = (B_{i'j'})$, and some i, j, i', j' such that*

- $A_{i,j} = B_{i'j'}$, *and*
- $fr(A_{ij}) \cup fr(B_{i'j'}) = \{t,l,r,b\}$.

In the case $A = B$, the matrix is said self-overlapping.

To illustrate the definition, the following examples are given:

- Given the two matrices

$$A = \left[\begin{array}{cc|cc|c} 1 & 2 & 1 & 1 & 2 \\ 0 & 1 & 0 & 3 & 0 \\ \hline 3 & 2 & 1 & 0 & 2 \\ 0 & 1 & 3 & 1 & 3 \end{array}\right] \quad \text{and} \quad B = \left[\begin{array}{cc} 2 & 1 \\ \hline 1 & 1 \\ 0 & 3 \end{array}\right],$$

  they overlap since the entries of the blocks A_{12} and B_{21} coincide. Moreover, we have $fr(A_{12}) = \{t\}$, $fr(B_{21}) = \{l,b,r\}$ so that $fr(A_{12}) \cup fr(B_{21}) = \{l,t,b,r\}$.

- If $B = \left[\begin{array}{c|cc} 3 & 1 & 2 \\ 2 & 0 & 1 \end{array}\right]$ the matrix A (as before) and the matrix B again overlap since $A_{11} = B_{12}$ and $fr(A_{11}) \cup fr(B_{12}) = \{l,r,b,t\}$ being $fr(A_{11}) = \{l,t\}$ and $fr(B_{12}) = \{t,r,b\}$.

- Note that if $B = \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 0 & 1 & 2 \end{array}\right]$, even if $A_{11} = B_{11}$, we have $fr(A_{11}) = \{l,t\}$ and $fr(B_{11}) = \{l,t,b\}$ so that $fr(A_{11}) \cup fr(B_{11}) = \{l,b,t\} \neq \{l,t,b,r\}$. Nevertheless, the two matrices are overlapping since, considering the block partitions $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \left[\begin{array}{c|cc} 1 & 2 & 3 \\ \hline 0 & 1 & 2 \end{array}\right]$ and

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \left[\begin{array}{cc|ccc} 1 & 2 & 1 & 1 & 2 \\ \hline 0 & 1 & 0 & 3 & 0 \\ 3 & 2 & 1 & 0 & 2 \\ 0 & 1 & 3 & 1 & 3 \end{array}\right],$$

we have $A_{11} = B_{22}$ and $fr(A_{11}) \cup fr(B_{22}) = \{l,t,b,r\}$.

- As a further example we consider the particular case where $A = [A_{11}]$ and $B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$ with $B_{22} = A_{11}$. Here, we have $fr(A_{11}) \cup fr(B_{22}) = \{t,b,l,r\} \cup \{\emptyset\} = \{t,b,l,r\}$ and the two matrices are overlapping.
- We conclude this list of examples showing two matrices A and B such that, even if they have two equal blocks ($A_{11} = B_{11}$), they are not overlapping since the second condition on the frames of the blocks of Definition 2 is not fulfilled (since $fr(A_{11}) \cup fr(B_{11}) = \{t,l\} \neq \{t,b,l,r\}$):

$$A = \left[\begin{array}{cc|cc|c} 1 & 2 & 1 & 1 & 2 \\ 0 & 1 & 0 & 3 & 0 \\ \hline 3 & 2 & 1 & 0 & 2 \\ 0 & 1 & 3 & 1 & 3 \end{array}\right] \quad , \quad B = \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 0 & 1 & 1 \\ \hline 1 & 0 & 3 \end{array}\right] .$$

From these examples, it should be clear that if two matrices are overlapping, then the common block naturally induces a block partition $(A_{i,j})$ for A (and a block partition $(B_{i,j})$) such that the number of blocks in each its row and column can be not larger than 3. Figure 1 shows two examples of the least fine block partitions for two overlapping matrices A and B induced by the (gray) common block. Therefore, the block partitions 1 involved in Definition 2 are such that $h,k \in \{1,2,3\}$.

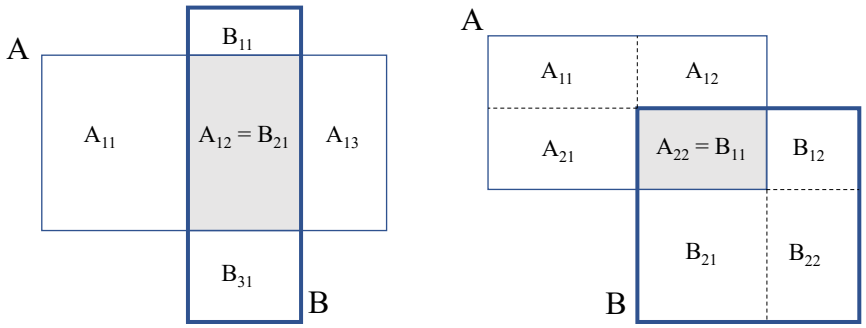


Figure 1: The least fine block partition in two examples of two overlapping matrices

We note that if a matrix is completely contained in the other, then the two matrices are overlapping according to Definition 2, as in the second to last example of the above list. In the context of strings, the scenario is different, as illustrated in the following. Two strings are said *overlapping* if there is a proper prefix of one that is equal to a proper suffix of the other. Consequently, they are said to be *non-overlapping* if there is no a proper prefix of one that is equal to a proper prefix of the other (these definitions are more formally recalled, later in this section). It can happen that, given two non-overlapping strings, one of them is an inner factor of the other, as in the case of the two binary strings 1111000 and 10. If this is not allowed, then the strings are said *strong non-overlapping* (i.e. two strings are strong non-overlapping if they are non-overlapping and if one of them is not an inner factor of the other), as in

the case of the two binary strings 1111000 and 10100. In short, being non-overlapping strings or strong non-overlapping strings are different concepts.

In our framework, if two matrices A and B are not overlapping then it can not happen that one of them (say B) is completely contained in the other. Indeed, if this were the case, then the smaller matrix B could be trivially partitioned in one block $B = B_{11}$ so that $fr(B_{11}) = \{t,b,l,r\}$). Moreover, it would be $B_{11} = A_{ij}$ for some block A_{ij} of the matrix A, and the matrices A and B would be overlapping, whatever the block $A_{i,j}$.

Therefore, when two matrices are not overlapping, we prefer to call them strong non-overlapping matrices (instead of simply non-overlapping matrices), in order to emphasize that certainly neither is contained in the other. Then, we give the following formal definitions characterizing two such matrices and a set of strong non-overlapping matrices:

Definition 3. *The matrices A and B are said* strong non-overlapping *if there does not exist any block partition for A and B, and any i,j,i',j' such that $A_{i,j} = B_{i',j'}$ or, if such block partitions exist, then $fr(A_{ij}) \cup fr(B_{i'j'}) \neq \{t,l,r,b\}$.*

Definition 4. *A set $\mathscr{P}$ of matrices is said to be* strong non-overlapping *if each matrix is self non-overlapping and if for any two matrices in $\mathscr{P}$ they are strong non-overlapping.*

For completeness, let us recall some notions about non-overlapping and strong non-overlapping sets of strings.

Given a finite alphabet Σ, a string $v \in \Sigma^*$ is said to be *self non-overlapping* (often said *unbordered* or equivalently *bifix-free*) if any proper prefix of v is different from any proper suffix of v (for more details see [11]).

Two self non-overlapping strings v, $v' \in \Sigma^*$ are said to be *non-overlapping* (or equivalently *cross bifix-free*) if any proper prefix of v is different from any proper suffix of v', and vice versa. A set of strings is said to be a *non-overlapping set* (or *cross bifix-free set*) of strings if each element of the set is slef non-overlapping and if any two strings are non-overlapping.

Definition 5. *Two non-overlapping strings v and v' are said to be* strong non-overlapping *if there do not exist $\alpha,\beta \in \Sigma^*$, with α and β not both empty, such that $v' = \alpha v \beta$ (or $v = \alpha v' \beta$).*

In other words, the strong non-overlapping property requires that the shortest string between v and v' (if any) does not occur as an inner factor in the other one ([6, 12]). For example, if $v = 1100$ and $v' = 1\mathbf{1100}100$, then v and v' are non-overlapping but they are not strong non-overlapping since v' contains an occurrence of v (in bold).

Definition 6. *A set of strings is said to be a* strong non-overlapping set *if any two strings of the set are strong non-overlapping.*

3 Construction of the set of matrices

Let $\mathscr{V}_n = \bigcup_{s \leq n} V^s$ be a variable dimension strong non-overlapping set of strings where each V^s is a non-overlapping set of strings of length s, for $s_0 \leq s \leq n$, where $s_0 \geq 2$ is the minimum string length. We now define a set of variable dimension matrices, using strings of a same length s of V^s as rows of a matrix.

In the following, the two matrices C and D of dimension $m_1 \times s$ and $m_2 \times t$, respectively, are constructed with the rows $C_i^s \in V^s$ and $D_j^t \in V^t$, with $i = 1,2,\ldots,m_1$ and $j = 1,2,\ldots,m_2$.

$$C = \begin{pmatrix} C_1^s \\ C_2^s \\ \vdots \\ \vdots \\ C_{m_1}^s \end{pmatrix} \qquad D = \begin{pmatrix} D_1^t \\ D_2^t \\ \vdots \\ D_{m_2}^t \end{pmatrix}$$

It is not difficult to show that if C and D have a different number of columns (then $s \neq t$) they can not be overlapping (see next proposition).

Unfortunately, in the case C and D have the same number of columns ($s = t$), then the two matrices can present a “vertical" overlap. More precisely:

- the matrix D could be equal to a sub-matrix of C constituted by m_2 consecutive rows of C (or vice versa):

$$C = \begin{bmatrix} C_{11} \\ C_{12} \\ C_{13} \end{bmatrix} = \begin{bmatrix} C_{11} \\ D \\ C_{13} \end{bmatrix}$$

  (with either blocks C_{11} or C_{13} possibly empty).

- the first (last) ℓ rows of D could be equal to the last (first) ℓ rows of C (or vice versa):

$$C = \begin{bmatrix} C_{11} \\ C_{12} \end{bmatrix} = \left[\begin{array}{c} C_{11} \\ \hline D_1^t \\ D_2^t \\ \vdots \\ D_\ell^t \end{array}\right] \qquad D = \begin{bmatrix} D_{11} \\ D_{12} \end{bmatrix} = \left[\begin{array}{c} D_1^t \\ D_2^t \\ \vdots \\ D_\ell^t \\ \hline D_{12} \end{array}\right].$$

In order to avoid the situations described above, we introduce a constraint for the first and the last row of each matrix: all the matrices with the same number s of columns must have the same first row $T^s \in V^s$ and the same last row $B^s \in V^s$, with $T^s \neq B^s$. Also, these two selected rows cannot appear as inner rows of any other matrix with that number s of columns. In other words, we force:

- the top row T^s of all the matrices with the same number s of columns to be the same;
- the bottom row B^s of all the matrices with the same number s of columns to be the same;
- $T^s \neq B^s$;
- the rows T^s and B^s not to occur in any other line of the matrix.

Formally, the matrices C with the same number s of columns must have the following structures:

$$C = \begin{pmatrix} T^s \\ \\ C_2^s \\ \vdots \\ \vdots \\ C_{m_1-1}^s \\ \\ B^s \end{pmatrix}$$

with $C_j^s \neq T^s, B^s$, for $j = 2,3,\ldots,m_1-1$, and $C_j^s, T^s, B^s \in V^s$.

We can now define the set $\mathscr{V}_{m\times n}^{(\leq)}$ of variable-dimension matrices as follows:

Definition 7. *Let $\mathscr{V}_n = \bigcup_{s\leq n} V^s$ be a variable dimension strong non-overlapping set of strings where each V^s is a non-overlapping set of strings of length s, for $s_0 \leq s \leq n$, where $s_0 \geq 2$ is the minimum string length. Moreover, let*

$$\mathscr{V}_{m\times n}^{(\leq)} = \bigcup M$$

be the union of the matrices M where $M \in \mathscr{M}_{h\times s}$, with $2 \leq h \leq m$ and $s_0 \leq s \leq n$, such that

$$M = \left\{ \begin{pmatrix} T^s \\ A_2^s \\ \vdots \\ A_{h-1}^s \\ B^s \end{pmatrix} \right\}$$

with $A_j^s, T^s, B^s \in V^s$ and $A_j^s \neq T^s, B^s$ for $j = 2,3,\ldots,h-1$.

The matrices $M \in \mathscr{V}_{m\times n}^{(\leq)}$ have at most m rows and n columns. They are constructed by means of $h \leq m$ strings of length $s \leq n$ belonging to $\mathscr{V}_n$. All the matrices M with the same number s of columns have the same bottom row B_s and the same top row T_s, which are not the same. Moreover, each inner row is different from T_s and B_s.

We have the following proposition:

Proposition 1. *The set $\mathscr{V}_{m\times n}^{(\leq)}$ is a strong non-overlapping set of variable-dimension matrices.*

Proof. Let $C, D \in \mathscr{V}_{m\times n}^{(\leq)}$ and suppose that C and D are two overlapping matrices: then there exists a block matrix $E \in \mathscr{M}_{r\times c}$ such that $E = C_{i,j} = D_{i',j'}$ fore some two blocks $C_{i,j}$ and $D_{i'j'}$ in two suitable block partitions of C and D, and with $fr(C_{i,j}) \cup fr(D_{i'j'}) = \{l,t,r,b\}$. We have

$$E = \begin{pmatrix} e_{11} & \ldots & e_{1c} \\ \vdots & \ldots & \vdots \\ e_{r1} & \ldots & e_{rc} \end{pmatrix} .$$

For each row e_ℓ, with $\ell = 1,2,\ldots,r$, there exist two rows $C_i, D_j \in \mathscr{V}_n$ such that one of the following cases occurs:

- $C_i = ue_\ell v$ and $D_j = e_\ell$, with either u or v possibly empty, where $u, v \in \Sigma^*$;
- $C_i = ue_\ell$ and $D_j = e_\ell v$;
- $C_i = e_\ell v$ and $D_j = ue_\ell$.

In any case, the strings C_i and D_j are not strong non-overlapping strings (since they overlap over e_ℓ) against the hypothesis $C_i, D_j \in \mathscr{V}_n$. □

We note that in the case $\mathscr{V}_n$ is a variable dimension non-overlapping set of strings (i.e. the non-overlapping property is not required to be strong), the resulting matrices are not strong non-overlapping according to Definition 2, since it is possible that one of the two matrices is completely contained in the other one as a suitable block. If we did not contemplate this possibility in Definition 2, then two matrices constructed with such a $\mathscr{V}_n$ could be considered still non-overlapping (according to a different definition of non-overlapping matrices).

Moreover, if $\mathscr{V}_n$ contains strings all of the same lengths, then Proposition 1 still holds: the matrices will have all the same number of columns.

Finally, if $|V^s|$ denotes the cardinality of the non-overlapping set V^s, it is straightforward to deduce the following formula for the cardinality of $\mathscr{V}^{(\leq)}_{m\times n}$:

$$|\mathscr{V}^{(\leq)}_{m\times n}| = \sum_{h\leq m} \sum_{s\leq n} (|V^s| - 2)^{h-2} \,. \tag{2}$$

The two terms -2 in the above formula take into account that the first and the last row in the matrices with s columns are fixed and can not occur as inner rows.

For the sake of clearness, we propose an example for the construction of a set of variable dimension strong non-overlapping matrices. Let $V^3 = \{110, 210, 310, 320\}$ and $V^5 = \{22000, 23000, 33000\}$ be two sets of non-overlapping strings over the alphabet $\Sigma = \{0, 1, 2, 3\}$. It is easily seen that $V^3 \cup V^5$ is a strong non-overlapping code. Then, we construct

$$\mathscr{V}^{(\leq)}_{4\times 5} = \mathscr{M}^{(\leq)}_{2\times 3} \cup \mathscr{M}^{(\leq)}_{3\times 3} \cup \mathscr{M}^{(\leq)}_{4\times 3} \cup \mathscr{M}^{(\leq)}_{2\times 5} \cup \mathscr{M}^{(\leq)}_{3\times 5} \cup \mathscr{M}^{(\leq)}_{4\times 5}$$

where:

$$\mathscr{M}_{2\times 3} = \left\{ \begin{pmatrix} 1 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix} \right\}$$

$$\mathscr{M}_{3\times 3} = \left\{ \begin{pmatrix} 1 & 1 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 3 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix} \right\}$$

$$\mathscr{M}_{4\times 3} = \left\{ \begin{pmatrix} 1 & 1 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 2 & 1 & 0 \\ 3 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 3 & 1 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 3 & 1 & 0 \\ 3 & 1 & 0 \\ 3 & 2 & 0 \end{pmatrix} \right\}$$

$$\mathscr{M}_{2\times 5} = \left\{ \begin{pmatrix} 2 & 2 & 0 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{pmatrix} \right\}$$

$$\mathscr{M}_{3\times 5} = \left\{ \begin{pmatrix} 2 & 2 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{pmatrix} \right\}$$

$$\mathscr{M}_{4\times 5} = \left\{ \begin{pmatrix} 2 & 2 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{pmatrix} \right\}$$

The reader can easily check that $\mathscr{V}_{4\times 5}^{(\leq)}$ is a set of variable dimension strong non-overlapping matrices having cardinality 10 according to (2).

4 Conclusions

The paper provides a simple and general method to generate a set of strong non-overlapping matrices over a finite alphabet, once a strong non-overlapping set of strings (over the same alphabet) is at our disposal. The crucial point is the constraint on the first and last rows which must be the same for all the matrices with the same number of columns.

Using the variable length strong non-overlapping sets of strings defined in [12] and [6], two different set of strong non-overlapping matrices arise which could be compared in terms of cardinality or its asymptotic behaviour.

Moreover, the construction we proposed, in the case of fixed dimension matrices, gives the possibility to list them in a Gray code sense, following the studies started in [2, 5, 7, 8, 9, 10] where different Gray codes are defined for several set of strings and matrices.

In this case, we generate the matrices moving from a set of non-overlapping strings V^s of length s and we suppose that there exists a Gray code GV^s for V^s:

$$GV^s = \{w_1, w_2 \ldots, w_t, w_{t+1}, w_{t+2}\} \text{ with } t > 0 \, .$$

Note that we require $|V^s| \geq 3$. We choose two strings from GV^s. Without loss of generality, we choose w_{t+1} and w_{t+2} and we define the set of matrices $M_{h+2,s}$ with $h+2$ rows and s columns where the first and last rows are, respectively, the strings w_{t+1} and w_{t+2}:

$$M_{h+2,s} = \left\{ \begin{pmatrix} w_{t+1} \\ C_1^s \\ \vdots \\ C_h^s \\ w_{t+2} \end{pmatrix} \middle| \, C_i^s \in V^s \setminus \{w_{t+1}, w_{t+2}\} \right\} \, .$$

Let $N_{h,s}$ be the set of matrices obtained by $M_{h+2,s}$ removing the first and last rows:

$$N_{h,s} = \left\{ \begin{pmatrix} C_1^s \\ \vdots \\ C_h^s \end{pmatrix} \middle| \, C_i^s \in V^s \setminus \{w_{t+1}, w_{t+2}\} \right\} \, .$$

Clearly, the cardinality of $N_{h,s}$ and $M_{h+2,s}$ is the same and denoting it by q it is $q = t^h$.

We now recursively define a Gray code $GN_{h,s}$ for the set $N_{h,s}$. If $h = 1$, then the list $GN_{1,s} = (w_1), (w_2), \ldots, (w_t)$ is a Gray code (since it is obtained by GV^s where the strings are read as matrices of dimension $1 \times s$). Suppose now that $GN_{h,s} = A_1, A_2, \ldots, A_q$ is a Gray code where $h \geq 1$ and $A_i \in N_{h,s}$, for $i = 1, 2, \ldots, q$. The following list $GN_{h+1,s}$of matrices, defined as block matrices,

$$GN_{h+1,s} = \left[\frac{w_1}{A_1}\right] \cdots \left[\frac{w_1}{A_q}\right] \left[\frac{w_2}{A_q}\right] \cdots \left[\frac{w_2}{A_1}\right] \cdots\cdots \left[\frac{w_t}{A_\ell}\right] \cdots \left[\frac{w_t}{A_{q+1-\ell}}\right] ,$$

where

$$\ell = \begin{cases} q, & \text{if } t \text{ is even} \\ 1, & \text{if } t \text{ is odd} \end{cases} ,$$

is easily seen to be a Gray code since the lists $A_1, A_2, \ldots, A_q$ and $w_1, w_2, \ldots, w_t$ are Gray codes for hypothesis.

Finally, adding the strings w_{t+1} and w_{t+2}, respectively, as first and last rows to all the q matrices $A_1, A_2, \ldots, A_q$ of $GN_{h,s}$ we obtain a Gray code $GM_{h+2,s}$ for the set $M_{h+2,s}$:

$$GM_{h+2,s} = \left[\begin{array}{c} w_{t+1} \\ \hline A_1 \\ \hline w_{t+2} \end{array}\right] \cdots\cdots\cdots \left[\begin{array}{c} w_{t+1} \\ \hline A_q \\ \hline w_{t+2} \end{array}\right] .$$

References

[1] D. Bajic & J. Stojanovic (2004): *Distributed sequences and search process*. In: *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, 1, pp. 514–518, doi:10.1109/ICC.2004.1312542.

[2] E. Barcucci, A. Bernini, S. Bilotta & R. Pinzani (2015): *Cross-bifix-free sets in two dimensions*. *Theoret. Comput. Sci.* 664, pp. 29–38, doi:10.1016/j.tcs.2015.08.032.

[3] E. Barcucci, A. Bernini, S. Bilotta & R. Pinzani (2017): *Non-overlapping matrices*. *Theoret. Comput. Sci.* 658, pp. 36–45, doi:10.1016/j.tcs.2016.05.009.

[4] E. Barcucci, A. Bernini, S. Bilotta & R. Pinzani (2018): *A 2D non-overlapping code over a q-ary alphabet*. *Cryptogr. Commun.* 10, pp. 667–683, doi:10.1007/s12095-017-0251-8.

[5] E. Barcucci, A. Bernini & R. Pinzani (2018): *A Gray code for a regular language*. In: *GASCom 2018, CEUR Workshop Proceedings*, 2113, pp. 87–93. Available at `https://ceur-ws.org/Vol-2113/paper8.pdf`.

[6] E. Barcucci, A. Bernini & R. Pinzani (2021): *A Strong non-overlapping Dyck Code*. In: *DLT 2021, Lecture Notes in Comput. Sci.*, 12811, pp. 43–53, doi:10.1007/978-3-030-81508-0_4.

[7] A. Bernini, S. Bilotta, R. Pinzani, A. Sabri & V. V. Vajnovszki (2014): *Prefix partitioned Gray codes for particular cross-bifix-free sets*. *Cryptogr. Commun.* 6, pp. 359–369, doi:10.1007/s12095-014-0105-6.

[8] A. Bernini, S. Bilotta, R. Pinzani, A. Sabri & V. V. Vajnovszki (2015): *Gray code orders for q-ary words avoiding a given factor*. *Acta Inform.* 52, pp. 573–592, doi:10.1007/s00236-015-0225-2.

[9] A. Bernini, S. Bilotta, R. Pinzani & V. V. Vajnovszki (2015): *A trace partitioned Gray code for q-ary generalized Fibonacci strings*. *J. Discrete Math. Sci. Cryptogr.* 18, pp. 751–761, doi:10.1080/09720529.2014.968360.

[10] A. Bernini, S. Bilotta, R. Pinzani & V. V. Vajnovszki (2017): *A Gray code for cross-bifix-free sets*. *Math. Structures Comput. Sci.* 27, pp. 184–196, doi:10.1017/S0960129515000067.

[11] J. Berstel & D. Perrin (1985): *Theory of codes*. Academic Press, Orlando.

[12] S. Bilotta (2017): *Variable-length non-overlapping codes*. *IEEE Trans. Inform. Theory* 63, pp. 6530–6537, doi:10.1109/TIT.2017.2742506.

[13] A. J. de Lind van Wijngaarden, T. J. & Willink (2000): *Frame synchronization using distributed sequences*. *IEEE Trans. Comm* 48, pp. 2127–2138, doi:10.1109/26.891223.

Duality of Lattices Associated to Left and Right Quotients

Jason Bell*
Department of Pure Mathematics
University of Waterloo, Canada
jpbell@uwaterloo.ca

Daniel Smertnig†
Institute for Mathematics and Scientific Computing
University of Graz, Austria
daniel.smertnig@uni-graz.at

Hellis Tamm‡
Department of Software Science
Tallinn University of Technology, Estonia
hellis@cs.ioc.ee

We associate lattices to the sets of unions and intersections of left and right quotients of a regular language. For both unions and intersections, we show that the lattices we produce using left and right quotients are dual to each other. We also give necessary and sufficient conditions for these lattices to have maximal possible complexity.

1 Introduction

Within the study of formal languages, a common theme is associating invariants that provide a measure of complexity of the language. A key example of this type is the entropy of languages (cf. Chomsky and Miller [4]), which gives a measure of their growth.

When one restricts to regular languages, one of the most essential notions of complexity comes from the observation that, given a finite alphabet Σ, a language $L \subseteq \Sigma^*$ is regular if and only if the number of its distinct left quotients is finite, where the left quotient of L by a word $w \in \Sigma^*$ is the language

$$w^{-1}L = \{x \in \Sigma^* : wx \in L\}.$$

In this sense, the number of distinct left quotients of a regular language provides a measure of its complexity (see the survey article [1] and references therein for more on quotient complexity). One can analogously define the right quotient of a language L by a word $v \in \Sigma^*$ to be the language

$$Lv^{-1} = \{u \in \Sigma^* : uv \in L\},$$

and again, L is regular exactly when it has a finite number of distinct right quotients. In particular, this gives an analogous notion of complexity. It should be noted, however, that these two notions of complexity do not coincide. For example, if $\Sigma = \{a,b\}$ and $L = \{\varepsilon, a, a^2, ba\}$, then the left quotients of L are the languages $L, \{\varepsilon, a\}, \{a\}, \{\varepsilon\}, \varnothing$, while the right quotients are the languages $L, \{\varepsilon, a, b\}, \{\varepsilon\}, \varnothing$.

The purpose of this paper is to show that when one instead forms lattices[1] associated with the left and right quotients of a regular language in a natural way, then a duality arises that provides a left-right

*Supported by NSERC grant RGPIN RGPIN-2022-02951.
†Supported by the Austrian Science Fund (FWF): P 36742-N.
‡Supported by the Estonian Research Council grant PRG1210.

[1] A lattice is simply a partially ordered set $(\Lambda, \leqslant)$ with the property that finite subsets have unique least upper bounds and unique greatest lower bounds; thus lattices have a join, $\vee$, and meet, $\wedge$, which are binary operations corresponding to taking respectively the least upper bound and greatest lower bound of two elements of Λ.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 35–50, doi:10.4204/EPTCS.386.5

symmetric measure of the complexity of the language in terms of quotients. To make this precise, we observe that if $L \subseteq \Sigma^*$ is a regular language with left quotients $L_0, \ldots, L_{n-1}$ and right quotients $R_0, \ldots, R_{m-1}$, then one can consider the following four lattices.

- The *left quotient union lattice*, $\mathrm{Latt}(L, \cup, \mathsf{L})$:

  the lattice whose elements are all sets that can be formed by taking a (possibly empty) union of left quotients $L_0, \ldots, L_{n-1}$.

- The *right quotient union lattice*, $\mathrm{Latt}(L, \cup, \mathsf{R})$:

  the lattice whose elements are all sets that can be formed by taking a (possibly empty) union of right quotients $R_0, \ldots, R_{m-1}$.

- The *left quotient intersection lattice*, $\mathrm{Latt}(L, \cap, \mathsf{L})$:

  the lattice whose elements are all sets that can be formed by taking a (possibly empty) intersection of left quotients $L_0, \ldots, L_{n-1}$.

- The *right quotient intersection lattice*, $\mathrm{Latt}(L, \cap, \mathsf{R})$:

  the lattice whose elements are all sets that can be formed by taking a (possibly empty) intersection of right quotients $R_0, \ldots, R_{m-1}$.

We observe that the above sets are partially ordered by inclusion and have a join operation, $\vee$, and a meet operation, $\wedge$. In the case of $\mathrm{Latt}(L, \cup, \mathsf{L})$ and $\mathrm{Latt}(L, \cup, \mathsf{R})$, the join of A and B is the union and the meet is the union of all elements of the set that are contained in $A \cap B$, where an empty union is the empty set. These two lattices have a unique smallest element (the empty set, which is the empty union) and a unique largest element, consisting of the union of all left (respectively right) quotients.

Similarly, in the case of $\mathrm{Latt}(L, \cap, \mathsf{L})$ and $\mathrm{Latt}(L, \cap, \mathsf{R})$, the meet is just the intersection and the join of two intersections of quotients, A and B, is the intersection of all quotients that contain the union $A \cup B$, where an empty intersection is taken to be Σ^*. Then these two lattices again have a unique maximal element Σ^* and a unique minimal element given by the intersection of all left (respectively right) quotients.

As a simple example, consider again the finite regular language $L = \{\varepsilon, a, a^2, ba\} \subseteq \{a, b\}^*$. Then the left quotients are the languages

$$L_0 = \{\varepsilon, a, a^2, ba\},\ L_1 = \{a\},\ L_2 = \{\varepsilon, a\},\ L_3 = \{\varepsilon\},\ L_4 = \varnothing \tag{1}$$

while the right quotients are

$$R_0 = \varnothing,\ R_1 = \{\varepsilon\},\ R_2 = \{\varepsilon, a, b\},\ R_3 = \{\varepsilon, a, a^2, ba\} \tag{2}$$

and we construct the four lattices we consider in this paper from these left and right quotients of $\{\varepsilon, a, a^2, ba\}$ in Figures 1 and 2.

Figures 1 and 2 hint at an unexpected duality. We recall that if Λ is a lattice, then we have a dual lattice Λ^*, which is Λ as a set, but where the partial order on Λ is reversed and the meet and join are exchanged. Intuitively, one can think of this as simply taking the lattice Λ and writing it "upside-down";

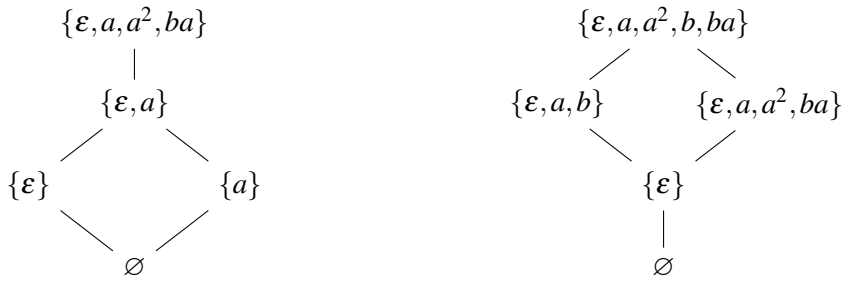


Figure 1: The lattices $\mathrm{Latt}(L,\cup,\mathsf{L})$ (left) and $\mathrm{Latt}(L,\cup,\mathsf{R})$ (right) for $L=\{\varepsilon,a,a^2,ba\}$.

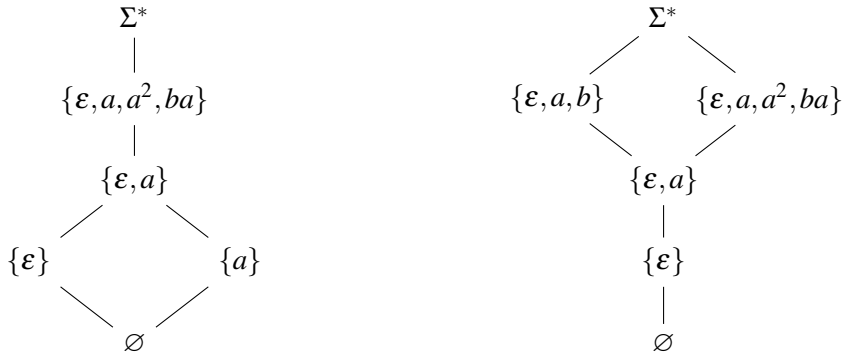


Figure 2: The lattices $\mathrm{Latt}(L,\cap,\mathsf{L})$ (left) and $\mathrm{Latt}(L,\cap,\mathsf{R})$ (right) for $L=\{\varepsilon,a,a^2,ba\}$.

in particular, the two lattices in Figure 1 are duals of each other and similarly for the two lattices in Figure 2.

We recall that two lattices Λ and Λ' are isomorphic (written $\Lambda\cong\Lambda'$) if there is a bijection $f:\Lambda\to\Lambda'$ such that $x<y$ in Λ if and only if $f(x)<f(y)$ in Λ' and such that $f(x\vee y)=f(x)\vee f(y)$ and $f(x\wedge y)=f(x)\wedge f(y)$ for all $x,y\in\Lambda$. Our main theorem shows that the duality occurring in Figures 1 and 2 is part of a general phenomenon.

Theorem 1. *Let $L\subseteq\Sigma^*$ be a regular language. Then we have:*

(a) $\mathrm{Latt}(L,\cup,\mathsf{L})$ *is isomorphic to the dual lattice of* $\mathrm{Latt}(L,\cup,\mathsf{R})$*;*

(b) $\mathrm{Latt}(L,\cap,\mathsf{L})$ *is isomorphic to the dual lattice of* $\mathrm{Latt}(L,\cap,\mathsf{R})$.

We note that the isomorphism given in Theorem 1(b), while not stated, can be obtained from the work of Im and Khovanov [5], if one carefully analyzes their constructions. In particular, it would also be interesting to know whether the isomorphism in Theorem 1(a) has any relevance to one-dimensional topological theories.

The outline of this paper is as follows. In §2 we present basic concepts needed from the theory of finite-state automata. In §3 we provide an overview of the theory of atoms of regular languages and in §4 we describe a key relationship between quotients and atoms. In §5 and §6 we give the proof of Theorem 1(a) and (b) respectively. In §7 we relate our results to Boolean semimodules and describe the duality algebraically. In §8 we present a brief analysis of when the lattices we construct are of maximal possible complexity, and §9 concludes the paper.

2 Automata and languages

A *nondeterministic finite automaton (NFA)* is a quintuple

$$\mathscr{N} = (Q, \Sigma, \delta, I, F),$$

where Q is a finite, non-empty set of *states*, Σ is a finite non-empty *alphabet*, $\delta : Q \times \Sigma \to 2^Q$ is the *transition function*, $I \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final states*. We can naturally extend the transition function to functions

$$\delta' : Q \times \Sigma^* \to 2^Q \qquad \delta'' : 2^Q \times \Sigma^* \to 2^Q,$$

which corresponds to taking elements of Σ^* as input for our automata and read them left-to-right to determine whether or not they are accepted; we henceforth use δ to denote all of these functions.

The *left language* of a state q of $\mathscr{N}$ is

$$\{w \in \Sigma^* : q \in \delta(I, w)\}, \tag{3}$$

and the *right language* of q is

$$\{w \in \Sigma^* : \delta(q, w) \cap F \neq \varnothing\}. \tag{4}$$

A state q of $\mathscr{N}$ is *reachable* if its left language is non-empty, and it is *empty* if its right language is empty. The *language accepted* by an NFA $\mathscr{N}$ is $L(\mathscr{N}) = \{w \in \Sigma^* : \delta(I, w) \cap F \neq \varnothing\}$, and we say that two NFAs are *equivalent* if they accept the same language. The *reverse* of an NFA $\mathscr{N} = (Q, \Sigma, \delta, I, F)$ is the NFA $\mathscr{N}^R = (Q, \Sigma, \delta^R, F, I)$, where $q \in \delta^R(p, a)$ if and only if $p \in \delta(q, a)$ for $p, q \in Q$ and $a \in \Sigma$. The reverse of an NFA $\mathscr{N}$ accepts the reverse of the language accepted by $\mathscr{N}$.

A *deterministic finite automaton (DFA)* is a quintuple $\mathscr{D} = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, and F are as in an NFA, $\delta : Q \times \Sigma \to Q$ is the transition function, and q_0 is the initial state.

We recall that a language L is regular if it is accepted by some DFA (or equivalently by an NFA). It is well known that the left quotients of the language L are precisely the right languages of the states of a minimal DFA for L. Any NFA $\mathscr{N}$ can be *determinized* by the well-known subset construction, yielding a DFA $\mathscr{N}^D$ that has only reachable states. We note that one can iteratively perform the reverse and determinization procedures; indeed, this plays a key role in the fundamental work of Brzozowski [2], and the following result is a slightly modified version of his work.

Proposition 2. *If an NFA $\mathscr{N}$ has no empty states and $\mathscr{N}^R$ is deterministic, then $\mathscr{N}^D$ is a minimal DFA.*

We note that by Proposition 2, for any NFA $\mathscr{N}$, the DFA $\mathscr{N}^{RDRD}$ is the minimal DFA equivalent to $\mathscr{N}$; this result is known as Brzozowski's double-reversal method for DFA minimization.

3 Atoms of a regular language

Let L be a non-empty regular language with left quotients $L_0, \ldots, L_{n-1}$. Given a subset $S \subseteq \{0, \ldots, n-1\}$ we can form a *left atomic intersection*

$$I_S := \left(\bigcap_{i \in S} L_i \right) \cap \left(\bigcap_{j \in \{0,\ldots,n-1\} \setminus S} \overline{L_j} \right), \tag{5}$$

where $\overline{L_i}$ is the complement of L_i in Σ^*.

A non-empty left atomic intersection is called a *left atom* of L [3].[2] A left atom is *initial* if it is contained in L and it is *final* if it contains the empty word ε. There is exactly one final left atom; namely the atom I_T where T is the set of i for which $\varepsilon \in L_i$.

If $\overline{L_0} \cap \cdots \cap \overline{L_{n-1}}$ is a left atom, then it is called the *negative* atom, with all other left atoms called *positive*. Thus left atoms of L are pairwise disjoint languages uniquely determined by L and they define a partition of Σ^*.

One can do a similar construction using right quotients: if $R_0, \ldots, R_{m-1}$ are the right quotients of L, then for each subset $T \subseteq \{0, \ldots, m-1\}$ we can form a right atomic intersection

$$J_T := \left(\bigcap_{i \in T} R_i \right) \cap \left(\bigcap_{j \in \{0,\ldots,m-1\} \setminus T} \overline{R_j} \right), \tag{6}$$

and we define right atoms of L to be the non-empty right atomic intersections.

We note that the left (resp., right) atoms of a language L are precisely the atoms of the Boolean algebra (regarded as a partially ordered set), generated by the left (resp., right) quotients of L.

As an example, if we take $L = \{\varepsilon, a, a^2, ba\}$ then the left atoms in this case are given by the partition

$$A_0 = \overline{\{\varepsilon, a, a^2, ba\}},\ A_1 = \{a^2, ba\},\ A_2 = \{a\},\ A_3 = \{\varepsilon\} \tag{7}$$

of Σ^*. These left atoms can be expressed as left atomic intersections as follows:

$$\overline{\{\varepsilon, a, a^2, ba\}} = \overline{\{\varepsilon, a, a^2, ba\}} \cap \overline{\{a\}} \cap \overline{\{\varepsilon, a\}} \cap \overline{\{\varepsilon\}} \cap \overline{\varnothing} = \overline{L_0} \cap \overline{L_1} \cap \overline{L_2} \cap \overline{L_3} \cap \overline{L_4}, \tag{8}$$

$$\{a^2, ba\} = \{\varepsilon, a, a^2, ba\} \cap \overline{\{a\}} \cap \overline{\{\varepsilon, a\}} \cap \overline{\{\varepsilon\}} \cap \overline{\varnothing} = L_0 \cap \overline{L_1} \cap \overline{L_2} \cap \overline{L_3} \cap \overline{L_4}, \tag{9}$$

$$\{a\} = \{\varepsilon, a, a^2, ba\} \cap \{a\} \cap \{\varepsilon, a\} \cap \overline{\{\varepsilon\}} \cap \overline{\varnothing} = L_0 \cap L_1 \cap L_2 \cap \overline{L_3} \cap \overline{L_4}, \tag{10}$$

and

$$\{\varepsilon\} = \{\varepsilon, a, a^2, ba\} \cap \overline{\{a\}} \cap \{\varepsilon, a\} \cap \{\varepsilon\} \cap \overline{\varnothing} = L_0 \cap \overline{L_1} \cap L_2 \cap L_3 \cap \overline{L_4}. \tag{11}$$

Here, the left atom $\overline{\{\varepsilon, a, a^2, ba\}}$ is negative, while the remaining left atoms are both positive and initial and the left atom $\{\varepsilon\}$ is the unique final atom.

On the other hand, the right atoms are given by the partition

$$B_0 = \{\varepsilon\},\ B_1 = \{b\},\ B_2 = \{a\},\ B_3 = \{ba, a^2\},\ B_4 = \overline{\{\varepsilon, a, a^2, b, ba\}}, \tag{12}$$

and they are obtained as right atomic intersections as

$$\{\varepsilon\} = \overline{\varnothing} \cap \{\varepsilon\} \cap \{\varepsilon, a, b\} \cap \{\varepsilon, a, a^2, ba\} = \overline{R_0} \cap R_1 \cap R_2 \cap R_3, \tag{13}$$

$$\{b\} = \overline{\varnothing} \cap \overline{\{\varepsilon\}} \cap \{\varepsilon, a, b\} \cap \overline{\{\varepsilon, a, a^2, ba\}} = \overline{R_0} \cap \overline{R_1} \cap R_2 \cap \overline{R_3}, \tag{14}$$

$$\{a\} = \overline{\varnothing} \cap \overline{\{\varepsilon\}} \cap \{\varepsilon, a, b\} \cap \{\varepsilon, a, a^2, ba\} = \overline{R_0} \cap \overline{R_1} \cap R_2 \cap R_3, \tag{15}$$

$$\{ba, a^2\} = \overline{\varnothing} \cap \overline{\{\varepsilon\}} \cap \overline{\{\varepsilon, a, b\}} \cap \{\varepsilon, a, a^2, ba\} = \overline{R_0} \cap \overline{R_1} \cap \overline{R_2} \cap R_3, \tag{16}$$

[2] In the literature, one generally just uses the term *atom* when speaking of what we call left atoms. However, to achieve our duality results it is convenient to use the adjective *left* when speaking of atoms obtained from left quotients.

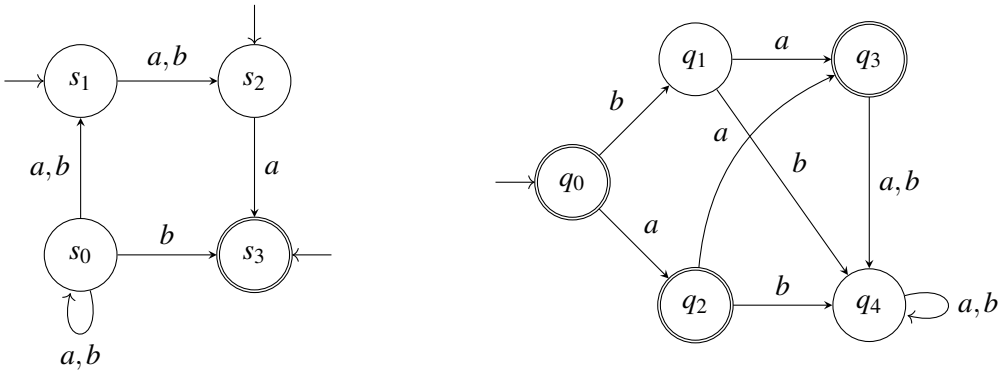


Figure 3: The átomaton (left) and the minimal DFA (right) for $L = \{\varepsilon, a, a^2, ba\}$.

and

$$\overline{\{\varepsilon, a, a^2, b, ba\}} = \overline{\varnothing} \cap \overline{\{\varepsilon\}} \cap \overline{\{\varepsilon, a, b\}} \cap \overline{\{\varepsilon, a, a^2, ba\}} = \overline{R_0} \cap \overline{R_1} \cap \overline{R_2} \cap \overline{R_3}. \tag{17}$$

We note that every left quotient of L (including L itself) is a (possibly empty) union of left atoms and similarly every right quotient is a union of right atoms.

It is well known that left quotients of L are in a one-to-one correspondence with the equivalence classes of the *Nerode right congruence* $\equiv_L$ of L [8] defined as follows: for $x, y \in \Sigma^*$, $x \equiv_L y$ if for every $v \in \Sigma^*$, $xv \in L$ if and only if $yv \in L$. Left atoms of L are the classes of the *left congruence* ${}_L\equiv$ of L: for $x, y \in \Sigma^*$, $x \, {}_L\equiv y$ if for every $u \in \Sigma^*$, $ux \in L$ if and only if $uy \in L$ [6]. Also, right quotients are in a one-to-one correspondence with the equivalence classes of the left congruence.

Let $A_0, \ldots, A_{m-1}$ denote the left atoms of L where we index so that A_{m-1} is the final atom, and let I denote the set of initial atoms.

The *átomaton* $\mathscr{A}$ of L is the NFA whose set of states is the set

$$S = \{s_0, \ldots, s_{m-1}\}, \tag{18}$$

which can be thought of as parameterizing the set of left atoms of L. More precisely, we take

$$\mathscr{A} = (S, \Sigma, \alpha, I, \{s_{m-1}\}),$$

where $s_j \in \alpha(s_i, a)$ if and only if $A_j \subseteq a^{-1}A_i$, for $i, j \in \{0, \ldots, m-1\}$ and $a \in \Sigma$. (We refer the reader to [3] for further details on átomata.)

In the running example in which we take $L = \{\varepsilon, a, a^2, ba\}$, by Equation (7), the left atoms are the sets

$$A_0 = \overline{\{\varepsilon, a, a^2, ba\}},\ A_1 = \{a^2, ba\},\ A_2 = \{a\},\ A_3 = \{\varepsilon\},$$

and we see that the átomaton associated to L is given in Figure 3 on the left, where the states s_1, s_2, s_3 are initial.

Observe that if we adopt the labelling given in Equations (2) and (7), then the right languages of the átomaton in Figure 3 are

$$A_0 = \{a, b\}^* \cdot \big(\{b\} \cup \{a, b\}^2 \cdot \{a\}\big) = \overline{\{\varepsilon, a, a^2, ba\}}$$

(for the state s_0), $A_1 = \{a^2, ba\}$ (for the state s_1), $A_2 = \{a\}$ (for the state s_2), and $A_3 = \{\varepsilon\}$ (for the state s_3), which are precisely the left atoms of the language given in Equation (7).

On the other hand, the left languages are $R_0 = \varnothing$ (for the state s_0), $R_1 = \{\varepsilon\}$ (for the state s_1), $R_2 = \{\varepsilon, a, b\}$ (for the state s_2), $R_3 = \{\varepsilon, a, a^2, ba\}$ (for the state s_3), and these are precisely the right quotients of L, as given in Equation (2).

In fact, these observations are part of general phenomena, as shown by Brzozowski and Tamm [3], which we record in the following proposition.

Proposition 3. *Let L be a non-empty regular language. Then the following hold:*

(i) the left quotients of L are precisely the right languages of the minimal DFA accepting L;

(ii) the left atoms of L are precisely the right languages of the átomaton $\mathscr{A}$ associated to L;

(iii) the right quotients of L are precisely the left languages of $\mathscr{A}$;

(iv) the right atoms of L are precisely the left languages of the minimal DFA accepting L.

In particular, we have set bijections

$$\{\textit{left atoms of } L\} \leftrightarrow \{\textit{right quotients of } L\}$$

and

$$\{\textit{right atoms of } L\} \leftrightarrow \{\textit{left quotients of } L\},$$

where in the first case we view a left atom of L as the right language of a state of $\mathscr{A}$ and then send it to the left language of this state and in the second case we view a right atom of L as the left language of a state of the minimal DFA of L and then send it to the right language of this state.

Proof. Item (*i*) is well known. It was shown in [3] that the left atoms of a regular language L are precisely the right languages of the states of the associated átomaton, so (*ii*) holds.

A modification of the isomorphism result from [3] shows that if $\mathscr{D}$ is the minimal DFA accepting L with state set $Q = \{q_0, q_1, \ldots, q_{n-1}\}$, then the átomaton, $\mathscr{A}$, associated to L is isomorphic to $\mathscr{D}^{RDR}$ as NFAs, via an isomorphism induced by the map which sends a state $s_i \in S$ from the state set of $\mathscr{A}$ to the set $\{q_j : j \in S\}$, where $S \subseteq \{0, \ldots, n-1\}$ has the property that A_i is the left atomic intersection I_S. Since by Proposition 2, the DFA $\mathscr{D}^{RD}$ is the minimal DFA of the reverse language of L, the left languages of $\mathscr{D}^{RDR} \cong \mathscr{A}$ are exactly the right quotients of L, which establishes (*iii*).

Finally, [3] shows that the reverse NFA of the átomaton of L is the minimal DFA of the reverse language of L, and so (*iv*) now follows, and the bijections are immediate from (*i*)–(*iv*). □

We again consider the regular language $L = \{\varepsilon, a, a^2, ba\}$ as an example. Then the automaton in Figure 3 on the right is the minimal DFA accepting L with the state set $\{q_0, q_1, q_2, q_3, q_4\}$.

Observe that for this DFA, if we adopt the labellings from Equations (1) and (12), the left language of q_0 is $B_0 = \{\varepsilon\}$ and the right language is $L_0 = L$; the left language of q_1 is the right atom $B_1 = \{b\}$ and the right language is the left quotient $L_1 = \{a\}$; the left language of q_2 is $B_2 = \{a\}$ and the right language is $L_2 = \{\varepsilon, a\}$; the left language of q_3 is $B_3 = \{ba, a^2\}$ and the right language is $L_3 = \{\varepsilon\}$; and finally the left language of q_4 is $B_4 = (\{ab, b^2\} \cup \{a^2, ba\}\{a, b\})\{a, b\}^* = \{\varepsilon, a, b, a^2, ba\}$ and the right language is $L_4 = \varnothing$. Similarly, the remarks preceding Proposition 3 give the bijection between left atoms and right quotients. We record these bijections in Figure 4, where $\mathscr{A}$ is the átomaton and $\mathscr{D}$ is the DFA from Figure 3.

State of $\mathscr{D}$	Left quotient of $\{\varepsilon, a, a^2, ba\}$	Right atom of $\{\varepsilon, a, a^2, ba\}$
q_0	$\{\varepsilon, a, a^2, ba\}$	$\{\varepsilon\}$
q_1	$\{a\}$	$\{b\}$
q_2	$\{\varepsilon, a\}$	$\{a\}$
q_3	$\{\varepsilon\}$	$\{ba, a^2\}$
q_4	$\varnothing$	$\overline{\{\varepsilon, a, b, a^2, ba\}}$

State of $\mathscr{A}$	Right quotient of $\{\varepsilon, a, a^2, ba\}$	Left atom of $\{\varepsilon, a, a^2, ba\}$
s_0	$\varnothing$	$\overline{\{\varepsilon, a, a^2, ba\}}$
s_1	$\{\varepsilon\}$	$\{a^2, ba\}$
s_2	$\{\varepsilon, a, b\}$	$\{a\}$
s_3	$\{\varepsilon, a, a^2, ba\}$	$\{\varepsilon\}$

Figure 4: Tables giving the bijections described in Proposition 3 between left quotients and right atoms and between right quotients and left atoms for the language $L = \{\varepsilon, a, a^2, ba\}$.

4 Relationships between quotients and atoms

In this section, we give key bijections between left quotients and right atoms and similarly for right quotients and left atoms.

We find it convenient to introduce notation that we will use in proving Theorem 1. The main aim of this notation is to capture the isomorphisms described in Proposition 3 and we henceforth adopt this notation in all results we prove.

Notation 4. *We introduce the following notation.*

(i) We let L be a non-empty regular language in Σ^* with Σ a finite alphabet.

(ii) We let $\mathscr{A}$ denote the átomaton of L and let $\mathscr{D}$ denote the minimal DFA accepting L on states $q_0, \ldots, q_{n-1}$.

(iii) We let $L_0, \ldots, L_{n-1}$ denote the left quotients of L.

(iv) We let $R_0, \ldots, R_{m-1}$ denote the right quotients of L.

(v) We let $A_0, \ldots, A_{m-1}$ denote the left atoms of L, where we index so that A_i corresponds to R_i under the bijection given in Proposition 3.

(vi) We let $B_0, \ldots, B_{n-1}$ be the right atoms of L, where we index so that B_i corresponds to L_i under the bijection given in Proposition 3.

Remark 5. *We note that in our running example where $L = \{\varepsilon, a, a^2, ba\}$, this notation is consistent with the labellings given in Equations (1), (2), (7), and (12), as shown by Figure 4.*

The following proposition gives a precise relationship between the left and right quotients of a regular language L and the left and right atoms of L.

Proposition 6. *Let $i \in \{0,\dots,m-1\}$, and let $j \in \{0,\dots,n-1\}$. Then*

$$R_i = \bigcup_{\{k:A_i \subseteq L_k\}} B_k \qquad \text{and} \qquad L_j = \bigcup_{\{\ell:B_j \subseteq R_\ell\}} A_\ell.$$

In particular, $A_i \subseteq L_j$ if and only if $B_j \subseteq R_i$.

Proof. As noted in the proof of Proposition 3, the modified argument of [3] shows that the NFAs $\mathscr{A}$ and $\mathscr{D}^{RDR}$ are isomorphic, with a state s_i in $\mathscr{A}$ corresponding to a set $\{q_i : i \in S\}$ for some set $S \subseteq \{0,\dots,n-1\}$ with the property that the left atom A_i is the (left) atomic intersection I_S described in Equation (5). By Proposition 3, the right quotients of L are the left languages of $\mathscr{D}^{RDR}$ and the right atoms of L are the left languages of $\mathscr{D}$. Hence, it is clear that the first equality holds.

The second equality is proved analogously, now using that the left quotients of L are the right languages of $\mathscr{D}$ and the left atoms of L are the right languages of $\mathscr{A} \cong \mathscr{D}^{RDR}$ by Proposition 3. The "in particular" clause follows immediately from these equalities. □

Lemma 7. *Let X be a union of left atoms of L, and let Y be a union of right atoms of L. Then we have:*

(1) $\bigcup_{\{i:A_i \not\subseteq X\}} R_i = \bigcup_{\{j:L_j \not\subseteq X\}} B_j$,

(2) $\bigcap_{\{j:A_j \subseteq X\}} R_j = \bigcup_{\{i:X \subseteq L_i\}} B_i$,

(3) $\bigcup_{\{i:B_i \not\subseteq Y\}} L_i = \bigcup_{\{j:R_j \not\subseteq Y\}} A_j$,

(4) $\bigcap_{\{j:B_j \subseteq Y\}} L_j = \bigcup_{\{i:Y \subseteq R_i\}} A_i$.

Proof. We consider the union of the right quotients $U = \bigcup_{A_i \not\subseteq X} R_i$, corresponding to the left atoms not contained in X.

Consider a left quotient L_j that is not contained in X. Then there is some left atom A_i such that $A_i \subseteq L_j$ and $A_i \not\subseteq X$. By Proposition 6, $A_i \subseteq L_j$ gives that $B_j \subseteq R_i$, and hence U contains all right atoms B_j such that L_j is not a subset of X, and so U contains $\bigcup_{\{j:L_j \not\subseteq X\}} B_j$.

On the other hand, if $L_j \subseteq X$ and $A_i \not\subseteq X$, then $A_i \not\subseteq L_j$, which by Proposition 6 gives that $B_j \not\subseteq R_i$. Hence, $B_j \not\subseteq U$ if $L_j \subseteq X$, and so we get the reverse containment, establishing (1).

By Proposition 6 we see that for each left quotient L_i, the inclusion $X \subseteq L_i$ holds if and only if $B_i \subseteq \bigcap_{\{j:A_j \subseteq X\}} R_j$ holds and so we obtain (2).

The proofs of (3) and (4) are done similarly to (1) and (2). □

A convenient tool for capturing much of this information comes from the *quotient-atom* matrix [7, 9]. If we adopt the notation of Notation 4, then this matrix is the $n \times m$ zero-one matrix whose (i,j)-entry (where we start our indices at zero) is 1 exactly when $i \in S$, where $S \subseteq \{0,1,\dots,n-1\}$ is the set giving the left atom A_j as a left atomic intersection I_S. Equivalently, this is the case when $A_j \subseteq L_i$.

In the case that L is the regular language $\{\varepsilon, a, a^2, ba\}$, the left quotients and left atoms are given in Equations (1) and (7), and the expressions for left atoms as left atomic intersections are given in Equations (8)–(11). Using these data, we see that the quotient-atom matrix for $L = \{\varepsilon, a, a^2, ba\}$ is given in Figure 5.

We note that one can do an analogous construction with right atoms and right quotients and one will then obtain the transpose of the quotient-atom matrix. The quotient-atom matrix allows one to understand non-empty intersections of non-empty sets of left and right quotients in terms of maximal grids of the quotient-atom matrix [7, 9].

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{19}$$

Figure 5: The quotient-atom matrix for $L = \{\varepsilon, a, a^2, ba\}$.

5 The isomorphism $\mathrm{Latt}(L, \cup, \mathsf{L}) \cong \mathrm{Latt}(L, \cup, \mathsf{R})^*$

In this section, we give the proof of Theorem 1(a).

We define a set map

$$\Psi : \mathrm{Latt}(L, \cup, \mathsf{L}) \to \mathrm{Latt}(L, \cup, \mathsf{R}) \tag{20}$$

by declaring that for $X \in \mathrm{Latt}(L, \cup, \mathsf{L})$,

$$\Psi(X) := \bigcup_{\{i : A_i \not\subseteq X\}} R_i. \tag{21}$$

We can similarly define a map

$$\Psi' : \mathrm{Latt}(L, \cup, \mathsf{R}) \to \mathrm{Latt}(L, \cup, \mathsf{L}), \tag{22}$$

where for $Y \in \mathrm{Latt}(L, \cup, \mathsf{R})$, we define

$$\Psi'(Y) := \bigcup_{\{i : B_i \not\subseteq Y\}} L_i. \tag{23}$$

We shall show that the maps Ψ and Ψ' are inverses of each other and that Ψ induces a lattice isomorphism between $\mathrm{Latt}(L, \cup, \mathsf{L})$ and $\mathrm{Latt}(L, \cup, \mathsf{R})^*$.

To continue with the example when $L = \{\varepsilon, a, a^2, ba\}$, it can be checked that the map Ψ is defined by the assignments $\Psi(\varnothing) = \{\varepsilon, a, a^2, b, ba\}$, $\Psi(\{\varepsilon\}) = \{\varepsilon, a, b\}$, $\Psi(\{a\}) = \{\varepsilon, a, a^2, ba\}$, $\Psi(\{\varepsilon, a\}) = \{\varepsilon\}$, and $\Psi(\{\varepsilon, a, a^2, ba\}) = \varnothing$, which is capturing the dual structure of the lattices in Figure 1.

Lemma 8. *Let Ψ and Ψ' be the maps defined in Equations (21) and (23). Then Ψ and Ψ' are set-theoretic inverses of each other.*

Proof. Let $X \in \mathrm{Latt}(L, \cup, \mathsf{L})$. Then $Y := \Psi(X)$ is the union of all R_i such that $A_i \not\subseteq X$. Using Lemma 7 we then see

$$\Psi(X) = \bigcup_{\{j : L_j \not\subseteq X\}} B_j. \tag{24}$$

Then $\Psi'(Y) = \bigcup_{B_k \not\subseteq Y} L_k$. Since right atoms are disjoint, from Equation (24) we see that B_j is not a subset of Y if and only if $L_j \subseteq X$. Thus $\Psi'(Y)$ is the union of all left quotients L_j contained in X, which is precisely X as X is a union of left quotients.

The fact that $\Psi \circ \Psi'$ is the identity of $\mathrm{Latt}(L, \cup, \mathsf{R})$ is proved with a symmetric argument, again using Lemma 7. □

Lemma 9. *Let Ψ and Ψ' be the maps defined in Equations (21) and (23). If U_1 and U_2 are unions of left quotients of L, then the following hold:*

(1) $U_1 \subseteq U_2 \iff \Psi(U_2) \subseteq \Psi(U_1)$*;*

(2) $\Psi(U_1 \cup U_2)$ *is the largest union of right quotients that is contained in the intersection* $\Psi(U_1) \cap \Psi(U_2)$*;*

(3) if V *is the largest union of left quotients contained in* $U_1 \cap U_2$*, then* $\Psi(V) = \Psi(U_1) \cup \Psi(U_2)$.

Proof. It is immediate from the definition that if $U_1 \subseteq U_2$ then $\Psi(U_2) \subseteq \Psi(U_1)$. Similarly, if V_1 and V_2 are unions of right quotients of L, then if $V_2 \subseteq V_1$ then $\Psi'(V_1) \subseteq \Psi'(V_2)$. Taking $V_i = \Psi(U_i)$ for $i = 1, 2$, by Lemma 8, if $\Psi(U_2) \subseteq \Psi(U_1)$ then $U_1 \subseteq U_2$, which establishes (1).

To see (2), observe that since Ψ reverses inclusions, we have $\Psi(U_1 \cup U_2) \subseteq \Psi(U_1) \cap \Psi(U_2)$. Now suppose that R_i is a right quotient that is contained in $\Psi(U_1) \cap \Psi(U_2)$. Then since Ψ reverses inclusions and Ψ' is the inverse of Ψ, we have that Ψ' also reverses inclusions and so $\Psi'(R_i) \supseteq U_1$ since $R_i \subseteq \Psi(U_1)$, and similarly $\Psi'(R_i) \supseteq U_2$. Hence $\Psi'(R_i) \supseteq U_1 \cup U_2$ and so applying Ψ and using once more that it reverses inclusions, we see that R_i is contained in $\Psi(U_1 \cup U_2)$. Thus $\Psi(U_1 \cup U_2)$ is the largest union of right quotients contained in $\Psi(U_1) \cap \Psi(U_2)$, which shows (2).

We now prove (3). Let V be the union of all left quotients contained in $U_1 \cap U_2$. Then since Ψ reverses inclusions, we have $\Psi(V) \supseteq \Psi(U_1)$ and similarly $\Psi(V) \supseteq \Psi(U_2)$, which shows that $\Psi(V) \supseteq \Psi(U_1) \cup \Psi(U_2)$. To show equality, notice that if $\Psi(V)$ strictly contains $\Psi(U_1) \cup \Psi(U_2)$, then there is some right atom B_k contained in $\Psi(V)$ that is neither contained in $\Psi(U_1)$ nor in $\Psi(U_2)$. Then since $B_k \subseteq \Psi(V)$, we have $L_k \not\subseteq V$ by Equation (21) and Lemma 7. But the fact that B_k is not contained in $\Psi(U_1)$ gives that $L_k \subseteq U_1$ and similarly $L_k \subseteq U_2$. Hence $L_k \subseteq U_1 \cap U_2$. But this contradicts the fact that we chose V to be the union of left quotients contained in $U_1 \cap U_2$. Thus we get (3). □

Proof of Theorem 1(a). The fact that Ψ gives a poset isomorphism between the lattice $\mathrm{Latt}(L, \cup, \mathsf{L})$ and the dual lattice $\mathrm{Latt}(L, \cup, \mathsf{R})^*$ follows from Lemmas 8 and 9 (1). Lemma 9 (2) and (3) show that Ψ preserves respectively the meet and join operations on these posets, as described in the definitions. □

6 The isomorphism $\mathrm{Latt}(L, \cap, \mathsf{L}) \cong \mathrm{Latt}(L, \cap, \mathsf{R})^*$

The aim of this section is to prove Theorem 1(b) involving intersections of left and right quotients.

We now define maps

$$\Phi : \mathrm{Latt}(L, \cap, \mathsf{L}) \to \mathrm{Latt}(L, \cap, \mathsf{R}) \tag{25}$$

and

$$\Phi' : \mathrm{Latt}(L, \cap, \mathsf{R}) \to \mathrm{Latt}(L, \cap, \mathsf{L}) \tag{26}$$

as follows. If $X \in \mathrm{Latt}(L, \cap, \mathsf{L})$, we define

$$\Phi(X) = \bigcap_{A_j \subseteq X} R_j, \tag{27}$$

and if Y is an intersection of right quotients of L, we define

$$\Phi'(Y) = \bigcap_{B_j \subseteq Y} L_j. \tag{28}$$

The following lemmas can be proved in a similar manner to the method of proof for Lemmas 8 and 9.

Lemma 10. *Let* Φ *and* Φ' *be the maps defined in Equations (27) and (28). Then* Φ *and* Φ' *are set-theoretic inverses of each other.*

Lemma 11. *Let Φ and Φ' be the maps defined in Equations (27) and (28). If U_1 and U_2 are intersections of left quotients of L, then the following hold:*

(1) $U_1 \subseteq U_2 \iff \Phi(U_2) \subseteq \Phi(U_1)$*;*

(2) $\Phi(U_1 \cap U_2)$ *is the smallest intersection of right quotients that contains the union* $\Phi(U_1) \cup \Phi(U_2)$*;*

(3) if V is the smallest intersection of left quotients that contains $U_1 \cup U_2$, then $\Phi(V) = \Phi(U_1) \cap \Phi(U_2)$.

Proof of Theorem 1(b). This is proved similarly to Theorem 1(a), but where we now use Lemmas 10 and 11. □

7 Semimodules and semilattices

In this section, we reinterpret our results algebraically and note connections with work of Im and Khovanov [5].

Let $\mathbb{B}$ denote the Boolean semiring, which is the set $\{0,1\}$ endowed with binary operations $+$ and $\cdot$ as in the tables from Figure 6.

$+$	0	1
0	0	1
1	1	1

$\cdot$	0	1
0	0	0
1	0	1

Figure 6: Addition and multiplication tables for the Boolean ring $\mathbb{B}$.

A Boolean semimodule M is a commutative monoid (written additively and with an identity element 0_M) equipped with a scalar multiplication map

$$\cdot : \mathbb{B} \times M \to M$$

satisfying

$1 \cdot m = m$ for all $m \in M$, $0 \cdot m = 0_M$ for all $m \in M$ and $b \cdot (m+n) = b \cdot m + b \cdot n$ and $(b+c) \cdot m = b \cdot m + c \cdot m$ for all $b, c \in \mathbb{B}$ and all $m, n \in M$.

In particular, if M is a Boolean semimodule then for $m \in M$ we have $m + m = (1+1) \cdot m = 1 \cdot m = m$, and so all elements of M are idempotent. A Boolean semimodule can be viewed as a join-semilattice (that is a partially ordered set in which any two elements have a least upper bound) as follows. Given a Boolean semimodule M we can define a partial order $\leqslant$ on M by declaring that $m \leqslant n$ whenever $m + n = n$. We can then define a join operation on M by declaring that $m \vee n := m + n$. It is straightforward to check that this gives M the structure of a join semilattice. Conversely, given a join semilattice Λ with a least element m_0, one can endow Λ with the structure of a Boolean semimodule by taking the join operation to be addition and taking m_0 to be the zero element. In case M is a finite semimodule, M is in fact a lattice with meet defined by taking $m \wedge n$ to be the join of all elements q that are less than or equal to both m and n, and with unique maximal element given by taking the join of all elements of the lattice.

Given a Boolean semimodule M, one has a dual module $M^* = \mathrm{Hom}_{\mathbb{B}}(M, \mathbb{B})$, where $\mathrm{Hom}_{\mathbb{B}}(M, \mathbb{B})$ is the set of $\mathbb{B}$-linear maps from M to $\mathbb{B}$. We observe that M^* is itself a Boolean semimodule, since we can add maps and have a zero map. We then have a natural $\mathbb{B}$-bilinear pairing $\langle , \rangle : M \times M^* \to \mathbb{B}$ given by $\langle m, f \rangle = f(m)$ for $m \in M$ and $f \in M^*$. For a finite Boolean semimodule M, viewed as a semilattice, M^* is just the dual semilattice of M.

We note that for a finite alphabet Σ, we can construct the Boolean lattice $\mathrm{Bool}(\Sigma) := 2^{\Sigma^*}$, consisting of subsets of Σ^* partially ordered by inclusion and where meet and join are given by intersection and union respectively. Then given a regular language $L \subseteq \Sigma^*$, we have a $\mathbb{B}$-bilinear map, which we call the *Im-Khovanov pairing* with respect to L,

$$\langle\ ,\ \rangle_L : \mathrm{Bool}(\Sigma) \times \mathrm{Bool}(\Sigma) \to \mathbb{B}$$

defined by

$$\langle A, B\rangle_L = \begin{cases} 1 & \text{if there exists } w \in A, v \in B \text{ such that } wv \in L; \\ 0 & \text{otherwise,} \end{cases} \tag{29}$$

for $A, B \subseteq \Sigma^*$. From its definition, this is easily seen to be $\mathbb{B}$-bilinear and this pairing appears in the work of Im and Khovanov [5, §4].

For the remainder of this section, we adopt the notation of Notation 4 and let $\langle\ ,\ \rangle$ denote the Im-Khovanov pairing with respect to L. Then by Proposition 3, B_i is the left language of a state q_i of the minimal DFA accepting L, and L_i is the corresponding right language of q_i. Therefore, $\langle B_i, A_j\rangle = 1$ if and only if $A_j \cap L_i \neq \varnothing$. Similarly, using the átomaton of L, we obtain that the property $\langle B_i, A_j\rangle = 1$ is equivalent to $B_i \cap R_j \neq \varnothing$. On the other hand, left quotients are unions of left atoms and left atoms are disjoint, and so $A_j \cap L_i$ is non-empty if and only if $A_j \subseteq L_i$, and we have an analogous fact for right quotients and right atoms. Hence, we have the equivalences

$$\langle B_i, A_j\rangle = 1 \iff B_i \subseteq R_j \iff A_j \subseteq L_i, \tag{30}$$

which can be thought of as an algebraic reformulation of Proposition 6. In general, if X is a union of left atoms, then we have $\langle B_i, X\rangle = 0 \iff X \cap L_i = \varnothing$, and if Z is a union of right atoms, we have $\langle Z, A_j\rangle = 0 \iff Z \cap R_j = \varnothing$.

The pairing $\langle\ ,\ \rangle$ restricts to pairings

$$\langle\ ,\ \rangle : \mathrm{Latt}(L, \cup, \mathsf{R}) \times \mathrm{Latt}(L, \cup, \mathsf{L}) \to \mathbb{B}$$

and

$$\langle\ ,\ \rangle : \mathrm{Latt}(L, \cap, \mathsf{R}) \times \mathrm{Latt}(L, \cap, \mathsf{L}) \to \mathbb{B}.$$

We now give a description of the maps Ψ and Φ from Equations (21) and (27) in terms of the Im-Khovanov pairing. In order to express this, for a subset Y of Σ^*, we let $Y^\perp$ denote the *orthogonal complement* of Y, which is the subset of Σ^* consisting of words w with the property that $\langle Y, \{w\}\rangle = 0$.

Proposition 12. *Let Ψ and Φ be the maps given in Equations (21) and (27). Then we have the following:*

(1) for X a union of left quotients, $\Psi(X) = \bigcup_{\{k : \langle B_k, \overline{X}\rangle = 1\}} B_k$;

(2) for Z an intersection of left quotients, $\Phi(Z) = \bigcup_{\{k\ :\ Z \cap B_k^\perp = \varnothing\}} B_k$.

Proof. Let X be a union of left quotients. Then by Equation (21),

$$\Psi(X) = \bigcup_{\{j : A_j \not\subseteq X\}} R_j.$$

Since each right quotient is a union of right atoms, and since right atoms are disjoint, we see that $\Psi(X)$ is uniquely expressible as a union of right atoms. Then $B_k \subseteq \Psi(X)$ if and only if there is some j such that $A_j \not\subseteq X$ and $B_k \subseteq R_j$. Notice that since X is a union of left atoms, $A_j \not\subseteq X$ if and only if $A_j \subseteq \overline{X}$,

and so we see by Equation (30) that $B_k \subseteq \Psi(X)$ if and only if there is an index j such that $A_j \subseteq \overline{X}$ and $\langle B_k, A_j \rangle = 1$. Finally, bilinearity of our pairing says that

$$B_k \subseteq \Psi(X) \iff \langle B_k, \overline{X} \rangle = 1.$$

This completes the proof of (1).

Next let Z be an intersection of left quotients. Then by Equation (27) we have

$$\Phi(Z) = \bigcap_{\{j : A_j \subseteq Z\}} R_j.$$

Notice that since right atoms are disjoint and since each right quotient is a union of right atoms, $B_k \subseteq \Phi(Z)$ if and only if $B_k \subseteq R_j$ for all j such that $A_j \subseteq Z$. Again, by Equation (30), this is equivalent to $\langle B_k, A_j \rangle = 1$ for all j such that $A_j \subseteq Z$. Notice that $\langle B_k, A_j \rangle = 1$ if and only if A_j is completely contained in L_k, and hence if $\langle B_k, A_j \rangle = 1$, then $\langle B_k, Y \rangle = 1$ for all non-empty subsets Y of A_j. Hence this is equivalent to saying that Z does not intersect the orthogonal complement of B_k, and so the result follows. □

One can also interpret the quotient-atom matrix in terms of the Im-Khovanov pairing, if one views the entries of the matrix as living in the Boolean semiring $\mathbb{B}$. For the quotient-atom matrix, the (i, j)-entry is 1 if L_i appears in the atomic intersection giving A_j. Equivalently, the (i, j)-entry is 1 precisely when $A_j \subseteq L_i$, which by Equation (30) occurs precisely when $\langle B_i, A_j \rangle = 1$. In particular, we have the following reinterpretation of the quotient-atom matrix.

Proposition 13. *The quotient-atom matrix is the $n \times m$ matrix whose (i, j)-entry is $\langle B_i, A_j \rangle$.*

8 Complexity

In this section, we look at when the lattices we construct can be in some sense as large as possible.

If we adopt the notation of Notation 4, then there are at most 2^n unions of left quotients and at most 2^m unions of right quotients of L. By Theorem 1, the number of unions of left quotients is equal to the number of unions of right quotients, and hence there are at most $2^{\min(m,n)}$ unions of left/right quotients of L.

It is also not difficult to see that if L has $2^n - 1$ positive atoms—that is, all possible positive atoms exist—then there are 2^n unions of left quotients. We show, however, to realize this maximal complexity, only n left atoms of L are required.

Proposition 14. *There are 2^n unions of left quotients of L if and only if all the left atomic intersections with one uncomplemented and $n-1$ complemented left quotients are non-empty.*

Proof. Let us suppose that all the left atomic intersections with one uncomplemented and $n-1$ complemented left quotients of L are non-empty. That is, for every $i \in \{0, \ldots, n-1\}$, the left atomic intersection $I_{\{i\}}$ is non-empty. Hence, for each left quotient L_i, there is at least one atom, namely $I_{\{i\}}$, contained in L_i and not contained in any other left quotient. Since the left atoms are pairwise disjoint, this implies that there are 2^n distinct unions of left quotients of L.

Conversely, if $I_{\{i\}}$ is empty for some i, then it is easily checked that

$$\bigcup_{j \neq i} L_j = \bigcup_j L_j$$

and so the number of unions of left quotients of L is strictly less than 2^n. □

A similar result can be achieved for the complexity of intersections of left quotients of L.

Proposition 15. *There are 2^n intersections of left quotients of L if and only if all the left atomic intersections with $n-1$ uncomplemented and one complemented left quotients are non-empty.*

Proof. First, let us assume that for every $k \in \{0,\dots,n-1\}$, $Z_k := I_{\{0,\dots,n-1\}\setminus\{k\}}$ is non-empty.

Now, consider any intersection of left quotients $X = L_{i_1} \cap \cdots \cap L_{i_s}$. Then one can verify that $Z_k \subseteq X$ if and only if $k \notin \{i_1,\dots,i_s\}$ for $k \in \{0,\dots,n-1\}$. Thus by checking which of the left atoms $Z_0,\dots,Z_{n-1}$ are subsets of an intersection of left quotients, we can uniquely recover the left quotients appearing in the intersection and so we obtain 2^n distinct intersections.

Conversely, suppose that for some k, the intersection Z_k is empty. Then $\bigcap_{j\neq k} L_j$ has empty intersection with $\overline{L_k}$ and thus is contained in L_k. Hence

$$\bigcap_{j\neq k} L_j = \bigcap_j L_j,$$

and so the number of intersections of left quotients of L is strictly smaller than 2^n. □

We get the following result as an immediate consequence of Propositions 14 and 15.

Corollary 16. *Let $n > 2$. Then the $2n$ atoms of L, described in Propositions 14 and 15, are necessary and sufficient to obtain the equalities*

$$|\mathrm{Latt}(L,\cup,\mathsf{L})| = |\mathrm{Latt}(L,\cap,\mathsf{L})| = 2^n.$$

9 Conclusions and further work

Corollary 16 gives an efficient means for checking that the lattices we obtain are of maximal possible complexity. It would be interesting to know whether other lattice-theoretic properties for the lattices we consider can be efficiently checked or even characterized in terms of the associated automata. Of particular interest is the question of when our lattices are distributive. In the framework considered by Im and Khovanov [5], the distributive property is key for associating topological quantum field theories to regular languages.

References

[1] J. Brzozowski, Towards a theory of complexity of regular languages. *J. Autom. Lang. Comb.* **23** (2018), no. 1–3, pp. 67–101, doi:10.25596/jalc-2018-067.

[2] J. Brzozowski, Canonical regular expressions and minimal state graphs for definite events. *Proc. Sympos. Math. Theory of Automata* (New York, 1962), Polytechnic Press of the Polytechnic Inst. of Brooklyn, Brooklyn, N.Y., pp. 529–561, 1963.

[3] J. Brzozowski and H. Tamm, Theory of átomata. *Theoret. Comput. Sci.* **539** (2014), pp. 13–27, doi:10.1016/j.tcs.2014.04.016.

[4] N. Chomsky and G. A. Miller, Finite state languages. *Information and Control* **1** (1958), pp. 91–112, doi:10.1016/S0019-9958(58)90082-2.

[5] M. S. Im and M. Khovanov, *Topological theories and automata*, doi:10.48550/arXiv.2202.13398, arXiv:2202.13398.

[6] S. Iván, Complexity of atoms, combinatorially. *Inform. Process. Lett.* **116** (2016), no. 5, pp. 356–360, doi:10.1016/j.ipl.2016.01.003.

[7] T. Kameda and P. Weiner, On the state minimization of nondeterministic finite automata. *IEEE Trans. Comput. C-19* (1970), no. 7, pp. 617–627, doi:10.1109/T-C.1970.222994.

[8] A. Nerode, Linear automaton transformations. *Proc. Amer. Math. Soc.* **9** (1958), pp. 541–544, doi:10.1090/S0002-9939-1958-0135681-9.

[9] H. Tamm, New interpretation and generalization of the Kameda-Weiner method. *43rd International Colloquium on Automata, Languages, and Programming*, Art. No. 116, 12 pp., LIPIcs. Leibniz Int. Proc. Inform., 55, Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016, doi:10.4230/LIPIcs.ICALP.2016.116.

Approximate State Reduction of Fuzzy Finite Automata*

Miroslav Ćirić Ivana Micić Stefan Stanimirović
University of Niš, Faculty of Sciences and Mathematics, Višegradska 33, Niš, Serbia
miroslav.ciric@pmf.edu.rs, ivana.micic@pmf.edu.rs, stefan.stanimirovic@pmf.edu.rs

Linh Anh Nguyen
Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland, and
Faculty of Information Technology, Nguyen Tat Thanh University, Ho Chi Minh City, Viet Nam
nguyen@mimuw.edu.pl

In this paper we introduce a new type of approximate state reductions where the behaviors of the reduced and the original automaton do not have to be identical, but they must match on all words of length less than or equal to some given natural number. We provide four methods for performing such reductions.

1 Introduction

Minimization and state reduction are related problems that belong to the fundamental problems of automata theory and have many significant applications. Minimization is the problem of finding an automaton with a minimal number of states equivalent to a given automaton. However, this problem cannot be solved efficiently (in polynomial time) for fuzzy finite or nondeterministic finite automata as their particular type (cf. [7] for more details). Therefore, the so-called state reduction problem for fuzzy finite automata is studied instead, where the goal is to find an automaton equivalent to the given automaton that is not necessarily minimal, but that can be treated sufficiently small or close enough to the minimal one when comparing the number of states. In turn, the state reduction algorithm can be performed efficiently. Ćirić et al. discussed this problem in [4], and then in [5], [17] and [16], where they proposed state reduction methods that construct sequences of fuzzy matrices. The drawback of these methods is that these sequences can be infinite when the underlying structure of truth values is not locally finite. However, even when the sequences of matrices are finite, the number of different elements in sequences can be high. For the reasons above, different authors have proposed an approximate approach not only in the context of state reduction but also in some other close contexts, such as containment and equivalence of fuzzy automata, as well as simulations and bisimulations between fuzzy automata (see [3, 8, 9, 10, 11, 13, 12, 15, 18, 19, 21] and articles cited there).

In the approximate state reduction problem, the main goal is to construct an automaton with a smaller number of states than a given automaton with behaviour that does not have to be identical to the behavior of a given automaton, but "close enough" to it. Dominantly, scholars have defined the closeness between the behaviors of two automata by the concept of the degree of equality of fuzzy sets [3, 21, 19, 11]. However, when we take membership values from the real unit interval, the conventional metric on that interval can also be used to define closeness [8, 20]. Furthermore, a different fuzzy similarity measure has been proposed recently in [15, 14] via relational lifting. In this paper, we approach approximate state

*This research was supported by the Science Fund of the Republic of Serbia, Grant no 7750185, Quantitative Automata Models: Fundamental Problems and Applications - QUAM

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 51–66, doi:10.4204/EPTCS.386.6

reductions differently. Namely, we require that the behaviors of a given automaton and its reduced one may not be strictly equivalent, but equivalent for all words with length not exceeding a given natural number k. This relaxation from the strict equivalence comes naturally, as when working with fuzzy automata in practical situations, one encounters words with finite (bounded) length. It is important to emphasize that Nguyen et al. recently employed a similar idea in [12] to generalize fuzzy simulations and fuzzy bisimulations for fuzzy automata [11]. In this paper, k-equivalence means the equivalence of a given fuzzy automaton and its reduced one for all words not exceeding length k. Similarly, k-reduction means a state reduction resulting in an automaton k-equivalent to the given automaton.

This paper provides four k-reduction methods based on state reduction methods developed in [17] that output a fuzzy automaton strictly equivalent to the given fuzzy automaton. Precisely, the first two methods consist of constructing a descending sequence of fuzzy quasi-order matrices. Theorems 4.1 and 4.2 prove that the fuzzy automaton formed by the different row vectors of the kth member of that sequence of matrices (the numbering starts from 0) is k-equivalent to the given fuzzy automaton. Moreover, if the number of different elements in this sequence is not greater than k, then the resulting fuzzy automaton is also strictly equivalent to the given fuzzy automaton. In locally finite structures, such as the Gödel or Łukasiewicz structure, the sequence necessarily has a finite number of different elements. Therefore, we can always pick a sufficiently high $k \in \mathbb{N}$ in these structures, so the resulting fuzzy automaton is also strictly equivalent to the given fuzzy automaton. On the other hand, for some non-locally structures satisfying some additional conditions [4, 17], such as the product structure, a reduced fuzzy automaton strictly equivalent to the original fuzzy automaton can be constructed from different row vectors of the infimum of this sequence. Therefore, by choosing the kth member of the sequence, the resulting k-equivalent reduced fuzzy automaton can be regarded as an approximation of the reduced fuzzy automaton strictly equivalent to the original fuzzy automaton.

The other two methods for performing state reduction introduced in [17] consist of constructing a family of fuzzy quasi-order matrices such that a reduced fuzzy automaton built from different row vectors of the infimum of this family is strictly equivalent to the original fuzzy automaton. These methods generally give better reductions than the first two, but their time complexity is generally higher. Here we transform that family into a sequence of fuzzy quasi-order matrices and prove that the fuzzy automaton formed by different row vectors of the kth member of that sequence (the numbering again starts from 0) is k-equivalent to the original automaton (Theorems 4.3 and 4.4). We also point out the advantages and disadvantages of the four proposed methods.

2 Preliminaries

Throughout this paper, $\mathbb{N}$ will denote the set of all natural numbers (including the zero).

A *resuduated lattice* is defined as an algebra $\mathbb{L} = (\mathbb{L}, \vee, \wedge, \otimes, \rightarrow, 0, 1)$, with four binary operations and two constants 0 and 1, which satisfies the following conditions:

(R1) $(\mathbb{L}, \vee, \wedge, 0, 1)$ is a bounded lattice with the least element 0 and the greatest element 1;

(R2) $(\mathbb{L}, \otimes, 1)$ is a commutative semigroup with the identity 1;

(R3) the pair $(\otimes, \rightarrow)$ satisfies the *adjunction* or *residuation property*: for all $a, b, c \in \mathbb{L}$,

$$a \otimes b \leqslant c \;\Leftrightarrow\; a \leqslant b \rightarrow c.$$

Here $\leqslant$ stands for the ordering in the lattice from (R1). The operation $\rightarrow$ is called the *residuum*, and $\otimes$ is called the *multiplication*. If the bounded lattice from (R1) is complete, then $\mathbb{L}$ is called a *complete residuated lattice*. As it is customary in the theory of algebraic structures to use the same notation for an

algebra and its carrier set, here we denote the residuated lattice and its carrier set with the same symbol $\mathbb{L}$ as well.

The main examples of complete residuated lattices are those whose carrier set is the real unit interval $\mathbb{I} = [0,1]$ and the multiplication is some triangular norm on $\mathbb{I}$, such as, for example, the Gödel structure, product structure and Łukasiewicz structure. For more information about complete residuated lattices and the mentioned structures on $\mathbb{I}$ we refer to the books [2, 1] and other papers listed below, in the list of references.

Let $\mathbb{L}$ be an arbitrary complete residuated lattice. For arbitrary $m,n \in \mathbb{N} \setminus \{0\}$, by $\mathbb{L}^{m\times n}$ we denote the set of all $m \times n$ matrices with entries in $\mathbb{L}$, and by $\mathbb{L}^n$ the set of all vectors of size n with entries in $\mathbb{L}$ (by the size of a vector we mean the number of its coordinates). A *fuzzy subset* of a non-empty set A is defined as any function $\alpha : A \to \mathbb{L}$, and a *fuzzy relation* on A is defined as any fuzzy subset of $A \times A$, that is, as any function $M : A \times A \to \mathbb{L}$. For $a \in A$, the value $\alpha(a)$ is called the *membership degree* of a in the fuzzy set α. Here we deal mostly with fuzzy subsets of a finite set, as well as with fuzzy relations on a finite set, and then, when dealing with a finite set $A = \{a_1, a_2, \ldots, a_n\}$, a fuzzy subset α of A is identified with a vector from $\mathbb{L}^n$ whose ith coordinate is $\alpha(a_i)$, while a fuzzy relation M on A is identified with a matrix from $\mathbb{L}^{n\times n}$ whose (i,j)-entry is $M(a_i,a_j)$. Without risk of confusion, the vector corresponding to the fuzzy subset α is denoted by the same symbol α, and its ith coordinate is denoted by $\alpha(i)$, while the matrix corresponding to the fuzzy relation M is denoted by the same symbol M, and its (i,j)-entry is denoted by $M(i,j)$.

For a matrix $M \in \mathbb{L}^{n\times n}$ and a fixed $i \in [1..n]$, where $[1..n] = \{1,2,\ldots,n\}$, the vector whose jth coordinate is $M(i,j)$, for any $j \in [1..n]$, is called the ith *row vector* of M, and for a fixed $j \in [1..n]$, the vector whose ith coordinate is $M(i,j)$, for any $i \in [1..n]$, is called the jth *column vector* of M.

The product $M \cdot N$ of two matrices $M,N \in \mathbb{L}^{n\times n}$ (fuzzy relations on A) is a matrix from $\mathbb{L}^{n\times n}$ (a fuzzy relation on A) defined by

$$(M \cdot N)(i,j) = \bigvee_{s=1}^{n} M(i,s) \otimes N(s,j),$$

for all $i,j \in [1..n]$, the products $\alpha \cdot M$ and $M \cdot \beta$ of vectors $\alpha,\beta \in \mathbb{L}^n$ (fuzzy subsets of A) and a matrix $M \in \mathbb{L}^{n\times n}$ (fuzzy relation on A) are vectors from $\mathbb{L}^n$ (fuzzy subsets of A) defined by

$$(\alpha \cdot M)(i) = \bigvee_{s=1}^{n} \alpha(s) \otimes M(s,i), \qquad (M \cdot \beta)(i) = \bigvee_{s=1}^{n} M(i,s) \otimes \beta(s),$$

for every $i \in [1..n]$, and the product $\alpha \cdot \beta$ of two vectors from $\mathbb{L}^n$ (fuzzy subsets of A) is the element from $\mathbb{L}$ defined by

$$\alpha \cdot \beta = \bigvee_{s=1}^{n} \alpha(s) \otimes \beta(s).$$

The last product $\alpha \cdot \beta$ is called the *scalar product* or *dot product* of vectors (fuzzy subsets) α and β.

The ordering $\leqslant$ on $\mathbb{L}^{n\times n}$ is defined entrywise by

$$M \leqslant N \;\Leftrightarrow\; M(i,j) \leqslant N(i,j), \text{ for all } i,j \in [1..n],$$

for all $M,N \in \mathbb{L}^{n\times n}$, and similarly, the ordering $\leqslant$ on $\mathbb{L}^n$ is defined coordinatewise by

$$\alpha \leqslant \beta \;\Leftrightarrow\; \alpha(i) \leqslant \beta(i), \text{ for each } i \in [1..n],$$

for all $\alpha,\beta \in \mathbb{L}^n$. It is easy to verify that these orderings on $\mathbb{L}^{n\times n}$ and $\mathbb{L}^n$ are compatible with matrix products and vector-matrix products, that is,

$$\begin{aligned} &\alpha \leqslant \beta \;\Rightarrow\; \alpha\cdot M \leqslant \beta\cdot M,\;\; M\cdot\alpha \leqslant M\cdot\beta \\ &M \leqslant N \;\Rightarrow\; K\cdot M \leqslant K\cdot N,\;\; M\cdot K \leqslant N\cdot K,\;\; \alpha\cdot M \leqslant \alpha\cdot N,\;\; M\cdot\alpha \leqslant N\cdot\alpha \end{aligned}$$

for all $\alpha,\beta \in \mathbb{L}^n$ and $K,M,N \in \mathbb{L}^{n\times n}$. The supremum and infimum of a family $\{M_s\}_{s\in I}$ of matrices from $\mathbb{L}^{n\times n}$ are respectively matrices from $\mathbb{L}^{n\times n}$ defined by

$$\big(\bigvee_{s\in I} M_s\big)(i,j) = \bigvee_{s\in I} M_s(i,j), \qquad \big(\bigwedge_{s\in I} M_s\big)(i,j) = \bigwedge_{s\in I} M_s(i,j),$$

for all $i,j \in [1..n]$.

The matrix $I_n \in \mathbb{L}^{n\times n}$ defined by $I_n(i,j) = 1$, for $i = j$, and $I_n(i,j) = 0$, for $i \neq j$, $i,j \in [1..n]$, is called the *identity matrix* of order n. A matrix $M \in \mathbb{L}^{n\times n}$ is *reflexive* if $I_n \leqslant M$, it is *transitive* if $M^2 \leqslant M$, where $M^2 = M\cdot M$, and it is *symmetric* if $M(i,j) = M(j,i)$, for all $i,j \in [1..n]$. A reflexive and transitive matrix is called a *fuzzy quasi-order matrix*, and a symmetric fuzzy quasi-order matrix is called a *fuzzy equivalence matrix*.

For matrices $M,N \in \mathbb{L}^{n\times n}$, the *right residual* of N by M is a matrix $M\backslash N \in \mathbb{L}^{n\times n}$ defined by

$$(M\backslash N)(j,k) = \bigwedge_{i=1}^{n} M(i,j) \to N(i,k), \tag{1}$$

for all $j,k \in [1..n]$, and the *left residual* of N by M is a matrix $N/M \in \mathbb{L}^{n\times n}$ defined by

$$(N/M)(i,j) = \bigwedge_{k=1}^{n} M(j,k) \to N(i,k), \tag{2}$$

for all $i,j \in [1..n]$. Matrix residuals are related with matrix multiplication by the following *residuation* (*adjunction*) *property*:

$$K\cdot M \leqslant N \;\Leftrightarrow\; M \leqslant K\backslash N \;\Leftrightarrow\; K \leqslant N/M, \tag{3}$$

for all $K,M,N \in \mathbb{L}^{n\times n}$. Next, for $\alpha,\beta \in \mathbb{L}^n$, the *right residual* of β by α is a matrix $\alpha\backslash\beta \in \mathbb{L}^{n\times n}$ given by

$$(\alpha\backslash\beta)(i,j) = \alpha(i) \to \beta(j), \tag{4}$$

for all $i,j \in [1..n]$, and the *left residual* of β by α is a matrix $\beta/\alpha \in \mathbb{L}^{n\times n}$ given by

$$(\beta/\alpha)(j,i) = \alpha(i) \to \beta(j), \tag{5}$$

for all $i,j \in [1..n]$. It is clear that $\alpha\backslash\beta = (\beta/\alpha)^\top$, that is, $\beta/\alpha = (\alpha\backslash\beta)^\top$ (here $M^\top$ denotes the *transpose* of a matrix M). The *residuation property* for these residuals is

$$\alpha\cdot M \leqslant \beta \;\Leftrightarrow\; M \leqslant \alpha\backslash\beta, \qquad\qquad M\cdot\alpha \leqslant \beta \;\Leftrightarrow\; M \leqslant \beta/\alpha, \tag{6}$$

for all $\alpha,\beta \in \mathbb{L}^n$ and $M \in \mathbb{L}^{n\times n}$.

The representation of the matrix $M \in \mathbb{L}^{m\times n}$ in the form of the product $M = L\cdot R$, where $L \in \mathbb{L}^{m\times r}$ and $R \in \mathbb{L}^{r\times n}$, is called the *r-factorization* of that matrix M. The smallest number r for which there is an

r-factorization of the matrix M is denoted by $\rho(M)$ and is called the *Schein's rank* or *factor rank* of M. The concepts of r-factorization and Schein's rank are defined for arbitrary matrices, but have particularly good properties when applied to fuzzy quasi-order matrices (cf. [16]). As shown in [6, 17], a fuzzy quasi-order matrix Q has the same number of different row vectors and different column vectors, which is denoted by $d(Q)$. In the general case, $\rho(Q) \leqslant d(Q)$, but for every fuzzy quasi-order matrix Q with entries in a complete residuated lattice $\mathbb{L}$ in which for all $a, b \in \mathbb{L}$ from $a \vee b = 1$ it follows $a = 1$ or $b = 1$ (for instance, this holds if $\mathbb{L}$ is linearly ordered), we have that $\rho(Q) = d(Q)$ (cf. [16]).

For undefined notions and notation we refer to [2, 1].

3 Fuzzy finite automata and the state reduction problem

Throughout this paper, if not noted otherwise, $\mathbb{L}$ will denote an arbitrary complete residuated lattice and X will denote an arbitrary non-empty alphabet. By X^* we denote the free monoid over X, whose identity, called the *empty word*, is denoted by ε, while by X^+ we denote the free semigroup over X, i.e., $X^+ = X^* \setminus \{\varepsilon\}$.

A *fuzzy finite automaton* over $\mathbb{L}$ and X is defined as a tuple $\mathcal{A} = (A, \sigma, \delta, \tau)$, where A is a non-empty finite set, while σ, τ and δ are functions such that $\sigma, \tau : A \to \mathbb{L}$ and $\delta : A \times X \times A \to \mathbb{L}$. The function δ is often replaced by the family of functions $\{\delta_x\}_{x \in X}$, where $\delta_x : A \times A \to \mathbb{L}$ is given by $\delta_x(a,b) = \delta(a,x,b)$, for all $a, b \in A$ and $x \in X$. We call A the *set of states*, σ the *fuzzy set of initial states*, τ the *fuzzy set of terminal states*, and δ and δ_x, $x \in X$, the *fuzzy transition functions*. The number of states of $\mathcal{A}$ will be denoted by $|\mathcal{A}|$.

The *behavior* of the fuzzy finite automaton $\mathcal{A}$ is a function $[\![\mathcal{A}]\!] : X^* \to \mathbb{L}$ (i.e., a fuzzy subset of X^*) defined by

$$[\![\mathcal{A}]\!](u) = \bigvee_{(a_0,a_1,\ldots,a_k) \in A^{k+1}} \sigma(a_0) \otimes \delta(a_0,x_1,a_1) \otimes \delta(a_1,x_2,a_2) \cdots \delta(a_{k-1},x_k,a_k) \otimes \tau(a_k), \tag{7}$$

for $u = x_1 x_2 \ldots x_k \in X^+$, $x_1, x_2, \ldots, x_k \in X$, and

$$[\![\mathcal{A}]\!](\varepsilon) = \bigvee_{a \in A} \sigma(a) \otimes \tau(a). \tag{8}$$

We say that $[\![\mathcal{A}]\!]$ is the *fuzzy language recognized (accepted) by* the fuzzy finite automaton $\mathcal{A}$, or in short just the *fuzzy language of* $\mathcal{A}$.

As we noted in the previous section, fuzzy subsets of a finite set with n elements can be treated as fuzzy vectors. i.e., as vectors from $\mathbb{L}^n$, while fuzzy relations on such a set can be treated as fuzzy matrices, i.e. as matrices from $\mathbb{L}^{n \times n}$. When such a way of viewing is applied to the fuzzy finite automaton $\mathcal{A}$, then σ and τ are treated as vectors from $\mathbb{L}^n$, called respectively the *initial fuzzy vector* and *terminal fuzzy vector*, while δ_x, $x \in X$, are treated as matrices from $\mathbb{L}^{n \times n}$, called the *transition fuzzy matrices*. Then the tuple is called the *linear representation* of the fuzzy finite automaton $\mathcal{A}$, while n is called the *dimension* of $\mathcal{A}$. Such a way of looking at σ, τ and δ_x's will be applied here as well. In the vector-matrix form, the behavior of the fuzzy finite automaton $\mathcal{A}$ is represented as follows:

$$[\![\mathcal{A}]\!](u) = \sigma \cdot \delta_{x_1} \cdot \delta_{x_2} \cdots \delta_{x_s} \cdot \tau = \sigma \cdot \delta_u \cdot \tau, \tag{9}$$

for $u = x_1 x_2 \ldots x_s \in X^+$, $x_1, x_2, \ldots, x_s \in X$, where $\delta_u = \delta_{x_1} \cdot \delta_{x_2} \cdots \delta_{x_k}$, and

$$[\![\mathcal{A}]\!](\varepsilon) = \sigma \cdot \tau. \tag{10}$$

As can be seen, here we treat σ as a row vector and τ as a column vector.

Two fuzzy finite automata $\mathcal{A}$ and $\mathcal{B}$ over $\mathbb{L}$ and X are said to be *equivalent* if

$$[\![\mathcal{A}]\!](u) = [\![\mathcal{B}]\!](u), \qquad \text{for every } u \in X^*, \tag{11}$$

i.e., if $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$, and for an arbitrary $k \in \mathbb{N}$, we say that $\mathcal{A}$ and $\mathcal{B}$ are *k-equivalent* if

$$[\![\mathcal{A}]\!](u) = [\![\mathcal{B}]\!](u), \qquad \text{for every } u \in X^* \text{ such that } |u| \leqslant k. \tag{12}$$

If $\mathcal{A}$ and $\mathcal{B}$ are equivalent or k-equivalent, we also say that $\mathcal{A}$ is *equivalent to* or *k-equivalent to* $\mathcal{B}$, and vice versa.

Let $Q \in \mathbb{L}^{n\times n}$ be an arbitrary fuzzy quasi-order matrix. Let $1 \leq i_1 \leq \ldots \leq i_k \leq n$ be the smallest indices of all pairwise distinct rows of Q. As mentioned earlier [6, 17], they are also the smallest indices of all pairwise distinct columns of Q. Let $Q_r \in \mathbb{L}^{k\times n}$ (respectively, $Q_c \in \mathbb{L}^{n\times k}$) be the matrix consisting of the rows (respectively, columns) $i_1, \ldots, i_k$ of Q. Let $\mathcal{A} = (A, \sigma, \delta, \tau)$ be a fuzzy finite automaton over $\mathbb{L}$ and X with n states. Then, a new fuzzy finite automaton, with the linear representation

$$\mathcal{A}_Q = (k, \sigma^Q, \{\delta_x^Q\}_{x\in X}, \tau^Q),$$

is defined by putting that

$$\begin{aligned}
&\sigma^Q = \sigma \cdot Q_c,\\
&\delta_x^Q = Q_r \cdot \delta_x \cdot Q_c, \quad \text{for all } x \in X,\\
&\tau^Q = Q_r \cdot \tau.
\end{aligned}$$

It is clear that $\sigma^Q, \tau^Q \in \mathbb{L}^k$ and $\delta_x^Q \in \mathbb{L}^{k\times k}$, for each $x \in X$, so $\mathcal{A}_Q$ is a well-defined fuzzy finite automaton. Since $Q = Q_c \cdot Q_r$ (cf. [16, Theorem 4.1], the behavior of $\mathcal{A}_Q$ is given in the vector-matrix form by

$$[\![\mathcal{A}_Q]\!](u) = \sigma \cdot Q \cdot \delta_{x_1} \cdot Q \cdot \delta_{x_2} \cdot \cdots \cdot Q \cdot \delta_{x_s} \cdot Q \cdot \tau, \tag{13}$$

for every $u = x_1 x_2 \ldots x_s \in X^+$, where $x_1, x_2, \ldots, x_s \in X$, and

$$[\![\mathcal{A}_Q]\!](\varepsilon) = \sigma \cdot Q \cdot \tau. \tag{14}$$

As we mentioned earlier, fuzzy matrices can be identified with fuzzy relations between finite sets. Since in the theory of fuzzy sets, the "rows" of fuzzy relations are known as *aftersets*, and the "columns" are known as *foresets*, the fuzzy finite automaton $\mathcal{A}_Q$ was called in [17] the *afterset automaton* of $\mathcal{A}$ corresponding to the fuzzy quasi-order matrix Q, but it can also be rightly called the *row automaton*. It was also shown in [17] that if in the construction of the automaton $\mathcal{A}_Q$ instead of the rows (aftersets) of the matrix Q we use its columns (foresets), then essentially nothing changes, because an isomorphic fuzzy finite automaton is obtained.

Let us note that the number of states of $\mathcal{A}_Q$ is $d(Q) \leqslant n$, that is, $\mathcal{A}_Q$ has less than or equal number of states with $\mathcal{A}$. Therefore, by constructing the automaton $\mathcal{A}_Q$ we can reduce the number of states of the automaton $\mathcal{A}$, provided that these two automata are equivalent. Consequently, the main question here is *under what conditions on Q the automaton $\mathcal{A}_Q$ is equivalent to $\mathcal{A}$*? This question can also be formulated as: *under what conditions on Q the construction of $\mathcal{A}_Q$ preserves the fuzzy language of the automaton*

$\mathcal{A}$? More answers to these questions were provided in [5] and [17]. *Right invariant matrices* were defined in [5, 17] as solutions of the system of matrix equations

(ri-1) $U \cdot \tau \leqslant \tau$;
(ri-2) $U \cdot \delta_x \leqslant \delta_x \cdot U$, for all $x \in X$,

where U is an unknown matrix taking values in $\mathbb{L}^{n\times n}$, and *left invariant matrices* were defined as solutions of the system

(li-1) $\sigma \cdot U \leqslant \sigma$;
(li-2) $\delta_x \cdot U \leqslant U \cdot \delta_x$, for all $x \in X$.

It was proved in [5] that both right and left invariant fuzzy equivalence matrices provide equivalence between $\mathcal{A}$ and the corresponding row automaton, while the same for fuzzy quasi-order matrices was proved in [17]. At the same time, it has been proven that fuzzy quasi-order matrices give better reductions than fuzzy equivalence matrices, in the sense that they produce fuzzy finite automata with a smaller number of states.

Procedures for computing the greatest right and left invariant matrices, which are necessarily fuzzy quasi-order matrices, were provided in [17]. For right invariant matrices, the procedure consists of building a decreasing sequence of matrices, which is defined in the next section by formula (17). When there are two equal consecutive members of that sequence, the sequence is finite and stabilizes at some member which is the greatest right invariant matrix. For instance, if $\mathbb{L}$ is the Gödel structure or Boolean algebra, then every such sequence is finite and the greatest right invariant matrix can be computed in a finite number of steps. However, there are also cases when this sequence is infinite and the greatest right invariant matrix can not be computed in a finite number of steps. For example, this may happen when $\mathbb{L}$ is the product structure. All this also applies to left-invariant matrices, where the procedure for computing the greatest such matrix is based on the decreasing sequence of matrices defined in the next section by formula (20).

In paper [17], another way was also provided to get a fuzzy quasi-order matrix Q for which the construction of $\mathcal{A}_Q$ will preserve the language of $\mathcal{A}$. *Weakly right invariant matrices* were defined in as solutions of the system of matrix equations

(wri) $U \cdot \tau_u \leqslant \tau_u$, for all $u \in X^*$,

where U is an unknown matrix taking values in $\mathbb{L}^{n\times n}$ and $\tau_u = \delta_u \cdot \tau$, and *weakly left invariant matrices* were defined as solutions of the system

(wli) $\sigma_u \cdot U \leqslant \sigma_u$, for all $u \in X^*$,

where $\sigma_u = \sigma \cdot \delta_u$. As shown in [17], both for any weakly right invariant or weakly left invariant fuzzy quasi-order matrix Q, the row automaton $\mathcal{A}_Q$ is equivalent to $\mathcal{A}$, and the greatest weakly right invariant fuzzy quasi-order, i.e., the greatest solution of (wri), can be expressed as

$$\bigwedge_{u\in X^*} \tau_u/\tau_u, \tag{15}$$

while the greatest weakly left invariant fuzzy quasi-order, i.e., the greatest solution of (wli), can be expressed as

$$\bigwedge_{u\in X^*} \sigma_u\backslash\sigma_u. \tag{16}$$

In general, the greatest weakly right invariant fuzzy quasi-order matrix provides better reduction than the greatest right invariant one, but its computation may be significantly more complex. There may also be a problem of efficient computation of these matrices, because the families $\{\tau_u \mid u \in X^*\}$ and $\{\sigma_u \mid u \in X^*\}$ can be infinite, and even when they are finite, the number of their members can be too large.

All the mentioned problems that concern the computation of the greatest invariant and weakly invariant fuzzy quasi-order matrices actualize the issue of approximate state reductions of fuzzy finite automata, which will be discussed in the next section.

Note that in [16] a method for additional reduction of the number of states of the automaton $\mathcal{A}_Q$ is offered, which is based on the r-factorization of the fuzzy quasi-order matrix Q. Namely, let $Q = L \cdot R$ be an r-factorization of Q, i.e., $L \in \mathbb{L}^{n\times r}$ and $R \in \mathbb{L}^{r\times n}$. Then we can construct a fuzzy finite automaton $\mathcal{A}_{L|R}$ with the linear representation

$$\mathcal{A}_{L|R} = (r, \sigma^{L|R}, \{\delta_x^{L|R}\}_{x\in X}, \tau^{L|R}),$$

where

$$\begin{aligned}
&\sigma^{L|R} = \sigma \cdot L,\\
&\delta_x^{L|R} = R \cdot \delta_x \cdot L, \quad \text{for all } x \in X,\\
&\tau^{L|R} = R \cdot \tau.
\end{aligned}$$

Then $\sigma^{L|R}, \tau^{L|R} \in \mathbb{L}^r$ and $\delta_x^{L|R} \in \mathbb{L}^{r\times r}$, so $\mathcal{A}_{L|R}$ is a well-defined fuzzy finite automaton with r states. According to (13), (14) and $Q = L \cdot R$ we have that $[\![\mathcal{A}_{L|R}]\!] = [\![\mathcal{A}_Q]\!]$. Therefore, when we reduce the number of states of the fuzzy finite automaton $\mathcal{A}$ using the greatest fuzzy quasi-order matrix Q, an additional reduction of the number of states could be performed with the help of an r-factorization of Q. Clearly, the smallest number of states we can obtain in this way is $\rho(Q)$, the Schein's rank of Q.

4 Approximate state reduction: k-reduction

As we already noted in the introduction, there were already several articles dealing with approximate state reduction, mainly in the context of studying approximate simulations and bisimulations between fuzzy finite automata. The approach used in those articles was to, starting from a given fuzzy finite automaton, construct a new fuzzy finite automaton with a smaller number of states, whose fuzzy language does not have to be identical to the fuzzy language of the original automaton, but is sufficiently similar to that fuzzy language, in relation to a certain measure of similarity. In most papers, that measure was based on subsethood and equality degrees between fuzzy sets.

In this paper, we use a different approach. We require that between fuzzy languages of the original and the reduced fuzzy finite automaton there is an exact match for all words of length less than or equal to k, while membership degrees for longer words do not matter to us. In the terminology from the previous section, this means that these fuzzy languages are k-equivalent. Such an approach is quite natural if we keep in mind that in practical applications of automata the length of input words is always limited, so the only thing that matters to us is that the membership degrees match for words whose length does not exceed that limit.

A procedure for reduction of the number of states of a fuzzy finite automaton $\mathcal{A}$, which results in a fuzzy finite automaton that is k-equivalent to $\mathcal{A}$, will be called a *k-reduction*. The following theorem provides such a procedure that is based on the procedure for reduction of the number of states, provided in [17], that results in an automaton which is strictly equivalent to the original automaton.

Theorem 4.1 *Let $\mathcal{A} = (A, \sigma, \delta, \tau)$ be a fuzzy finite automaton over $\mathbb{L}$ and X with n states.*

Let us inductively define a sequence of matrices $\{Q_k\}_{k\in\mathbb{N}} \subset \mathbb{L}^{n\times n}$ as follows:

$$Q_0 = \tau/\tau, \qquad Q_{k+1} = Q_k \wedge \bigwedge_{x\in X} [(\delta_x \cdot Q_k)/\delta_x], \quad \text{for every } k \in \mathbb{N}. \tag{17}$$

Then for an arbitrary $k \in \mathbb{N}$ the following statements hold:

(a) *Q_k is a fuzzy quasi-order matrix;*
(b) *$\mathcal{A}_{Q_k}$ is k-equivalent to $\mathcal{A}$;*
(c) *if $Q_k = L \cdot R$ is an r-factorization of Q_k, for some $r \leqslant d(Q_k)$, then $\mathcal{A}_{L|R}$ is k-equivalent to $\mathcal{A}$;*
(d) *if $Q_s = Q_{s+1}$, for some $s \leqslant k$, then $Q_k = Q_s$ and both $\mathcal{A}_{Q_k}$ and $\mathcal{A}_{L|R}$ are equivalent to $\mathcal{A}$.*

Proof. (a) By its definition, Q_0 is the greatest solution of the inequation $U \cdot \tau \leqslant \tau$, where U is an unknown matrix taking values in $\mathbb{L}^{n\times n}$. Since I_n is also a solution to this inequation, we conclude that $I_n \leqslant Q_0$. Moreover, we have that

$$Q_0^2 \cdot \tau = Q_0 \cdot Q_0 \cdot \tau \leqslant Q_0 \cdot \tau \leqslant \tau,$$

which means that Q_0^2 is also a solution of $U \cdot \tau \leqslant \tau$, and since Q_0 is the greatest solution to this inequation, we conclude that $Q_0^2 \leqslant Q_0$. This proves that Q_0 is a fuzzy quasi-order matrix.

Suppose that Q_s is a fuzzy quasi-order matrix, for some $s \in \mathbb{N}_0$. Let us observe that for every $x \in X$ the matrix $M_x = (\delta_x \cdot Q_s)/\delta_x$ is the greatest solution of the inequation $U \cdot \delta_x \leqslant \delta_x \cdot Q_s$, with an unknown matrix U. We also have that $I_n \leqslant Q_s$, whence

$$I_n \cdot \delta_x = \delta_x \cdot I_n \leqslant \delta_x \cdot Q_s,$$

which means that I_n is also a solution to the inequation $U \cdot \delta_x \leqslant \delta_x \cdot Q_s$, and consequently, $I_n \leqslant M_x$, since M_x is the greatest solution of this inequation.

On the other hand, we have that

$$M_x^2 \cdot \delta_x = M_x \cdot M_x \cdot \delta_x \leqslant M_x \cdot \delta_x \cdot Q_s \leqslant \delta_x \cdot Q_s \cdot Q_s = \delta_x \cdot Q_s^2 = \delta_x \cdot Q_s,$$

which means that M_x^2 is also a solution of $U \cdot \delta_x \leqslant \delta_x \cdot Q_s$, and thus, $M_x^2 \leqslant M_x$. Hence, $M_x = (\delta_x \cdot Q_s)/\delta_x$ is a fuzzy quasi-order matrix, for each $x \in X$, and the matrix Q_{s+1} is also a fuzzy quasi-order matrix, as the intersection of the family of fuzzy quasi-order matrices $M_x = (\delta_x \cdot Q_s)/\delta_x$, $x \in X$, and the fuzzy quasi-order matrix Q_s.

Let us also note that Q_{s+1} is the greatest solution of the system of linear matrix inequations

$$U \cdot \delta_x \leqslant \delta_x \cdot Q_s, \quad x \in X, \tag{18}$$

contained in Q_s, where U is an unknown matrix taking values in $\mathbb{L}^{n\times n}$.

(b) According to (17), we have that $Q_0 \cdot \tau \leqslant \tau$ and $Q_{t+1} \cdot \delta_x \leqslant \delta_x \cdot Q_t$, for arbitrary $t \in \mathbb{N}_0$ and $x \in X$. On the other hand, from $Q_t \leqslant Q_0$ it follows that $Q_t \cdot \tau \leqslant Q_0 \cdot \tau \leqslant \tau$, whereas from $I_n \leqslant Q_t$ we obtain that $\tau = I_n \cdot \tau \leqslant Q_t \cdot \tau$. Furthermore, since $Q_{t+1} \cdot \delta_x \leqslant \delta_x \cdot Q_t$, $Q_t^2 = Q_t$ and $I_n \leqslant Q_{t+1}$, we have that

$$Q_{t+1} \cdot \delta_x \cdot Q_t \leqslant \delta_x \cdot Q_t^2 = \delta_x \cdot Q_t, \qquad \delta_x \cdot Q_t = I_n \cdot \delta_x \cdot Q_t \leqslant Q_{t+1} \cdot \delta_x \cdot Q_t.$$

Therefore, we have proved that

$$Q_t \cdot \tau = \tau, \quad Q_{t+1} \cdot \delta_x \cdot Q_t = \delta_x \cdot Q_t, \tag{19}$$

for all $t \in \mathbb{N}_0$ and $x \in X$.

Now we have that

$$[\![\mathcal{A}_{Q_k}]\!](\varepsilon) = \sigma \cdot Q_k \cdot \tau = \sigma \cdot \tau = [\![\mathcal{A}]\!](\varepsilon),$$

and for $u = x_1 x_2 \ldots x_s \in X^+$, where $x_1, x_2, \ldots, x_s \in X$ and $s \leqslant k$, we have that

$$[\![\mathcal{A}]\!](u) = \sigma \cdot \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot \tau \leqslant \sigma \cdot Q_k \cdot \delta_{x_1} \cdot Q_k \cdot \delta_{x_2} \cdot \ldots \cdot Q_k \cdot \delta_{x_s} \cdot Q_k \cdot \tau = [\![\mathcal{A}_{Q_k}]\!](u),$$

and

$$\begin{aligned}[\![\mathcal{A}_{Q_k}]\!](u) &= \sigma \cdot Q_k \cdot \delta_{x_1} \cdot Q_k \cdot \delta_{x_2} \cdot \ldots \cdot Q_k \cdot \delta_{x_s} \cdot Q_k \cdot \tau \leqslant \sigma \cdot Q_s \cdot \delta_{x_1} \cdot Q_{s-1} \cdot \delta_{x_2} \cdot \ldots \cdot Q_1 \cdot \delta_{x_s} \cdot Q_0 \cdot \tau \\ &= \sigma \cdot \delta_{x_1} \cdot Q_{s-1} \cdot \delta_{x_2} \cdot \ldots \cdot Q_1 \cdot \delta_{x_s} \cdot Q_0 \cdot \tau = \sigma \cdot \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot Q_1 \cdot \delta_{x_s} \cdot Q_0 \cdot \tau \\ &= \ldots = \sigma \cdot \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot Q_0 \cdot \tau = \sigma \cdot \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot \tau = [\![\mathcal{A}]\!](u).\end{aligned}$$

Therefore, we have proved that $[\![\mathcal{A}_{Q_k}]\!](u) = [\![\mathcal{A}]\!](u)$, for every $u \in X^*$ such that $|u| = s \leqslant k$, which means that the fuzzy automaton $\mathcal{A}_{Q_k}$ is k-equivalent to $\mathcal{A}$.

(c) This follows by the fact that $[\![\mathcal{A}_{Q_k}]\!] = [\![\mathcal{A}_{L|R}]\!]$ (cf. [16]).

(d) Assume that $Q_s = Q_{s+1}$, for some $s \leqslant k$. As shown in [17], then $Q_s = Q_t$, for every $t \geqslant s$, $t \in \mathbb{N}_0$, and $[\![\mathcal{A}_{Q_s}]\!] = [\![\mathcal{A}]\!]$ holds. Therefore, we have that $Q_k = Q_s$, so $[\![\mathcal{A}_{Q_k}]\!] = [\![\mathcal{A}_{L|R}]\!] = [\![\mathcal{A}_{Q_s}]\!] = [\![\mathcal{A}]\!]$. □

It is worth noting that in assertion (c) of Theorem 4.1 we are talking about the r-decomposition of the matrix Q_k for some $r \in \mathbb{N}$ for which $\rho(Q_k) \leqslant r \leqslant d(Q_k)$, rather than on the $\rho(k)$-decomposition, which would give a smallest row automaton. The reason is that the r-decomposition could be done using an algorithm provided in [16], which consists in removing those row vectors of Q_k that can be represented as linear combinations of other row vectors. The result of that procedure strongly depends on the choice and order of the vectors we remove, so that one can get an r-decomposition for any r such that $\rho(Q_k) \leqslant r \leqslant d(Q_k)$.

Let us also note that the use of the r-decompositions in k-reductions (and reductions in general) is useless if the underlying complete residuated lattice $\mathbb{L}$ satisfies the condition that $a \vee b = 1$ implies that $a = 1$ or $b = 1$, since in this case we have that $\rho(Q_k) = d(Q_k)$ (as shown in [16]), so there is no reduction whatsoever. For instance, this holds when $\mathbb{L}$ is linearly ordered, what includes all the cases when $\mathbb{L}$ is the structure defined on the real unit interval by means of triangular norms.

Similarly to Theorem 4.1, we can prove the following theorem, which provides an alternative way for k-reduction.

Theorem 4.2 *Let $\mathcal{A} = (A, \sigma, \delta, \tau)$ be a fuzzy finite automaton over $\mathbb{L}$ and X with n states.*

Let us inductively define a sequence of matrices $\{P_k\}_{k \in \mathbb{N}} \subset \mathbb{L}^{n \times n}$ as follows:

$$P_0 = \sigma \backslash \sigma, \qquad P_{k+1} = P_k \wedge \bigwedge_{x \in X} [\delta_x \backslash (P_k \cdot \delta_x)], \quad \text{for every } k \in \mathbb{N}. \tag{20}$$

Then for an arbitrary $k \in \mathbb{N}$ the following statements hold:

(a) *P_k is a fuzzy quasi-order matrix;*
(b) *$\mathcal{A}_{P_k}$ is k-equivalent to $\mathcal{A}$;*
(c) *if $P_k = L \cdot R$ is an r-factorization of P_k, for some $r \leqslant d(P_k)$, then $\mathcal{A}_{L|R}$ is k-equivalent to $\mathcal{A}$;*
(d) *if $P_s = P_{s+1}$, for some $s \leqslant k$, then $P_k = P_s$ and both $\mathcal{A}_{P_k}$ and $\mathcal{A}_{L|R}$ are equivalent to $\mathcal{A}$.*

The third way to perform the k-reduction is given by the next theorem.

Theorem 4.3 *Let $\mathcal{A} = (A, \sigma, \delta, \tau)$ be a fuzzy finite automaton over $\mathbb{L}$ and X with n states.*
Let us define a sequence of matrices $\{\widehat{Q}_k\}_{k\in\mathbb{N}} \subset \mathbb{L}^{n\times n}$ as follows:

$$\widehat{Q}_k = \bigwedge_{|u|\leqslant k} \tau_u / \tau_u, \tag{21}$$

for each $k \in \mathbb{N}$. Then for an arbitrary $k \in \mathbb{N}_0$ the following statements hold:
(a) *$\widehat{Q}_k$ is a fuzzy quasi-order matrix and $Q_k \leqslant \widehat{Q}_k$, where Q_k is as in Theorem 4.1;*
(b) *$\mathcal{A}_{\widehat{Q}_k}$ is k-equivalent to $\mathcal{A}$;*
(c) *if $\widehat{Q}_k = L \cdot R$ is an r-factorization of $\widehat{Q}_k$, for some $r \leqslant d(\widehat{Q}_k)$, then $\mathcal{A}_{L|R}$ is k-equivalent to $\mathcal{A}$.*

Proof. (a) We have that $\widehat{Q}_k$ is a fuzzy quasi-order matrix as the infimum of a family of fuzzy quasi-order matrices τ_u/τ_u, $u \in X^*$, $|u| \leqslant k$.

Consider an arbitrary $u \in X^*$ such that $|u| \leqslant k$. Then $u = x_1x_2\ldots x_s$, where $x_1, x_2, \ldots, x_s \in X$ and $s \leqslant k$, and we get

$$\begin{aligned} Q_k \cdot \tau_u &= Q_k \cdot \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot \tau \leqslant Q_s \cdot \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot \tau \leqslant \delta_{x_1} \cdot Q_{s-1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot \tau \\ &\leqslant \delta_{x_1} \cdot \delta_{x_2} \cdot Q_{s-2} \cdot \ldots \cdot \delta_{x_s} \cdot \tau \leqslant \ldots \leqslant \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot Q_1 \cdot \delta_{x_s} \cdot \tau \leqslant \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot Q_0 \cdot \tau \\ &\leqslant \delta_{x_1} \cdot \delta_{x_2} \cdot \ldots \cdot \delta_{x_s} \cdot \tau = \tau_u. \end{aligned}$$

Therefore, $Q_k \cdot \tau_u \leqslant \tau_u$. i.e., $Q_k \leqslant \tau_u/\tau_u$, for every $u \in X^*$ such that $|u| \leqslant k$, so

$$Q_k \leqslant \bigwedge_{|u|\leqslant k} \tau_u/\tau_u = \widehat{Q}_k.$$

(b) According to the definition of $\widehat{Q}_k$ we obtain that $\widehat{Q}_k \cdot \tau_u \leqslant \tau_u$, for every $u \in X^*$, $|u| \leqslant k$, and since $\tau_u = I_n \cdot \tau_u \leqslant \widehat{Q}_k \cdot \tau_u$, we conclude that

$$\widehat{Q}_k \cdot \tau_u = \tau_u, \quad \text{for every } u \in X^*, |u| \leqslant k. \tag{22}$$

Consider again an arbitrary $u \in X^*$ such that $|u| \leqslant k$. According to (22) we get

$$\begin{aligned} [\![\mathcal{A}_{\widehat{Q}_k}]\!](u) &= \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \widehat{Q}_k \cdot \delta_{x_2} \cdot \widehat{Q}_k \cdot \ldots \cdot \widehat{Q}_k \cdot \delta_{x_s} \cdot \widehat{Q}_k \cdot \tau = \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \widehat{Q}_k \cdot \delta_{x_2} \cdot \widehat{Q}_k \cdot \ldots \cdot \widehat{Q}_k \cdot \delta_{x_s} \cdot \tau \\ &= \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \widehat{Q}_k \cdot \delta_{x_2} \cdot \widehat{Q}_k \cdot \ldots \cdot \widehat{Q}_k \cdot \tau_{x_s} = \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \widehat{Q}_k \cdot \delta_{x_2} \cdot \widehat{Q}_k \cdot \ldots \cdot \tau_{x_s} = \ldots \\ &= \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \widehat{Q}_k \cdot \delta_{x_2} \cdot \widehat{Q}_k \cdot \tau_{x_3\ldots x_s} = \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \widehat{Q}_k \cdot \delta_{x_2} \cdot \tau_{x_3\ldots x_s} = \\ &= \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \widehat{Q}_k \cdot \tau_{x_2x_3\ldots x_s} = \sigma \cdot \widehat{Q}_k \cdot \delta_{x_1} \cdot \tau_{x_2x_3\ldots x_s} = \sigma \cdot \widehat{Q}_k \cdot \tau_{x_1x_2x_3\ldots x_s} \\ &= \sigma \cdot \tau_{x_1x_2x_3\ldots x_s} = \sigma \cdot \tau_u = [\![\mathcal{A}]\!](u). \end{aligned}$$

Therefore, we have proved that $\mathcal{A}_{\widehat{Q}_k}$ is k-equivalent to $\mathcal{A}$.
(c) This is proved in the same way as the statement (c) in Theorem 4.1. □

In a similar way we can prove the theorem that provides the fourth way to perform the k-reduction.

Theorem 4.4 *Let $\mathcal{A} = (A, \sigma, \delta, \tau)$ be a fuzzy finite automaton over $\mathbb{L}$ and X with n states.*
Let us define a sequence of matrices $\{\widehat{P}_k\}_{k\in\mathbb{N}} \subset \mathbb{L}^{n\times n}$ as follows:

$$\widehat{P}_k = \bigwedge_{|u|\leqslant k} \sigma_u \backslash \sigma_u, \tag{23}$$

for every $k \in \mathbb{N}$.

Then for an arbitrary $k \in \mathbb{N}$ the following statements hold:

(a) *$\widehat{P}_k$ is a fuzzy quasi-order matrix and $P_k \leqslant \widehat{P}_k$, where P_k is as in Theorem 4.2;*

(b) *$\mathcal{A}_{\widehat{P}_k}$ is k-equivalent to $\mathcal{A}$;*

(c) *if $\widehat{P}_k = L \cdot R$ is an r-factorization of $\widehat{P}_k$, for some $r \leqslant d(\widehat{P}_k)$, then $\mathcal{A}_{L|R}$ is k-equivalent to $\mathcal{A}$.*

The previous four theorems provide four different methods for k-reduction of fuzzy finite automata. The question naturally arises: Do we really need all four methods? In other words, are any of these methods better than others, so that others are not necessary? In the sequel, we will show that each of these methods has some advantages, but also some disadvantages in relation to the others, as well as in relation to methods for full state reduction (reduction that ensures full equivalence) provided in [17].

First we note that the sequence of matrices we use in computing the greatest right invariant (and also left invariant) fuzzy quasi-order matrix may be infinite, and in such cases, the efficiency of reductions using such matrices becomes questionable. In contrast, k-reductions by means of the kth members of sequences defined by (17) and (20) can always be realized in a finite number of steps. In addition, members of those arrays with smaller indices produce automata with fewer states than members with larger indices (see example below), which means that k-reductions generally yield fewer fuzzy finite automata than reductions that result in strictly equivalent fuzzy finite automata.

Regarding k-reductions by means of the kth members of sequences defined by (17) and (21), by $Q_k \leqslant \widehat{Q}_k$ it follows that $\widehat{Q}_k$ generally yields better reduction than Q_k (see Example 4.5), for each $k \in \mathbb{N}_0$. However, computing the matrix $\widehat{Q}_k$ can be significantly more difficult than computing the matrix Q_k, because it requires computing the vectors τ_u, for $u \in X^*$, $|u| \leqslant k$, and their the number can be more than m^k, where m is the number of input letters.

Finally, as far as k-reductions by means of the kth members of sequences defined by (17) and (20) are concerned, they are not comparable. In some cases one of them will give better results, and in other cases the another one. In Example 4.5 we have a case where P_k, for each $k \geqslant 1$, does not perform reduction, while Q_k does, but the opposite would happen on the reverse automaton of the automaton considered in that example. The same can be said for k-reductions by means of the kth members of sequences defined by (21) and (23).

In the following example, we assumed the structure of membership values to be the two-element Boolean algebra $\mathbb{B}$. The reason why we decided so is that $\mathbb{B}$ is a subalgebra of every complete residuated lattice, and every automaton over the two-element Boolean algebra $\mathbb{B}$ can also be considered an automaton over an arbitrary complete residuated lattice.

Example 4.5 Let $\mathcal{A} = (A, \sigma, \delta, \tau)$ be a fuzzy finite automaton over the two-element Boolean algebra $\mathbb{B} = \{0,1\}$ and an inpit alphabet $X = \{x,y\}$ given by

$$\sigma = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \delta_x = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \delta_y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad \tau = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

Applying formula (17) we get

$$Q_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad Q_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$Q_3 = Q_4 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

with $d(Q_0) = 2$, $d(Q_1) = 3$, $d(Q_2) = 4$ and $d(Q_3) = 5$, where $Q_3 = Q_4$ is the greatest right invariant fuzzy quasi-order matrix, and applying (21) we get

$$\widehat{Q}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad \widehat{Q}_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

with $d(\widehat{Q}_0) = 2$ and $d(\widehat{Q}_1) = 3$, where $\widehat{Q}_1$ is the greatest weakly right invariant fuzzy quasi-order matrix.

We have that $\mathcal{A}_{Q_k}$ is k-equivalent to $\mathcal{A}$, for each $k \in \{0,1,2\}$, while $\mathcal{A}_{Q_3}$ is strictly equivalent to $\mathcal{A}$, and we also have that $|\mathcal{A}_{Q_0}| < |\mathcal{A}_{Q_1}| < |\mathcal{A}_{Q_2}| < |\mathcal{A}_{Q_3}|$. On the other hand, $\mathcal{A}_{\widehat{Q}_0}$ is 0-equivalent to $\mathcal{A}$, while $\mathcal{A}_{\widehat{Q}_1}$ is strictly equivalent to $\mathcal{A}$. We see that $\widehat{Q}_1$ provides better reduction than Q_3.

The sequence computed according to formula (20) is the following:

$$P_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad P_2 = P_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

with $d(P_0) = 2$, $d(P_1) = 6$ and $d(P_2) = 6$, $P_2 = P_3$ is the greatest left invariant fuzzy quasi-order matrix, while the sequence computed using (23) is the following:

$$\widehat{P}_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \widehat{P}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \widehat{P}_2 = \widehat{P}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

with $d(\widehat{P}_0) = 2$, $d(\widehat{P}_1) = 6$ and $d(\widehat{P}_2) = 6$, while $\widehat{P}_2$ is the greatest weakly left invariant fuzzy quasi-order matrix. Although $\mathcal{A}_{P_1}$ and $\mathcal{A}_{\widehat{P}_1}$ are 1-equivalent to $\mathcal{A}$, and $\mathcal{A}_{P_2}$ and $\mathcal{A}_{\widehat{P}_2}$ are strictly equivalent to $\mathcal{A}$, this is of no significance as there is no any reduction.

5 Complexity issues

Let n denote the number of states of a fuzzy finite automaton $\mathcal{A} = (A, \sigma, \delta, \tau)$ and m the number of letters in the input alphabet X, and let $c_\vee$, $c_\wedge$, $c_\otimes$ and $c_\to$ be respectively computational costs of the operations $\vee$, $\wedge$, $\otimes$ and $\to$ in the underlying complete residuated lattice $\mathbb{L}$. If $\mathbb{L}$ is linearly ordered, we can assume that $c_\vee = c_\wedge = 1$, and if $\mathbb{L}$ is the Gödel structure, we can also assume that $c_\otimes = c_\to = 1$.

First we consider the computational time of the procedure from Theorem 4.1 (or Theorem 4.2), for a given $k \in \mathbb{N}$. It is clear that the time required to compute Q_0 is $O(n^2 c_\to)$. When we have computed Q_s, for some $s \in \mathbb{N}$, $s < k$, and we are computing Q_{s+1} from Q_s, we have the following:

1) For a fixed $x \in X$, the time required to compute the product $\delta_x \cdot Q_s$ is $O(n^3(c_\otimes + c_\vee))$, and when this product is computed, we need an additional time $O(n^3(c_\to + c_\wedge))$ to compute the residual $(\delta_x \cdot Q_s)/\delta_x$. Thus, the total time required to compute $(\delta_x \cdot Q_s)/\delta_x$ is $O(n^3(c_\to + c_\wedge + c_\otimes + c_\vee))$.
2) Now, to compute $(\delta_x \cdot Q_s)/\delta_x$ for all $x \in X$ we need time $O(mn^3(c_\to + c_\wedge + c_\otimes + c_\vee))$.
3) Next, when all matrices from 2) have been computed, to compute all infima in $Q_s \wedge \bigwedge_{x \in X}(\delta_x \cdot Q_s)/\delta_x$ and obtain Q_{s+1} we need time $O(n^2 m c_\wedge)$.
4) Finally, the total time required to compute Q_{s+1} from Q_s is $O(mn^3(c_\to + c_\wedge + c_\otimes + c_\vee))$.

Therefore, the time required to compute all matrices Q_0, Q_1, …, Q_k, i.e., the total computational time of the procedure from Theorem 4.1, amounts $O(kmn^3(c_\to + c_\wedge + c_\otimes + c_\vee))$. This is also the total computational time of the procedure from Theorem 4.2. That computational time can be even better in cases where $Q_s = Q_{s+1}$, for some $s < k$, because then we do not have to compute all the matrices between Q_{s+1} and Q_k which are all equal to each other. However, to achieve that better computational time, after computing the matrix Q_{s+1} we need to check whether $Q_{s+1} = Q_s$. The time required for such a check is $O(n^2)$, and for all such checks it is at most $O(kn^2)$, which obviously does not affect the total computational time order of our procedure.

Next we consider the computational time of the procedure from Theorem 4.3 (or Theorem 4.4), for a given $k \in \mathbb{N}$. To compute $\widehat{Q}_k$ we have to compute the family of vectors $\{\tau_u\}_{|u| \leqslant k}$. That family forms a perfect m-ary tree with the root corresponding to the vector τ, and computing the members of the family is reduced to filling that tree level by level, starting from the root. Consequently, the number of members of that family is at most $O(m^k)$. When a vector τ_u, for some $u \in X^*$, $|u| < k$, is computed, then for each $x \in X$ we compute τ_{xu} according to the formula $\tau_{xu} = \delta_x \cdot \tau_u$. Therefore, the time needed to compute this product, i.e., the time nedded to compute each member of the considered family, is $O(n^2(c_\otimes + c_\vee))$. Moreover, when τ_u, for some $u \in X^+$, $|u| \leqslant k$, is computed, the residual τ_u/τ_u can be computed in time $O(n^2 c_\to)$, so the time required to compute a single τ_u and the residual τ_u/τ_u is $O(n^2(c_\otimes + c_\vee + c_\to))$, and the total time required to compute the whole family $\{\tau_u\}_{|u| \leqslant k}$ and all residuals τ_u/τ_u, for $u \in X^*$, $|u| \leqslant k$, amounts $O(m^k n^2(c_\otimes + c_\vee + c_\to))$. Finally, to compute $\widehat{Q}_k$ we have to apply the operation $\wedge$ between matrices τ_u/τ_u, $u \in X^*$, $|u| \leqslant k$, at most $O(m^k)$ times, and since the computational time for a single application of the operation $\wedge$ is $O(n^2 c_\wedge)$, the total computational time for all such operations is $O(m^k n^2 c_\wedge)$. Hence, the total time required to compute $\widehat{Q}_k$, i.e., the total computational time of the procedure from Theorem 4.3 is $O(m^k n^2(c_\otimes + c_\vee + c_\to + c_\wedge))$. This is also the total computational time of the procedure from Theorem 4.4.

Applying the technique of [12], the procedures from Theorems 4.1–4.4 can be improved so that the factor n^2 in their complexity estimate is reduced to the sum of n and the number of non-zero fuzzy transitions of the input fuzzy automaton $\mathcal{A}$.

References

[1] Radim Bělohlávek & Vilém Vychodil (2005): *Fuzzy Equational Logic*. In: *Fuzzy Equational Logic, Studies in Fuzziness and Soft Computing* 186, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 139–170, doi:10.1007/11376422_3.

[2] Radim Bělohlávek (2002): *Fuzzy Relational Systems: Foundations and Principles*. Kluwer Academic Publishers, USA, doi:10.1007/978-1-4615-0633-1.

[3] Radim Bělohlávek & Michal Krupka (2009): *On Approximate Minimization of Fuzzy Automata*. *Journal of Multiple-Valued Logic and Soft Computing* 15(2-3), pp. 125–135, doi:10.1142/9789812709677_0194.

[4] Miroslav Ćirić, Aleksandar Stamenković, Jelena Ignjatović & Tatjana Petković (2007): *Factorization of Fuzzy Automata*. In Erzsébet Csuhaj-Varjú & Zoltán Ésik, editors: *Fundamentals of Computation Theory, Lecture Notes in Computer Science* 4639, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 213–225, doi:10.1007/978-3-540-74240-1_19.

[5] Miroslav Ćirić, Aleksandar Stamenković, Jelena Ignjatović & Tatjana Petković (2010): *Fuzzy relation equations and reduction of fuzzy automata*. *Journal of Computer and System Sciences* 76, pp. 609–633, doi:10.1016/j.jcss.2009.10.015.

[6] Jelena Ignjatović, Miroslav Ćirić, Branimir Šešelja & Andreja Tepavčević (2015): *Fuzzy relational inequalities and equations, fuzzy quasi-orders, closures and openings of fuzzy sets*. *Fuzzy Sets and Systems* 260, pp. 1–24, doi:10.1016/j.fss.2014.05.006.

[7] Lvzhou Li & Daowen Qiu (2015): *On the State Minimization of Fuzzy Automata*. *IEEE Transactions on Fuzzy Systems* 23(2), pp. 434–443, doi:10.1109/TFUZZ.2014.2315620.

[8] Yongming Li (2008): *Approximation and robustness of fuzzy finite automata*. *International Journal of Approximate Reasoning* 47(2), pp. 247–257, doi:10.1016/j.ijar.2007.05.004.

[9] Ivana Micić, Zorana Jančić & Stefan Stanimirović (2022): *Computation of solutions to certain nonlinear systems of fuzzy relation inequations*. In Dimitros Poulakis & George Rahonis, editors: *Algebraic Informatics, 9th International Conference, CAI 2022, Lecture Notes in Computer Science* 13706, Thessaloniki, Greece, pp. 192–202, doi:10.1007/978-3-031-19685-0_14.

[10] Ivana Micić, Linh Anh Nguyen & Stefan Stanimirović (2022): *Characterization and computation of approximate bisimulations for fuzzy automata*. *Fuzzy Sets and Systems* 442, pp. 331–350, doi:10.1016/j.fss.2022.05.003.

[11] Linh Anh Nguyen (2023): *Fuzzy simulations and bisimulations between fuzzy automata*. *International Journal of Approximate Reasoning*, pp. 113–131, doi:10.1016/j.ijar.2023.02.002.

[12] Linh Anh Nguyen, Ivana Micić & Stefan Stanimirović (2023): *Depth-Bounded Fuzzy Simulations and Bisimulations between Fuzzy Automata*, doi:10.48550/arXiv.2307.03318. arXiv:2307.03318.

[13] Linh Anh Nguyen, Ivana Micić & Stefan Stanimirović (2023): *Fuzzy minimax nets*. *IEEE Transactions on Fuzzy Systems* 31(8), pp. 2799–2808, doi:10.1109/TFUZZ.2023.3237936.

[14] Sha Qiao, Ping Zhu & Jun e Feng (2022): *Fuzzy bisimulations for nondeterministic fuzzy transition systems*. *IEEE Transactions on Fuzzy Systems* 31(7), pp. 2450–2463, doi:10.1109/TFUZZ.2022.3227400.

[15] Sha Qiao, Ping Zhu & Witold Pedrycz (2023): *Approximate bisimulations for fuzzy-transition systems*. *Fuzzy Sets and Systems*, p. 108533, doi:10.1016/j.fss.2023.108533.

[16] Aleksandar Stamenković, Miroslav Ćirić & Milan Bašić (2018): *Ranks of fuzzy matrices. Applications in state reduction of fuzzy automata*. *Fuzzy Sets and Systems* 333, pp. 124–139, doi:10.1016/j.fss.2017.05.028.

[17] Aleksandar Stamenković, Miroslav Ćirić & Jelena Ignjatović (2014): *Reduction of fuzzy automata by means of fuzzy quasi-orders*. *Information Sciences* 275, pp. 168–198, doi:10.1016/j.ins.2014.02.028.

[18] Stefan Stanimirović & Ivana Micić (2022): *On the solvability of weakly linear systems of fuzzy relation equations*. *Information Sciences* 607, pp. 670–687, doi:10.1016/j.ins.2022.05.111.

[19] Stefan Stanimirović, Ivana Micić & Miroslav Ćirić (2022): *Approximate bisimulations for fuzzy automata over complete Heyting algebras*. IEEE Transactions on Fuzzy Systems 30(2), pp. 437–447, doi:10.1109/TFUZZ.2020.3039968.

[20] Chao Yang & Yongming Li (2018): *ε-bisimulation relations for fuzzy automata*. IEEE Transactions on Fuzzy Systems 26(4), pp. 2017–2029, doi:10.1109/TFUZZ.2017.2760278.

[21] Chao Yang & Yongming Li (2020): *Approximate bisimulations and state reduction of fuzzy automata under fuzzy similarity measures*. Fuzzy Sets and Systems 391, pp. 72–95, doi:10.1016/j.fss.2019.07.010.

Weighted Automata over Vector Spaces*

Nada Damljanović
University of Kragujevac, Faculty of Technical Sciences, Svetog Save 65, Čačak, Serbia
nada.damljanovic@ftn.kg.ac.rs

Miroslav Ćirić Jelena Ignjatović
University of Niš, Faculty of Sciences and Mathematics, Višegradska 33, Niš, Serbia
miroslav.ciric@pmf.edu.rs jelena.ignjatovic@pmf.edu.rs

In this paper we deal with three models of weighted automata that take weights in the field of real numbers. The first of these models are classical weighted finite automata, the second one are crisp-deterministic weighted automata, and the third one are weighted automata over a vector space. We explore the interrelationships between weighted automata over a vector space and other two models.

1 Introduction

Weighted automata belong to the fundamental models of computation in computer science. They can be understood as an extension of conventional automata in which the transitions and states carry numerical or other values called weights. These weights may model quantitative properties like the cost, the amount of resources needed for the execution of a transition, the reliability or probability of the successful execution of the transitions, or many other things. Different models of weighted automata differ in the algebraic structures within which the weights are taken, as well as in the way in which these weights are manipulated.

In this paper, we deal with weighted automata that take weights in the field of real numbers. Such automata have been the subject of study since the very beginning of the theory of weighted automata, since the seminal work of Schützenberger [23] who studied weighted automata over the field. Today, they are very popular due to their significant applications, primarily in formal specification and verification of systems, as well as in the field of machine learning, where they are successfully used as an alternative to recurrent neural networks. We discuss three models of weighted automata with weights taken in the field of real numbers.

The first of these models are classical *weighted finite automata*. The common way of viewing deterministic and nondeterministic finite automata as labelled graphs has also been used for weighted finite automata from the very beginning of their studying. From such a point of view, a weighted finite automaton is represented by a directed multi-graph whose edges carry two labells, the input letter and the weight, while nodes carry two weights, the initial and terminal weight. The computation along a path in the graph is performed by concatenation of the input letters and multiplication of the initial weight of the starting node, the weights of edges along the path, and the terminal weight of the final node, and then the sum of the weights of all paths labelled with the same input word is computed and assigned to this input word. This determines the behavior of the considered weighted finite automaton, that is, the word function computed by that automaton. Such an understanding of the behaviour can be called

*This research was supported by the Science Fund of the Republic of Serbia, Grant no 7750185, Quantitative Automata Models: Fundamental Problems and Applications - QUAM

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 67–81, doi:10.4204/EPTCS.386.7

the *dept-first semantics*. Another way of looking at weighted finite automata, through vector and matrix operations, has also been present since their very beginnings. From that point of view, the behaviour of a weighted finite automaton can be expressed as the product of the row vector representing the initial weights, matrices representing the weights of the transitions induced by input letters, and the column vector representing the terminal weights. Such a representation of a weighted finite automaton is called a *linear representation*, while such an understanding of the behaviour can be called the *breadth-first semantics*. In the case of weighted finite automata over a semiring these two semantics are the same. The linear algebraic approach proved to be extremely powerful and useful, especially in the study of simulations and bisimulations, as well as in the reduction of the number of states. That approach was successfully applied to nondeterministic finite automata [8], fuzzy finite automata [9, 10, 11, 25, 24] and weighted finite automata over an additively-idempotent semiring [13], and research is underway in which that approach is applied in the context of weighted automata over the max-plus semiring and the field of real numbers. The mentioned approach also plays a key role in this paper.

The second model of weighted automata that we deal with here are the so-called *crisp-deterministic weighted automata*. These are classical automata with a single initial state and deterministic transitions in which the set of terminal states is replaced by a function which assigns a terminal weight to each state. When such an automaton starts working from the initial state and performs a sequence of transitions conducted by a given input word, the weight assigned to that word is the terminal weight of the destination state. Those automata were studied for the first time in [16], in the context of fuzzy automata, and the most general definition of crisp-deterministic weighted automata was given in [17]. The name crisp-deterministic was introduced in [7] to distinguish it from a related type of automata for which the name deterministic weighted automata is used. An extensive study of crisp-deterministic weighted automata was carried out in [17], and in [7, 16, 18, 19, 20, 22] various procedures for converting a weighted finite automaton into an equivalent crisp-deterministic weighted automaton were provided. Such procedures are called *crisp-determinization*. If we allow a crisp-deterministic weighted automaton to have an infinite set of states, as we do in this paper, then any weighted finite automaton can be converted into an equivalent crisp-deterministic weighted automaton, and the basic problem is to perform such a conversion that will provide an equivalent crisp-deterministic weighted automaton with a finite number of states, as small as possible. For information on crisp-determinization of weighted tree automata we refer to [14, 15].

The main role in the crisp-determinization is played by the concept of the *Nerode automaton* assigned to the weighted finite automaton that is determinized. The construction of the Nerode automaton was first introduced in [16] as a counterpart to the *accessible subset construction* on which the determinization of classical nondeterministic finite automata is based. According to that construction, the states of a Nerode automaton are vectors with entries from the underlying structure of weights, but in the mentioned papers dealing with crisp-determinization, such nature of states was neglected, and the Nerode automaton was considered as an ordinary crisp-deterministic weighted automaton. If the vector nature of states is taken into account, this leads us to the third model of weighted automata that is considered here, to *weighted automata over a vector space* or *weighted automata with vector states*. Various forms of such automata were studied in [2, 3, 4, 5, 12, 21], and a related model of automata, called *automata with fuzzy states*, was studied within the framework of fuzzy automata theory (see [26] and sources cited there). The concept of a weighted automaton over a vector space discussed here differs slightly from the corresponding concepts studied in the cited articles. The first difference concerns the underlying vector space. Except in [12], in all the other mentioned articles, it is assumed that this vector space is finite-dimensional. Here we not only allow that space to be infinite-dimensional, but also introduce an extremely interesting weighted automaton over an infinite-dimensional space, the so-called *derivative*

automaton. The second difference concerns the set of states of these automata. In all the mentioned articles, except in [4], states are assumed to be all vectors from the underlying vector space V. However, in that case the set of states is always infinite and a huge number of states are unreachable from the initial state, and therefore redundant. For this reason, we take the set of states to be a subset of V, which can be both finite and infinite. The third difference relates to transition functions. In almost all cited articles, the transition functions induced by the input letters were required to be linear operators on V. In [21], a more general model of weighted automata over a vector space was proposed, where the transition functions do not have to be linear. This leads to the distinction between *linear* and *nonlinear* weighted automata over a vector space. Here we give a definition of a linear weighted automaton over a vector space which also includes the cases when the set of states is not the entire vector space and when the underlying vector space is infinite-dimensional.

This paper is the beginning of our extensive investigations of weighted automata with weights taken in the field of real numbers, and our aim here is to examine some general relations between weighted automata over vector spaces and other two models. First, by Theorem 4.1, we show that any crisp-deterministic weighted automaton can be naturally turned into a language-equivalent weighted automaton over a vector space, where the set of vector states can be any set of vectors that has the same cardinality as the set of states of that crisp-deterministic weighted automaton. Then by Theorem 4.2 we show that any finite-dimensional linear weighted automaton over a vector space can be turned into a completely language-equivalent weighted finite automaton, and conversely, any weighted finite automaton can be turned into a completely language-equivalent finite-dimensional linear weighted automaton over a vector space. Actually, we show that the previously mentioned Nerode automaton of a weighted finite automaton $\mathcal{A}$ is a finite-dimensional linear weighted automaton over a vector space that is completely equivalent to $\mathcal{A}$. Theorem 4.3 gives us an elegant procedure for checking whether a given finite weighted automaton over a vector space is linear. At the end of the paper, we introduce the concept of the derivative automaton of a given word function and prove that it is a linear weighted automaton over a vector space that computes this word function and generates its prefix closure. In addition, we show that the derivative automaton is a minimal weighted automaton over a vector space which computes this word function.

2 Preliminaries

Throughout this paper, $\mathbb{N}$ denotes the set of all natural numbers (without zero) and $\mathbb{R}$ denotes the field of real numbers. For $i, j \in \mathbb{N}$ such that $i \leq j$ we use the notation $[i..j] = \{k \in \mathbb{N} \mid i \leq k \leq j\}$.

A *vector space* over $\mathbb{R}$ is a triple $(V, +, \cdot)$ such that:

- V is a non-empty set, whose members are called *vectors*;
- $+ : V \times V \to V$ given by $+ : (\alpha, \beta) \mapsto \alpha + \beta$, for $\alpha, \beta \in V$, is the *vector addition* operation;
- $\cdot : \mathbb{R} \times V \to V$ given by $\cdot : (r, \alpha) \mapsto r \cdot \alpha$, for $r \in \mathbb{R}$, $\alpha \in V$, is the *scalar multiplication* operation;
- vector addition and scalar multiplication satisfy the following axioms:

  (V1) $(V, +)$ is a commutative group,
  (V2) $r \cdot (\alpha + \beta) = r \cdot \alpha + r \cdot \beta$,
  (V3) $(r + s) \cdot \alpha = r \cdot \alpha + s \cdot \alpha$,
  (V4) $(r \cdot s) \cdot \alpha = r \cdot (s \cdot \alpha)$,
  (V5) $1 \cdot \alpha = \alpha$,
  for all $r, s \in \mathbb{R}$ and $\alpha, \beta \in V$.

Note that a vector space over an arbitrary field can be defined in the same way.

The basic example of a vector space over $\mathbb{R}$ is the vector space $\mathbb{R}^n$ consisting of all n-tuples of real numbers, with vector addition and scalar multiplication defined coordinatewise. Another example of a vector space over $\mathbb{R}$ that is important here is the vector space $\mathbb{R}^T$ consisting of all functions from a set T into $\mathbb{R}$, with vector addition and scalar multiplication defined by $(\alpha+\beta)(t) = \alpha(t)+\beta(t)$ and $(r\cdot\alpha)(t) = r\cdot\alpha(t)$, for all $\alpha,\beta \in \mathbb{R}^T$, $r \in \mathbb{R}$ and $t \in T$. Such vector spaces are called *function spaces*.

Let V and W be vector spaces over $\mathbb{R}$. A function $h : V \to W$ is called a *homomorphism* or *linear transformation* of V into W if $h(\alpha+\beta) = h(\alpha)+h(\beta)$ and $h(r\cdot\alpha) = r\cdot h(\alpha)$, for all $\alpha,\beta \in V$ and $r \in \mathbb{R}$. If h is a bijective homomorphism, then it is called an *isomorphism* of V into W, and we say that V and W are *isomorphic* vector spaces. A vector space V over $\mathbb{R}$ is said to be *finite-dimensional* if it is isomorphic to the vector space $\mathbb{R}^n$, for some $n \in \mathbb{N}$. In this case n is the unique natural number having this property and it is called the *dimension* of V. A vector space which is not finite-dimensional is called *inifinite-dimensional*. A homomorphism (linear transformation) of a vector space V into itself is called a *linear operator* on V.

Let V be a vector space over $\mathbb{R}$. A *linear combination* of vectors $\alpha_1,\alpha_2,\ldots,\alpha_k \in V$ is any expression of the form $r_1\cdot\alpha_1 + r_2\cdot\alpha_2 + \cdots + r_k\cdot\alpha_k$, where $r_1,r_2,\ldots,r_k \in \mathbb{R}$. For any set $S \subseteq V$, the set of all linear combinations of vectors from S is called the *span* of S and denoted by $\mathrm{span}(S)$. In other words,

$$\mathrm{span}(S) = \{\alpha \in V \mid (\exists k \in \mathbb{N})(\exists \alpha_1,\alpha_2,\ldots,\alpha_k \in S)(\exists r_1,r_2,\ldots,r_k \in \mathbb{R})\, \alpha = r_1\cdot\alpha_1 + r_2\cdot\alpha_2 + \cdots + r_k\cdot\alpha_k\}.$$

It is well-known that $\mathrm{span}(S)$ is a vector space with vector addition and scalar multiplication inherited from V, i.e., it is a *subspace* of V.

Given natural numbers $m,n \in \mathbb{N}$. A *matrix* of type $m \times n$ with entries in the field of real numbers $\mathbb{R}$, or a real $m \times n$-matrix, is defined as a mapping $M : [1..m] \times [1..n] \to \mathbb{R}$. For a pair $(i,j) \in [1..m] \times [1..n]$ the value $M(i,j)$ is called the (i,j)-entry of the matrix M. The set of all real matrices of type $m \times n$ is denoted by $\mathbb{R}^{m\times n}$. Similarly, a *vector* of length n with entries in $\mathbb{R}$, or real vector is defined as a mapping $v : [1..n] \to \mathbb{R}$. For each $i \in [1..n]$ the value $v(i)$ is called the ith entry or ith coordinate of the vector v. The set of all real vectors of length n is denoted by $\mathbb{R}^n$.

The zero matrix of type $m \times n$, denoted by $O_{m\times n}$, is a matrix of type $m \times n$ whose all entries are 0. Similarly, the zero vector of length n, denoted by o_n, is a vector of length n whose all entries are 0. For each $n \in \mathbb{N}$, a matrix of type $n \times n$ is called a *square matrix* of order n. The identity matrix of order n, denoted by I_n, is a square matrix of order n which satisfies $I_n(i,i) = 1$, for each $i \in [1..n]$, and $I_n(i,j) = 0$, for all $i,j \in [1..n]$ such that $i \neq j$. The transpose of a matrix M is denoted by $M^\top$. For a matrix $M \in \mathbb{R}^{m\times n}$ and $k \in [1..n]$, by $c_k(M)$ we denote the kth column vector of M.

For all pairs of matrices from $\mathbb{R}^{m\times n}$ the *matrix addition* is defined pointwise:

$$(M+N)(i,j) = M(i,j)+N(i,j), \tag{1}$$

for all $M,N \in \mathbb{R}^{m\times n}, i \in [1..m]$ and $j \in [1..n]$. It is an associative and commutative operation on $\mathbb{R}^{m\times n}$, and in particular, $(\mathbb{R}^{m\times n},+,O_{m\times n})$ forms a commutative monoid. The *matrix product* is defined between matrices from $\mathbb{R}^{m\times n}$ and $\mathbb{R}^{n\times p}$ as follows: for $M \in \mathbb{R}^{m\times n}$ and $N \in \mathbb{R}^{n\times p}$ their product is a matrix $M\cdot N \in \mathbb{R}^{m\times p}$ with entries given by

$$(M\cdot N)(i,k) = \sum_{j=1}^{n} M(i,j)\cdot M(j,k), \tag{2}$$

for all $(i,j) \in [1..m] \times [1..p]$. The matrix product is associative whenever it is defined, i.e., $(M\cdot N)\cdot P = M\cdot(N\cdot P)$, for all $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$ and $P \in \mathbb{R}^{p\times q}$. In particular, $(\mathbb{R}^{n\times n},+,\cdot,O_{n\times n},I_n)$ is a semiring.

Given a matrix $M \in \mathbb{R}^{m\times n}$ and vectors $\mu \in \mathbb{R}^m$ and $\nu \in \mathbb{R}^n$. When μ is treated as a matrix of type $1 \times m$ (row vector) and ν as a matrix of type $n \times 1$ (column vector), the *vector-matrix product* $\mu \cdot M$ and the *matrix-vector product* $M \cdot \nu$ are defined as matrix products. The *scalar product* or *dot product* of vectors $\mu, \nu \in \mathbb{R}^n$ is an element of $\mathbb{R}$ given by

$$\mu \cdot \nu = \sum_{i=1}^{n} \mu(i) \cdot \nu(i). \tag{3}$$

A matrix $M \in \mathbb{R}^{m\times n}$ is said to be in the *row echelon form* if it satisfies the following properties:

- If a row of M does not consist entirely of zeros, then the first nonzero entry in this row is 1. It is called a *leading* 1.
- If there are any rows that consist entirely of zeros, then they are grouped together at the bottom of the matrix M.
- In any two successive rows of M that do not consist entirely of zeros, the leading 1 in the lower row occurs farther to the right than the leading 1 in the higher row.

Moreover, M is said to be in the *reduced row echelon form* if, in addition to these three properties, it also satisfies the condition

- Every column of M that contains a leading 1 has zeros everywhere else.

Every matrix $N \in \mathbb{R}^{m\times n}$ can be transformed to a row echelon form or a reduced row echelon form by applying some sequence of *elementary row operations* (multiplying a row by a nonzero scalar, interchanging two rows, and adding a multiple of one row to another). It should be noted that the reduced row echelon form of the matrix N is unique, in the sense that reducing the matrix N to the reduced row echelon form by applying any sequence of elementary row operations always yields the same matrix in the reduced row echelon form. This matrix will be denoted by $RREF(N)$. The *rank* of a matrix N, denoted by $\mathrm{rank}(N)$, is defined as the number of nonzero rows in $RREF(N)$.

For matrices $M_1 \in \mathbb{R}^{m\times n_1}$, $M_2 \in \mathbb{R}^{m\times n_2}$, $\ldots$, $M_k \in \mathbb{R}^{m\times n_k}$, where $k, m, n_1, n_2, \ldots, n_k \in \mathbb{N}$, by concatenating them from left to right we obtain a matrix $[M_1 \mid M_2 \mid \ldots \mid M_k] \in \mathbb{R}^{m\times n}$, where $n = n_1 + n_2 + \cdots + n_k$, which is called the *augmented matrix* (obtained from $M_1, M_2, \ldots, M_k$).

For undefined notions and notation concerning vector spaces, vectors and matrices we refer to the book [1].

3 Three models of weighted automata

In terms of real matrices and their properties, we will investigate three models of weighted automata with weights in the field of real numbers.

3.1 Weighted finite automata

Let X be an alphabet. A *weighted finite automaton* over $\mathbb{R}$ and X is defined as a tuple $\mathcal{A} = (A, \sigma, \delta, \tau)$, where A is a non-empty finite set, while $\sigma, \tau : A \to \mathbb{R}$ and $\delta : A \times X \times A \to \mathbb{R}$. The function δ is often replaced by the family of functions $\{\delta_x\}_{x\in X}$, where $\delta_x : A \times A \to \mathbb{R}$ is given by $\delta_x(a,b) = \delta(a,x,b)$, for all $a, b \in A$ and $x \in X$. We call A the *set of states*, σ the *initial weights function*, τ the *terminal weights function*, and δ and δ_x, $x \in X$, the *transition weights functions*.

The behavior of the weighted finite automaton $\mathcal{A}$ is a word function $[\![\mathcal{A}]\!] : X^* \to \mathbb{R}$ defined by

$$[\![\mathcal{A}]\!](u) = \sum_{(a_0,a_1,\ldots,a_k)\in A^{k+1}} \sigma(a_0)\cdot\delta(a_0,x_1,a_1)\cdot\delta(a_1,x_2,a_2)\cdots\delta(a_{k-1},x_k,a_k)\cdot\tau(a_k), \tag{4}$$

for $u = x_1x_2\ldots x_k \in X^+$, $x_1,x_2,\ldots,x_k \in X$, and

$$[\![\mathcal{A}]\!](\varepsilon) = \sum_{a\in A} \sigma(a)\cdot\tau(a). \tag{5}$$

We say that $\mathcal{A}$ *computes* the function $[\![\mathcal{A}]\!]$.

Assume that n is the number of elements of A, i.e., $A = \{a_1,a_2,\ldots,a_n\}$. In many situations, instead of as functions, it is more convenient to treat σ and τ as vectors in $\mathbb{R}^n$, and δ_x, $x \in X$, as $n\times n$ matrices with entries in $\mathbb{R}$. In other words, σ will be identified with a vector in $\mathbb{R}^n$ whose ith coordinate is $\sigma(a_i)$, and τ will be identified with a vector in $\mathbb{R}^n$ whose ith coordinate is $\tau(a_i)$. In order to clearly distinguish between matrices and vectors, we will use capital letters of the Latin alphabet to denote matrices, while vectors will be denoted by small letters of the Greek alphabet. Therefore, $\{M_x\}_{x\in X}$ will be a family of $n\times n$ matrices over $\mathbb{R}$ such that the (i,j)-entry of M_x is equal to $\delta_x(a_i,a_j)$. A weighted finite automaton $\mathcal{A}$ is then treated as a tuple $\mathcal{A} = (n,\sigma,\{M_x\}_{x\in X},\tau)$, where we cal n the *dimension*, σ the *initial weights vector*, τ the *terminal weights vector*, and M_x, $x \in X$, the *transition weights matrices* of $\mathcal{A}$. We cal this tuple the *linear representation* of the weighted finite automaton $\mathcal{A}$.

When $\mathcal{A}$ is given by the linear representation, its behavior is represented by

$$[\![\mathcal{A}]\!](u) = \sigma\cdot M_{x_1}\cdot M_{x_2}\cdots M_{x_k}\cdot\tau = \sigma\cdot M_u\cdot\tau, \tag{6}$$

for $u = x_1x_2\ldots x_k \in X^+$, $x_1,x_2,\ldots,x_k \in X$, where $M_u = M_{x_1}\cdot M_{x_2}\cdots M_{x_k}$, and

$$[\![\mathcal{A}]\!](\varepsilon) = \sigma\cdot\tau. \tag{7}$$

In applications of automata in the theory of discrete event systems, apart from the function $[\![\mathcal{A}]\!]$ computed by the automaton $\mathcal{A}$, another function plays an important role – the function $[\![\mathcal{A}]\!]_g$ generated by the automaton $\mathcal{A}$. For a weighted finite automaton $\mathcal{A} = (n,\sigma,\{M_x\}_{x\in X},\tau)$ this function can be defined with:

$$[\![\mathcal{A}]\!]_g(u) = \|\sigma_u\|_\infty, \tag{8}$$

for every $u \in X^*$, where $\sigma_u = \sigma\cdot M_u$, for $u \in X^+$, and $\sigma_\varepsilon = \sigma$, while $\|\cdot\|_\infty$ denotes the *maximum norm* (called also a *uniform norm*) on $\mathbb{R}^n$ that is given by

$$\|\alpha\|_\infty = \max_{i\in[1..n]} |\alpha_i|, \tag{9}$$

for each $\alpha = (\alpha_1,\alpha_2,\ldots,\alpha_n) \in \mathbb{R}^n$.

Let us note that in the case of convential nondeterministic finite automata, $[\![\mathcal{A}]\!]_g$ is the language consisting of all words for which a transition is defined, i.e., of all words which "lead somewhere"(cf. [6]). A nondeterministic finite automaton can be considered as a weighted finite automaton over the two-element Boolean semiring, and then $[\![\mathcal{A}]\!]_g$ consists of all words u for which σ_u is a non-zero Boolean vector, i.e., for which

$$\max_{i\in[1..n]} |\alpha_i| = 1,$$

where $\sigma_u = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \{0,1\}^n$. From conventional nondeterministic finite automata, such a definition was also extended to the case of fuzzy finite automata, where $[\![\mathcal{A}]\!]_g(u)$ is interpreted as the degree to which the word u leads somewhere. A similar interpretation can be given here as well, when it comes to weighted finite automata over the field of real numbers, where $[\![\mathcal{A}]\!]_g(u) = \|\sigma_u\|_\infty$ could be interpreted as the maximal probability of the existence of a transition determined by u, i.e. the probability that u leads somewhere. In order for this to be consistent with the conventional view of probability, the values for $\|\sigma_u\|_\infty$, which are always non-negative, can be translated by some monotone function (for example, by the function $1 - e^{-x}$), to values from the real unit interval $[0,1]$.

3.2 Crisp-deterministic weighted automata

Another model of weighted automata is a *crisp-deterministic weighted automaton*, which is defined as a tuple $\mathcal{D} = (D, d_0, \Delta, \theta)$, where D is a non-empty *set of states*, $d_0 \in D$ is a distinguished state that is called the *initial state*, $\Delta : D \times X \to D$ is a function called the *transition function*, and $\theta : D \to \mathbb{R}$ is a function called the *terminal weights function*, or the *terminal weights vector*, if θ is considered as a vector in the space $\mathbb{R}^D$. Here it is not necessary that the set D is finite, so we will also allow the possibility that D is infinite. If the set of states D is finite, then $\mathcal{D}$ is called a *finite crisp-deterministic weighted automaton*

A finite crisp-deterministic weighted automaton can be considered as a special weighted finite automaton $\mathcal{A} = (D, \sigma, \delta, \tau)$ in which for every $a \in D$ and $x \in X$ there exists $a' = \Delta(a,x) \in D$ such that $\delta(a,x,a') = 1$, while $\delta(a,x,b) = 0$, for every $b \in D \setminus \{a'\}$, and there exists $a \in D$ such that $\sigma(a) = 1$, while $\sigma(b) = 0$, for every $b \in D \setminus \{a\}$ (then we assume that $d_0 = a$). In other words, a finite crisp-deterministic weighted automaton is a weighted finite automaton with a single initial state and a deterministic transition function, in which all weights are concentrated in the terminal weights vector.

The transition function Δ of a crisp-deterministic weighted automaton $\mathcal{D} = (D, d_0, \Delta, \theta)$ extends to a function $\Delta^* : D \times X^* \to D$ by putting $\Delta^*(a, \varepsilon) = a$ and $\Delta^*(a, ux) = \Delta(\Delta^*(a,u), x)$, for all $a \in D$, $u \in X^*$ and $x \in X$, and the behavior $[\![\mathcal{D}]\!] : X^* \to \mathbb{R}$ of $\mathcal{D}$ is given by

$$[\![\mathcal{D}]\!](u) = \theta(\Delta^*(d_0, u)), \tag{10}$$

for every $u \in X^*$.

From the transition function $\Delta^* : D \times X^* \to D$ we can extract a family of functions $\{\Delta_u\}_{u \in X^*}$, where $\Delta_u : D \to D$ is defined by $\Delta_u(a) = \Delta^*(a,u)$, for all $u \in X^*$ and $a \in D$. These functions will be also called *transition functions*. If $u = x_1 x_2 \ldots x_k$, for $x_1, x_2, \ldots, x_k \in X$, then

$$\Delta_u(a) = \Delta_{x_k}(\ldots(\Delta_{x_2}(\Delta_{x_1}(a)))),$$

for every $a \in D$, that is, Δ_u is the composition $\Delta_u = \Delta_{x_1}\Delta_{x_2}\cdots\Delta_{x_k}$ of transition functions $\Delta_{x_1}, \Delta_{x_2}, \ldots, \Delta_{x_k}$.

3.3 Weighted automata over a vector space

Let V be a vector space over the field $\mathbb{R}$ of real numbers. The third model of weighted automata is a *weighted automaton over a vector space* (*with vector states*), defined as a tuple $\mathcal{A} = (S, \sigma, \delta, \Theta)$, where $S \subseteq V$ is a nonempty set of vectors, called the *set of vector states*, $\sigma \in S$ is a vector called the *initial vector state*, $\delta : S \times X \to S$ is a deterministic *transition function*, and $\Theta : S \to \mathbb{R}$ is a function called the *terminal weights function*. Here we also allow S to be infinite. Furthermore, in some sources S is taken to be the entire space V, but here we allow S to be only a subset of the space V to enable it to be finite. If the set S of vector states is finite, then $\mathcal{A}$ is called a *finite weighted automaton over a vector space*, and

if V is a finite-dimensional vector space of dimension n, then $\mathcal{A}$ is also said to be a *finite-dimensional weighted automaton over a vector space of dimension n*. If the cardinality of the set of states of $\mathcal{A}$ is less than or equal to the cardinality of the set of states of any other weighted automaton over the vector space V, that we say that $\mathcal{A}$ is a *minimal weighted automaton over the vector space V*.

As with crisp-deterministic weighted automata, δ can be extended to a function $\delta^* : S \times X^* \to S$ by $\delta^*(\alpha, \varepsilon) = \alpha$ and $\delta^*(\alpha, ux) = \delta(\delta^*(\alpha, u), x)$, for all $\alpha \in S$, $u \in X^*$, $x \in X$, and the function δ^* determines a family of functions $\{\delta_u\}_{u \in X^*}$, where $\delta_u : S \to S$ is defined by $\delta_u(\alpha) = \delta^*(\alpha, u)$, for all $u \in X^*$ and $\alpha \in S$. These functions are also called *transition functions*. If $u = x_1 x_2 \dots x_k$, for $x_1, x_2, \dots, x_k \in X$, then

$$\delta_u(\alpha) = \delta_{x_k}(\dots(\delta_{x_2}(\delta_{x_1}(\alpha)))),$$

for every $\alpha \in V$, that is, δ_u is the composition $\delta_u = \delta_{x_1}\delta_{x_2}\cdots\delta_{x_k}$ of transition functions $\delta_{x_1}, \delta_{x_2}, \dots, \delta_{x_k}$. Then $\mathcal{A}$ can be equivalently represented as a tuple $\mathcal{A} = (S, \sigma, \{\delta_x\}_{x \in X}, \Theta)$. In addition, for every $u \in X^*$, with σ_u we denote a vector from S given by $\sigma_u = \delta^*(\sigma, u) = \delta_u(\sigma)$.

The behavior $[\![\mathcal{A}]\!] : X^* \to \mathbb{R}$ of the weighted automaton $\mathcal{A}$ over a vector space V is given by

$$[\![\mathcal{A}]\!](u) = \Theta(\delta^*(\sigma, u)), \tag{11}$$

for every $u \in X^*$. On the other hand, the function $[\![\mathcal{A}]\!]_g$ generated by $\mathcal{A}$ is defined with

$$[\![\mathcal{A}]\!]_g(u) = \|\delta^*(\sigma, u)\| = \|\delta_u(\sigma)\|, \tag{12}$$

for each $u \in X^*$, where $\|\cdot\|$ denotes some norm on the vector space V. If V is a finite-dimensional space we will assume that $\|\cdot\|$ is the maximum norm $\|\cdot\|_\infty$ (see (9)), and if $V = \mathbb{R}^T$ is some function space (later we will consider the function space $\mathbb{R}^{X^*}$), then we will assume that $\|\cdot\|$ is the *supremum norm* $\|\cdot\|_\infty$ (also called the *uniform norm*) that is given by

$$\|f\|_\infty = \sup_{t \in T} |f(t)|,$$

for every $f : T \to \mathbb{R}$ (clearly, for $T = [1..n]$ we obtain (9), i.e., the maximum norm).

4 Relationships between different types of weighted automata

Let $\mathcal{A}$ and $\mathcal{B}$ be weighted automata of any of the three types discussed in the previous section, where they can be of different types. If $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$, then $\mathcal{A}$ and $\mathcal{B}$ are said to be *language-equivalent*. On the other hand, if each of these automata is a weighted finite automaton or a weighted automaton over a vector space, where they do not have to be of the same type, and if $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$ and $[\![\mathcal{A}]\!]_g = [\![\mathcal{B}]\!]_g$, then $\mathcal{A}$ and $\mathcal{B}$ are said to be *completely language-equivalent*.

Theorem 4.1 *Let $\mathcal{D} = (D, d_0, \Delta, \theta)$ be a crisp-deterministic weighted automaton over $\mathbb{R}$, let V be a vector space and let $S \subseteq V$ be a set of vectors which has the same cardinality as D.*

Then $\mathcal{D}$ can be turned into a language-equivalent weighted automaton over the vector space V having S as its set of states.

Proof. Let $\phi : S \to D$ be an arbitrary bijective function between S and D. Then we can define an initial vector state $\sigma \in S$ by

$$\sigma = \phi^{-1}(d_0).$$

Let $\{\delta_x\}_{x\in X}$ be a family of transition functions defined in the following way: For each $x \in X$,

$$\delta_x(\alpha) = \phi^{-1}(\Delta(\phi(\alpha), x)), \quad \text{for every } \alpha \in S.$$

Let the terminal weights function $\Theta : S \to \mathbb{R}$ be defined by

$$\Theta(\alpha) = \theta(\phi(\alpha)), \quad \text{for every } \alpha \in S.$$

Then $\mathcal{A} = (S, \sigma, \{\delta_x\}_{x\in X}, \Theta)$ is a weighted automaton over the vector space V having S as its set of states.

Clearly, for each $\alpha \in S$ and $u \in X^*$ we have

$$\delta_u(\alpha) = \phi^{-1}(\Delta(\phi(\alpha), u).$$

Furthermore, for every $u \in X^*$ the following holds

$$\begin{aligned}[\![\mathcal{A}]\!](u) &= \Theta(\delta^*(\sigma, u)) = \Theta(\delta_u(\sigma)) = \theta(\phi(\delta_u(\sigma))) = \theta(\phi(\phi^{-1}(\Delta(\phi(\sigma), u)))) = \\ &= \theta(\phi(\phi^{-1}(\Delta(\phi(\phi^{-1}(d_0)), u)))) = \theta(\Delta(d_0, u)) = [\![\mathcal{D}]\!](u)\end{aligned}$$

and therefore, $\mathcal{A}$ and $\mathcal{D}$ are language-equivalent. □

T. Li, G. Rabusseau and D. Precup in [21] defined a *nonlinear weighted finite automaton* over the field of real numbers $\mathbb{R}$ as a tuple $(\sigma, \{\delta_x\}_{x\in X}, \Theta)$, where $\sigma \in \mathbb{R}^n$ is a vector of initial weights, $\{\delta_x\}_{x\in X}$ is a family of transformations such that $\delta_x : \mathbb{R}^n \to \mathbb{R}^n$, for each $x \in X$, which are called transition functions, and $\theta : \mathbb{R}^n \to \mathbb{R}$ is a termination function. This definition is almost identical to our definition of a weighted automaton over a vector space. The difference is that the set S of vector states is not explicitly stated in the mentioned paper, but we can assume that S is the smallest set of vectors from $\mathbb{R}^n$ that contains σ and is closed for all transformations from the family $\{\delta_x\}_{x\in X}$, that is, $S = \{\sigma_u\}_{u\in X^*}$, where $\sigma_u = \delta^*(\sigma, u) = \delta_u(\sigma)$, for each $u \in X^*$. Another difference is that the transformations $\{\delta_x\}_{x\in X}$ are defined on $\mathbb{R}^n$, but nothing changes significantly if we replace them with their restrictions on S. The third difference is that Lee, Rabusseau and Precup considered automata over the finite-dimensional vector space $\mathbb{R}^n$, while in our definition we provide the possibility that the underlying vector space V be also infinite-dimensional. Therefore, the concepts of a nonlinear weighted finite automaton and a weighted automaton over a vector space are almost the same. Let us note that the adjective "finite" in the name of nonlinear weighted finite automata does not refer to the finiteness of the set S of vector states, as in our definition, but to the finite dimension of the space $\mathbb{R}^n$. In addition, regardless of the adjective nonlinear in the name, in the nonlinear weighted finite automaton among the transformations δ_x, $x \in X$, there can be both linear and nonlinear ones.

Here, a weighted automaton $\mathcal{A} = (S, \sigma, \{\delta_x\}_{x\in X}, \Theta)$ over a vector space V is defined to be *linear* if for any $x \in X$ the function δ_x is the restriction of some linear operator $\delta'_x : \operatorname{span}(S) \to \operatorname{span}(S)$ to the set S, and also, Θ is the restriction of some linear functional (linear form) $\Theta' : \operatorname{span}(S) \to \mathbb{R}$ to the set S, i.e.,

$$\delta'_x(s\alpha + t\beta) = s\delta'_x(\alpha) + t\delta'_x(\beta), \qquad \Theta'(s\alpha + t\beta) = s\Theta'(\alpha) + t\Theta'(\beta), \tag{13}$$

for all $x \in X$, $\alpha, \beta \in \operatorname{span}(S)$ and $s, t \in \mathbb{R}$. Otherwise, if some of the mappings δ_x, $x \in X$, and Θ can not be represented in this way, then $\mathcal{A}$ is said to be a *nonlinear weighted automaton over a vector space*. If $V \subseteq \mathbb{R}^n$, for some $n \in \mathbb{N}$, then $\mathcal{A}$ is linear if and only if for each $x \in X$ there is a matrix $M_x \in \mathbb{R}^{n\times n}$ such that $\delta_x(\alpha) = \alpha \cdot M_x$, for each $\alpha \in S$, and there is also a vector $\tau \in \mathbb{R}^n$ such that $\Theta(\alpha) = \alpha \cdot \tau$, for each $\alpha \in S$ (here $\alpha \cdot \tau$ denotes the scalar product of α and τ).

Let us note that our linear weighted automata over a vector space are almost identical to automata studied by Balle, Gourdeau and Panangaden in [3] (which are called there only weighted finite automata), the only difference is that there V was assumed to be a finite-dimensional space and $S = V$. However, even this small difference concerning the set of vector states can be significant. Namely, let $\mathcal{A} = (S, \sigma, \{\delta_x\}_{x\in X}, \Theta)$ be any linear weighted automaton over the vector space $V = \mathbb{R}^n$ with the set of vector states $S \subset V$ such that $\operatorname{span}(S) \neq V$, for each $x \in X$ let δ_x be the restriction of some linear operator $\delta'_x : \operatorname{span}(S) \to \operatorname{span}(S)$ to the set S, and let Θ be the restriction of some linear functional $\Theta' : \operatorname{span}(S) \to \mathbb{R}$ to the set S. Then we can easily extend any δ'_x to an operator $\delta''_x : V \to V$ which is not linear, for example, by taking δ''_x to concide with δ'_x on $\operatorname{span}(S)$ and

$$\delta''_x\big(\begin{bmatrix} s_1 & s_2 & \dots & s_n \end{bmatrix}\big) = \begin{bmatrix} s_1^2 & s_2^2 & \dots & s_n^2 \end{bmatrix},$$

for every $\begin{bmatrix} s_1 & s_2 & \dots & s_n \end{bmatrix} \in V \setminus \operatorname{span}(S)$, and we can extend Θ' to a non-linear function $\Theta'' : V \to \mathbb{R}$. Therefore, $\mathcal{A}'' = (V, \sigma, \{\delta''_x\}_{x\in X}, \Theta'')$ is a non-linear weighted automaton over the vector space V with the set of vector states V, but if we assume that the set of vector states is S, then $\mathcal{A}''$ becomes linear.

The following theorem explains the connection between finite-dimensional linear weighted automata over a vector space and weighted finite automata.

Theorem 4.2 *Every finite-dimensional linear weighted automaton over a vector space can be turned into a completely language-equivalent weighted finite automaton.*

Conversely, every weighted finite automaton can be turned into a completely language-equivalent finite-dimensional linear weighted automaton over a vector space.

Proof. Let $\mathcal{A} = (S, \sigma, \delta, \Theta)$ be a finite-dimensional linear weighted automaton over a vector space V of dimension n. Since $\mathcal{A}$ is linear, we have that for each $x \in X$ there exists a matrix $M_x \in \mathbb{R}^{n\times n}$ such that $\sigma \cdot M_x = \delta(\sigma, x)$, and moreover, there exists a vector $\tau \in \mathbb{R}^n$ such that $\Theta(\sigma) = \sigma \cdot \tau$. In this way, we obtain a weighted finite automaton $\mathcal{A}'$ which is given by the linear representation $\mathcal{A}' = (n, \sigma, \{M_x\}_{x\in X}, \tau)$.

Now, for every $u \in X^+$ such that $u = x_1 x_2 \cdots x_k$, where $x_1, x_2, \dots, x_k \in X$, we have

$$[\![\mathcal{A}']\!](u) = \sigma \cdot M_{x_1} \cdot M_{x_2} \cdot \cdots \cdot M_{x_k} \cdot \tau = \sigma \cdot M_u \cdot \tau = \delta^*(\sigma, u) \cdot \tau = \Theta(\delta^*(\sigma, u)) = [\![\mathcal{A}]\!](u),$$

and

$$[\![\mathcal{A}']\!](\varepsilon) = \sigma \cdot \tau = \delta^*(\sigma, \varepsilon) \cdot \tau = \Theta(\delta^*(\sigma, \varepsilon)) = [\![\mathcal{A}]\!](\varepsilon).$$

Finally, we have that

$$[\![\mathcal{A}']\!]_g(u) = \|\sigma_u\|_\infty = \|\sigma \cdot M_u\|_\infty = \|\delta^*(\sigma, u)\|_\infty = [\![\mathcal{A}]\!]_g(u),$$

for every $u \in X^*$.

Therefore, we have proved that the finite-dimensional linear weighted automaton $\mathcal{A}$ is completely language-equivalent to the weighted finite automaton $\mathcal{A}'$.

Conversely, let $\mathcal{A} = (A, \sigma, \delta, \tau)$ be a weighted finite automaton with n states. We define a weighted automaton $\mathcal{A}_N = (S_N, \sigma^N, \delta^N, \Theta^N)$ over the vector space $\mathbb{R}^n$ in the following way: we set

$$S_N = \{\sigma_u \,|\, u \in X^*\}, \quad \sigma^N = \sigma,$$

and we define functions $\delta^N : S_N \times X \to S_N$ and $\Theta^N : S_N \to \mathbb{R}$ by

$$\delta^N(\sigma_u, x) = \sigma_u \cdot \delta_x = \sigma_{ux}, \qquad \Theta^N(\sigma_u) = \sigma_u \cdot \tau,$$

for every $u \in X^*$. The automaton $\mathcal{A}_N$ is obviously well-defined and is called in [17] the *Nerode automaton* of the automaton $\mathcal{A}$. It remains to prove that the Nerode automaton $\mathcal{A}_N$ is completely language-equivalent to the original weighted finite automaton $\mathcal{A}$.

For an arbitrary word $u \in X^*$ we have that

$$[\![\mathcal{A}_N]\!](u) = \Theta^N((\delta^N)^*(\sigma, u)) = \Theta^N(\sigma_u) = \sigma_u \cdot \tau = [\![\mathcal{A}]\!](u).$$

and also,

$$[\![\mathcal{A}_N]\!]_g(u) = \|(\delta^N)^*(\sigma, u)\|_\infty = \|\sigma_u\|_\infty = [\![\mathcal{A}]\!]_g(u).$$

Therefore, the Nerode automaton $\mathcal{A}_N$ of $\mathcal{A}$ is completely language-equivalent to $\mathcal{A}$. □

Let $\mathcal{A} = (S, \sigma, \{\delta_x\}_{x\in X}, \Theta)$ be a finite weighted automaton over the vector space $V = \mathbb{R}^n$, and let us assume that $S = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ and $X = \{x_1, x_2, \ldots, x_s\}$. Let us form $m \times n$-matrices N and N_{x_i}, $i \in [1..s]$, and a column vector ($1 \times m$-matrix) ϑ such that rows in N are $\alpha_1, \alpha_2, \ldots, \alpha_m$, rows in N_{x_i} are $\delta_{x_i}(\alpha_1), \delta_{x_i}(\alpha_2), \ldots, \delta_{x_i}(\alpha_m)$, and entries in ϑ are $\Theta(\alpha_1), \Theta(\alpha_2), \ldots, \Theta(\alpha_m)$, in that order, i.e.

$$N = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}, \quad N_{x_i} = \begin{bmatrix} \delta_{x_i}(\alpha_1) \\ \delta_{x_i}(\alpha_2) \\ \vdots \\ \delta_{x_i}(\alpha_m) \end{bmatrix}, \ i \in [1..s], \quad \vartheta = \begin{bmatrix} \Theta(\alpha_1) \\ \Theta(\alpha_2) \\ \vdots \\ \Theta(\alpha_m) \end{bmatrix}$$

We will call N the *state matrix* and ϑ the *terminal vector*, while for each $i \in [1..s]$, we call N_{x_i} the *destination matrix* corresponding to the input letter x_i.

The following theorem provides a procedure for testing the linearity of a finite weighted automaton over a finite-dimensional vector space.

Theorem 4.3 *Let $\mathcal{A} = (S, \sigma, \{\delta_x\}_{x\in X}, \Theta)$ be a finite weighted automaton over the vector space $V \subseteq \mathbb{R}^n$. Then $\mathcal{A}$ is linear if and only if the matrix N has the same rank as the augmented matrix*

$$[N \mid N_{x_1} \mid N_{x_2} \mid \ldots \mid N_{x_s} \mid \vartheta].$$

Proof. First we prove that $\mathcal{A}$ is linear if and only if each of the following equations is solvable

$$N \cdot X_1 = N_{x_1}, \ N \cdot X_2 = N_{x_2}, \ \ldots, \ N \cdot X_s = N_{x_s}, \ N \cdot \chi = \vartheta, \tag{14}$$

where $X_1, X_2, \ldots, X_n$ are unknown $n \times n$-matrices, and χ is an unknown vector taking values in $\mathbb{R}^n$.

Assume that $\mathcal{A}$ is linear, i.e., that there are matrices $M_{x_i} \in \mathbb{R}^{n\times n}$, for each $i \in [1..s]$, and a vector $\tau \in \mathbb{R}^n$ such that

$$\delta_{x_i}(\alpha_j) = \alpha_j \cdot M_{x_i}, \ \text{for all } i \in [1..n], \ j \in [1..m], \qquad \Theta(\alpha_j) = \alpha_j \cdot \tau, \ \text{for each } j \in [1..m]. \tag{15}$$

According to the well-known row rule for matrix multiplication we obtain that

$$N \cdot M_{x_i} = \begin{bmatrix} \alpha_1 \cdot M_{x_i} \\ \alpha_2 \cdot M_{x_i} \\ \cdots \\ \alpha_m \cdot M_{x_i} \end{bmatrix} = \begin{bmatrix} \delta_{x_i}(\alpha_1) \\ \delta_{x_i}(\alpha_2) \\ \cdots \\ \delta_{x_i}(\alpha_m) \end{bmatrix} = N_{x_i}, \qquad N \cdot \tau = \begin{bmatrix} \alpha_1 \cdot \tau \\ \alpha_2 \cdot \tau \\ \cdots \\ \alpha_m \cdot \tau \end{bmatrix} = \begin{bmatrix} \Theta(\alpha_1) \\ \Theta(\alpha_2) \\ \cdots \\ \Theta(\alpha_m) \end{bmatrix} = \vartheta, \tag{16}$$

for every $i \in [1..s]$, and we conclude that $X_1 = M_{x_1}$, $X_2 = M_{x_2}$, $\ldots$, $X_s = M_{x_s}$ and $\chi = \tau$ are solutions of equations from (14).

Conversely, let any of the equations from (14) is solvable, and assume that $X_i = M_{x_i}$, for $i \in [1..s]$, and $\chi = \tau$, are solutions to these equations. From there we obtain that (16) holds, which further implies that (15) holds, and therefore, $\mathcal{A}$ is a linear weighted automaton over a vector space V.

Next we prove that each of the equations from (14) is solvable if and only if the rank of N is equal to the rank of the augmented matrix $N^* = [N \mid N_{x_1} \mid N_{x_2} \mid \ldots \mid N_{x_s} \mid \vartheta]$. Let R be the reduced row echelon form of the augmented matrix N^*, and let R be partitioned as follows:

$$R = [R_0 \,|\, R_1 \,|\, R_2 \,|\, \ldots \,|\, R_s \,|\, \nu]$$

where $R_0, R_1, R_2, \ldots, R_s$ are $m \times n$-matrices and ν is a column vector of dimension m. Then R_0 is the reduced row echelon form of N, for each $i \in [1..s]$, $[R_0 \,|\, R_i]$ is the reduced row echelon form of $[N \,|\, N_{x_i}]$, and $[R_0 \,|\, \nu]$ is the reduced row echelon form of $[N \,|\, \vartheta]$.

For an arbitrary $i \in [1..s]$ we have that the equation $N \cdot X_i = N_{x_i}$ is solvable if and only the equation $N \cdot c_k(X_i) = c_k(N_{x_i})$ is solvable for every $k \in [1..n]$. On the other hand, for every $k \in [1..n]$ we have that $N \cdot c_k(X_i) = c_k(N_{x_i})$ is solvable if and only if $\operatorname{rank}(N) = \operatorname{rank}([N \,|\, c_k(N_{x_i})])$, and this holds if and only if for every $j \in [1..m]$ for which the jth row of R_0 is the zero vector it follows that the jth coordinate of $c_k(N_{x_i})$ is equal to zero. Similarly, the equation $N \cdot \chi = \vartheta$ is solvable if and only if for every $j \in [1..m]$ for which the jth row of R_0 is the zero vector it follows that the jth coordinate of ϑ is equal to zero.

Therefore, we conclude that all equations in (14) are solvable if and only if for every $j \in [1..m]$ for which the jth row of R_0 is the zero vector, we have that all remaining entries in the jth row of the matrix $R = [R_0 \,|\, R_1 \,|\, R_2 \,|\, \ldots \,|\, R_s \,|\, \nu]$ are equal to zero, and this is equivalent to $\operatorname{rank}(N) = \operatorname{rank}(N^*)$.

Let us note that if R_0 does not have zero rows then all equations in (14) are solvable if and only if

$$\operatorname{rank}(N) = \operatorname{rank}(N^*) = \operatorname{rank}([N \,|\, N_{x_i}]) = \operatorname{rank}([N \,|\, \vartheta]) = \operatorname{rank}([N \,|\, c_k(N_{x_i})]) = m \ (\leqslant n),$$

for all $i \in [1..s]$ and $k \in [1..n]$. This completes the proof of the theorem. □

Example 4.4 Let $\mathcal{A} = (S, \sigma, \{\delta_x\}_{x \in X}, \Theta)$ be a finite weighted automaton with three vector states over the vector space $\mathbb{R}^2$ and an alphabet $X = \{x, y\}$, where the vector states $\sigma = \alpha_1$, α_2 and α_3, and the terminal vector ϑ are given with

$$\sigma = \alpha_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad \alpha_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad \alpha_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad \vartheta = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

while the transition graph is the one given in Figure 1.

Then the augmented matrix $N^* = [N \mid N_x \mid N_y \mid \vartheta]$ and its reduced row echelon form $RREF(N^*)$ are given by

$$N^* = \left[\begin{array}{cc|cc|cc|c} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right], \quad RREF(N^*) = \left[\begin{array}{cc|cc|cc|c} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & -1 & 1 \end{array}\right],$$

and it is clear that $\operatorname{rank}(N) = 2 \neq 3 = \operatorname{rank}(N^*)$. From there we get that $\mathcal{A}$ is nonlinear.

Let us also note that the equations (14) become

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \qquad \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

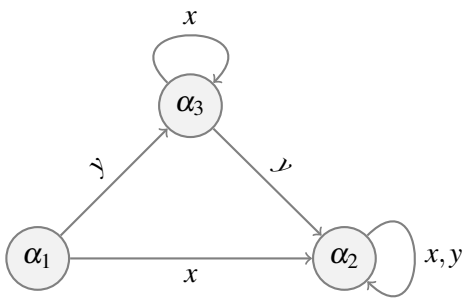


Figure 1: The transition graph of the automaton $\mathcal{A}$ from Example 4.4

It can be seen that none of these equations has a solution.

For a word function $f : X^* \to \mathbb{R}$ and a word $u \in X^*$, a word function $f_u : X^* \to \mathbb{R}$ defined by

$$f_u(v) = f(uv), \qquad \text{for every } v \in X^*, \tag{17}$$

is called the *derivative* of f with respect to u. Derivatives of conventional languages are also known as *right quotients*, *quotients* or *residuals* of languages.

We define a weighted automaton $\mathcal{A}_f = (S_f, \sigma^f, \delta^f, \Theta^f)$ over a vector space $\mathbb{R}^{X^*}$ as follows: the set S_f of vector states is given by $S_f = \{f_u \,|\, u \in X^*\}$, $\sigma^f = f$, and functions $\delta^f : S^f \times X \to S^f$ and $\Theta^f : S^f \to \mathbb{R}$ are given by

$$\delta^f(g,x) = g_x, \quad \Theta^f(g) = g(\varepsilon), \qquad \text{for all } g \in S_f \text{ and } x \in X. \tag{18}$$

It is clear that $\mathcal{A}_f$ is well-defined, and we will call it the *derivative automaton* of the word function f.

For a word function $f : X^* \to \mathbb{R}$, the *prefix closure* of f is a word function $\overline{f} : X^* \to \mathbb{R}$ defined by

$$\overline{f}(u) = \sup_{v \in X^*} |f(uv)| = \|f_u\|_\infty, \tag{19}$$

for every $u \in X^*$.

Theorem 4.5 *The derivative automaton $\mathcal{A}_f$ of a word function $f \in \mathbb{R}^{X^*}$ is a linear weighted automaton over the vector space $\mathbb{R}^{X^*}$ that computes f and generates the prefix closure $\overline{f}$ of f.*

In addition, $\mathcal{A}_f$ is a minimal weighted automaton over a vector space which computes f.

Proof. For the sake of simplicity, let us assume that $\mathbb{R}^{X^*} = V$.

First, we prove that $\mathcal{A}_f$ is linear. To that end, define functions $\delta_x : V \to V$, for every $x \in X$, and $\Theta : V \to \mathbb{R}$ as follows: $\delta_x(g) = g_x$ and $\Theta(g) = g(\varepsilon)$, for all $g \in V$ and $x \in X$. Further, consider arbitrary $g, h \in V$ and $s, t \in \mathbb{R}$. Then for arbitrary $x \in X$ and $u \in X^*$ we have that

$$(sg + th)_x(u) = (sg + th)(xu) = sg(xu) + th(xu) = sg_x(u) + th_x(u) = (sg_x + th_x)(u),$$

so we conclude that $(sg + th)_x = sg_x + th_x$, and now we have that

$$\delta_x(sg + th) = (sg + th)_x = sg_x + th_x = s\delta_x(g) + t\delta_x(h),$$
$$\Theta(sg + th) = (sg + th)(\varepsilon) = sg(\varepsilon) + th(\varepsilon) = s\Theta(g) + t\Theta(h).$$

Hence, δ_x, $x \in X$, and Θ are linear operators on V, and it is clear that δ_x^f (where $\delta_x^f(g) = \delta^f(g,x)$) is the restriction of δ_x to S_f, for each $x \in X$, and Θ^f is the restriction of Θ on S_f. This means that $\mathcal{A}_f$ is a linear weighted automaton over the vector space V. Next, for an arbitrary $u \in X^*$ we have that

$$[\![\mathcal{A}_f]\!](u) = \Theta((\delta^f)^*(\sigma^f,u)) = \Theta((\delta^f)^*(f,u)) = \Theta(f_u) = f_u(\varepsilon) = f(u\varepsilon) = f(u),$$
$$[\![\mathcal{A}_f]\!]_g(u) = \|(\delta^f)^*(\sigma^f,u)\|_\infty = \|(\delta^f)^*(f,u)\|_\infty = \|f_u\|_\infty = \overline{f}(u),$$

which means that the automaton $\mathcal{A}_f$ computes f and generates $\overline{f}$.

Let $\mathcal{A} = (S,\sigma,\delta,\Theta)$ be an arbitrary weighted automaton over a vector space that computes f, and let $S' = \{\sigma_u \,|\, u \in X^*\} \subseteq S$. Define a function $\phi : S' \to S_f$ by putting that $\phi(\sigma_u) = f_u$, for every $u \in X^*$. First we prove that ϕ is well-defined, i.e., that for any $u,v \in X^*$, from $\sigma_u = \sigma_v$ it follows $f_u = f_v$. Thus, consider $u,v \in X^*$ such that $\sigma_u = \sigma_v$, and an arbitrary $w \in X^*$. Then

$$\begin{aligned} f_u(w) &= f(uw) = [\![\mathcal{A}]\!](uw) = \Theta(\delta^*(\sigma,uw)) = \Theta(\delta^*(\delta^*(\sigma,u),w)) \\ &= \Theta(\delta^*(\sigma_u,w)) = \Theta(\delta^*(\sigma_v,w)) = \Theta(\delta^*(\delta^*(\sigma,v),w)) \\ &= \Theta(\delta^*(\sigma,vw)) = [\![\mathcal{A}]\!](vw) = f(vw) = f_v(w), \end{aligned}$$

so we conclude that $f_u = f_v$. Therefore, we get that ϕ is a well-defined function, and it is obvious that ϕ is surjective. This means that the cardinality of S_f is less than or equal to the cardinality of S', which is less than or equal to the cardinality of S. From there, we conclude that $\mathcal{A}_f$ is a minimal weighted automaton over a vector space which computes f. □

References

[1] Howard Anton & Robert C. Busby (2023): *Contemporary Linear Algebra*, 2nd edition. John Wiley & Sons.

[2] Borja Balle, Pascale Gourdeau & Prakash Panangaden (2017): *Bisimulation metrics for weighted automata*. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn & Anca Muscholl, editors: *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, 80, Leibniz International Proceedings in Informatics (LIPIcs), pp. 103:1–103:14, doi:10.4230/LIPIcs.ICALP.2017.103.

[3] Borja Balle, Pascale Gourdeau & Prakash Panangaden (2022): *Bisimulation metrics and norms for real-weighted automata*. *Information and Computation* 282, p. 104649, doi:10.1016/j.ic.2020.104649.

[4] Filippo Bonchi, Marcello Bonsangue, Michele Boreale, Jan Rutten & Alexandra Silva (2012): *A coalgebraic perspective on linear weighted automata*. *Information and Computation* 211, pp. 77–105, doi:10.1016/j.ic.2011.12.002.

[5] Michele Boreale (2009): *Weighted bisimulation in linear algebraic form*. In M. Bravetti & G. Zavattaro, editors: *CONCUR 2009 – Concurrency Theory, 20th Intern. Conference*, *Lecture Notes in Computer Science* 5710, Springer, Bologna, Italy, pp. 163–177, doi:10.1007/978-3-642-04081-8_12.

[6] Christos G. Cassandras & Stéphane Lafortune (2008): *Introduction to Discrete Event Systems*. Springer, doi:10.1007/978-0-387-68612-7.

[7] Miroslav Ćirić, Manfred Droste, Jelena Ignjatović & Heiko Vogler (2010): *Determinization of weighted finite automata over strong bimonoids*. *Information Sciences* 180, pp. 3497–3520, doi:10.1016/j.ins.2010.05.020.

[8] Miroslav Ćirić, Jelena Ignjatović, Milan Bašić & Ivana Jančić (2014): *Nondeterministic automata: equivalence, bisimulations, and uniform relations*. *Information Sciences*, pp. 185–218, doi:10.1016/j.ins.2013.07.029.

[9] Miroslav Ćirić, Jelena Ignjatović, Nada Damljanović & Milan Bašić (2012): *Bisimulations for fuzzy automata*. *Fuzzy Sets and Systems* 186, pp. 100–139, doi:10.1016/j.fss.2011.07.003.

[10] Miroslav Ćirić, Jelena Ignjatović, Ivana Jančić & Nada Damljanović (2012): *Computation of the greatest simulations and bisimulations between fuzzy automata*. Fuzzy Sets and Systems 208, pp. 22–42, doi:10.1016/j.fss.2012.05.006.

[11] Miroslav Ćirić, Aleksandar Stamenković, Jelena Ignjatović & Tatjana Petković (2010): *Fuzzy relation equations and reduction of fuzzy automata*. Journal of Computer and System Sciences 76, pp. 609–633, doi:10.1016/j.jcss.2009.10.015.

[12] Thomas Colcombet & Daniela Petrisan (2017): *Automata and minimization*. ACM SIGLOG News 4(2), pp. 4–27, doi:10.1145/3090064.3090066.

[13] Nada Damljanović, Miroslav Ćirić & Jelena Ignjatović (2014): *Bisimulations for weighted automata over an additively idempotent semiring*. Theoretical Computer Science 534, pp. 86–100, doi:10.1016/j.tcs.2014.02.032.

[14] Mandfed Droste, Zoltán Fülöp, Dávid Kószó & Heiko Vogler (2020): *Crisp-determinization of weighted tree automata over additively locally finite and past-finite monotonic strong bimonoids is decidable*. In Galina Jirásková & Giovanni Pighizzini, editors: DCFS 2020, Lecture Notes in Computer Science 12442, pp. 39–51, doi:10.1007/978-3-030-62536-8_4.

[15] Zoltán Fülöp, Dávid Kószó & Heiko Vogler (2021): *Crisp-determinization of weighted tree automata over strong bimonoids*. Discrete Mathematics and Theoretical Computer Science 23(1), doi:10.46298/dmtcs.5943.

[16] Jelena Ignjatović, Miroslav Ćirić & Stojan Bogdanović (2008): *Determinization of fuzzy automata with membership values in complete residuated lattices*. Information Sciences 178, pp. 164–180, doi:10.1016/j.ins.2007.08.003.

[17] Jelena Ignjatović, Miroslav Ćirić, Stojan Bogdanović & Tatjana Petković (2010): *Myhill-Nerode type theory for fuzzy languages and automata*. Fuzzy Sets and Systems 161, pp. 1288–1324, doi:10.1016/j.fss.2009.06.007.

[18] Zorana Jančić & Miroslav Ćirić (2014): *Brzozowski type determinization for fuzzy automata*. Fuzzy Sets and Systems 249, pp. 73–82, doi:10.1016/j.fss.2014.02.021.

[19] Zorana Jančić, Jelena Ignjatović & Miroslav Ćirić (2011): *An improved algorithm for determinization of weighted and fuzzy automata*. Information Sciences 181, pp. 1358–1368, doi:10.1016/j.ins.2010.12.008.

[20] Zorana Jančić, Ivana Micić, Jelena Ignjatović & Miroslav Ćirić (2016): *Further improvements of determinization methods for fuzzy finite automata*. Fuzzy Sets and Systems 301, pp. 79–102, doi:10.1016/j.fss.2015.11.019.

[21] Tianyu Li, Guillaume Rabusseau & Doina Precup (2018): *Nonlinear weighted finite automata*. In: Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research 84, pp. 679–688.

[22] Ivana Micić, Zorana Jančić, Jelena Ignjatović & Miroslav Ćirić (2015): *Determinization of fuzzy automata by means of the degrees of language inclusion*. IEEE Transactions on Fuzzy Systems 23(6), pp. 2144–2153, doi:10.1109/TFUZZ.2015.2404348.

[23] Marcel-Paul Schützenberger (1961): *On the definition of a family of automata*. Information and Control, pp. 245–270, doi:10.1016/S0019-9958(61)80020-X.

[24] Aleksandar Stamenković, Miroslav Ćirić & Milan Bašić (2018): *Ranks of fuzzy matrices. Applications in state reduction of fuzzy automata*. Fuzzy Sets and Systems 333, pp. 124–139, doi:10.1016/j.fss.2017.05.028.

[25] Aleksandar Stamenković, Miroslav Ćirić & Jelena Ignjatović (2014): *Reduction of fuzzy automata by means of fuzzy quasi-orders*. Information Sciences 275, pp. 168–198, doi:10.1016/j.ins.2014.02.028.

[26] Aleksandar Stamenković, Miroslav Ćirić & Jelena Ignjatović (2015): *Different models of automata with fuzzy states*. Facta Universitatis, Series Mathematics and Informatics 30(3), pp. 235–253.

Freezing 1-Tag Systems with States

Szilárd Zsolt Fazekas*
Akita University
Akita, Japan
szilard.fazekas@ie.akita-u.ac.jp

Shinnosuke Seki†
University of Electro-Communications
Tokyo, Japan
s.seki@uec.ac.jp

We study 1-tag systems with states obeying the freezing property that only allows constant bounded number of rewrites of symbols. We look at examples of languages accepted by such systems, the accepting power of the model, as well as certain closure properties and decision problems. Finally we discuss a restriction of the system where the working alphabet must match the input alphabet.

1 Introduction

Tag systems are a class of deterministic string rewriting systems. In each step they read out the first letter, say a, of the current word along with the succeeding $n-1$ letters (where n is a system parameter), refer to the system's transition table (function) δ with the letter read as a key, and append the word $\delta(a)$ at the end of the current word. The system stops if the key is a special halting symbol or if there are less than n letters left in the word. A system in this class is often called an n-tag system including the value of n explicitly. It is well known that n-tag systems are capable of simulating Turing machines for any $n \geq 2$ (see, e.g., [2, 3]). The initial definition is stateless, has no additional storage or intricate rules describing its dynamics; hence such systems are ideal for being simulated in other computational models such as 1D cellular automata or molecular computation models, particularly in their early stage of development wherein knowledge and techniques for programming are yet to be developed.

Recently, the oritatami model of RNA co-transcriptional folding has been introduced [5]. In this model, an abstract RNA sequence folds upon itself greedily while being synthesized in order to achieve various computational tasks *in-silico*. These tasks are usually relatively simple from the viewpoint of computational complexity theory, such as, the detection of some specific molecule for gene expression regulation [15].

The cyclic variant of tag systems (cts) introduced by Cook [3] has been simulated in the oritatami model in order to prove its Turing completeness [5]; periodicity supposed for the sequence to be folded in the oritatami computation also favored this variant. With more tools available for oritatami programming including finite state control [12] and the molecular implementation of the oritatami model within view, a class of tag systems or their functional enhancements with extra features that are not as computationally powerful as the Turing machine gets more significant.

Including states enables even 1-tag systems to simulate Turing machines and they characterize the class of context-sensitive languages if all appended words are restricted to be of length at most 1, that is, the word never gets strictly longer than the initial one. This was shown a long time ago [17], with such 1-tag systems with states being referred to as circular automata. A primary source of computability even under this restriction is that each "cell" of the input tape can be rewritten endlessly in an arbitrary

*Szilárd Zsolt Fazekas was supported by JSPS KAKENHI Grant Number JP23K10976.
†Shinnosuke Seki was supported by JSPS KAKENHI Grant-in-Aids for Scientific Research (B) 20H04141 and (C) 20K11672.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 82–95, doi:10.4204/EPTCS.386.8

manner. As it is experimentally not trivial to implement arbitrarily-rewritable media, it makes sense to suppose the *freezing property*, under which a letter can be replaced with only a smaller letter according to some pre-determined order.

Many types of machines processing their input using some first-in-first-out storage have been investigated starting with queue automata and various restrictions thereof [9]. Such models use queues *in addition to* their input tape and are generally quite powerful, whereas freezing 1-tag systems (Fr1TASS) are towards the lower end of computational complexity, as we will see. A model that is rather close in spirit to our subject is the iterated uniform finite transducer [8]. Such transducers can simulate freezing 1-tag systems in a straightforward manner, but are much more powerful in the general case due to the lack of the freezing property. However, limiting the number of so-called 'sweeps' performed by these transducers by a function that is linear in the length of the input might produce systems that are similar in accepting power to our tag systems, although the bound on sweeps does not directly translate into the constant bound on the number of rewritings per position. Another recent computing device with a similarity to freezing tag systems is the one-way jumping automaton, which reads and erases letters on a circular tape [1], a behavior that can be simulated by freezing tag systems, but it is easy to see that those automata are strictly weaker than our current model.

In this paper, we explore basic properties of freezing 1-tag systems. In Section 2 we define the model and two different accepting modes borrowed from the theory of pushdown automata. We show that the accepting modes are equivalent in the general case. In Section 3 we start the study of the accepting power of the model. We can prove that it is between the regular and the context-sensitive languages and that it is not included in the context-free class, but at present we cannot show that the inclusion does not hold in the other direction either. Next, in Section 4 we show that the class of languages accepted is closed under Boolean operations and that with a simple idea one can trade off description size for time complexity when constructing systems for intersection or union. In Section 5 we study some fundamental decision problems such as emptiness, universality, equivalence, and show that they are not decidable by reduction to the Post Correspondence Problem. In Section 6 we discuss a restriction of Fr1TASS where the tape alphabet is the same as the input alphabet. In their restricted form these systems are much weaker and we can use a 'computation flattening' argument to prove negative results about systems with accepting state mode. We conclude the paper with some remarks and suggestions for future research in Section 7.

2 Preliminaries

Let Σ be a finite alphabet and Σ^* be a set of all words over Σ including the empty word λ. The length of a word $w \in \Sigma^*$ is the number of letters that occur in w, and is denoted by $|w|$.

A *1-tag system with states* (1TASS) is a string rewriting system denoted by a tuple $(\Sigma, \Gamma, Q, q_0, F, \delta)$, where Σ is a finite input alphabet, Γ is a finite tape alphabet that includes Σ as its subset, Q is a finite set of states including the initial state q_0, $F \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Gamma \to Q \times \Gamma^*$ is a transition function. The transition function is *freezing* if, for all $q \in Q$, $a \in \Gamma$, and $(p,u) \in \delta(q,a)$, we have $|u| \leq 1$ (length-non-increasing), and furthermore, either $u = \lambda$ or $u \leq a$ according to some predefined order $\leq$ over the elements of Γ. In this case, the 1TASS itself is also said to be freezing. We will refer to these systems as Fr1TASS.

A configuration of a freezing 1TASS $M = (\Sigma, \Gamma, Q, q_0, F, \delta, \tau)$ is a pair (q,w) of the current state $q \in Q$ and a current word $w \in \Gamma^*$. Suppose this system is in the configuration $(p, a_1 a_2 \cdots a_n)$ for some $p \in Q$, $n \geq 0$, and $a_1, a_2, \ldots, a_n \in \Gamma$. Then it can transfer to a configuration (q,v) if $(q,b) \in \delta(p,a_1)$ and $v = a_2 \cdots a_n b$; in this case, we write $(p, a_1 \cdots a_n) \to_M (q,v)$. The reflexive and transitive closure of the

relation $\to_M$ is denoted by $\to_M^*$. In the 1TASS, a word can be considered as being written on a cyclic tape along which a finite-state control moves in the clockwise direction while rewriting.

Empty tape vs accepting state. The model above can be introduced with or without deletion, which as we will see shortly, does not make a difference with respect to the accepting power. In the first case we allow transitions in which a letter is replaced by the empty word, effectively erasing the cell from the tape. In this scenario we can define acceptance conditions similar to pushdown automata: the machine accepts when the tape is empty (ET) or when the machine enters an accepting state (AS). Formally, starting from an initial configuration (q_0, w), M can accept an input word $w \in \Sigma^*$ in two different modes: *by an accepting state* if $(q_0, w) \to_M^* (q_f, v)$ for some accepting state $q_f \in F$ and a word $v \in \Gamma^*$, while *by the empty tape* if $(q_0, w) \to_M^* (q, \lambda)$ for some $q \in Q$. Where the distinction is necessary, the languages accepted by the system M by accepting state and by empty tape will be denoted by $L(M)_{AS}$ and $L(M)_{ET}$, respectively. Note that the AS mode as defined here introduces a technical problem: a system in this mode either does not accept the empty word or it accepts every input. This is due to fact that accepting the empty word requires that the initial state is final, as well. However, then such a system accepts any input right away as it is already in a final state. Therefore, to simplify the presentation we allow AS mode systems to have a special transition from the initial state to a final state on reading λ and require that such transition is only used when the input is empty. This allows stating our results in a more general form without emphasizing this caveat whenever talking about AS mode Fr1TASS.

First we show that the two conditions lead to the same computational power, which simplifies our exposition further on as we will not have to specify the accepting mode. The following technical lemma states that AS mode machines do not actually need to erase any symbol from their tape.

Lemma 1. *For any Fr1TASS $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ there exists a Fr1TASS $B = (\Sigma, \Gamma_2, Q_1, q_1, F_1, \delta_2)$ such that $L(A)_{AS} = L(B)_{AS}$ and the transition function of B does not erase symbols, that is, $\delta_2(q,a) = (q', \lambda)$ is not allowed for any $q, q' \in Q$ and $a \in \Gamma_2$.*

Proof. The proof is straightforward. If A does not erase symbols, then $B = A$ concludes the argument. If it does, then B simulates all non-erasing transition of A and for each erasing transition of A of the form $\delta_1(q,a) = (q', \lambda)$ we set $\delta_2(q,a) = (q', \square)$, where $\square \in \Gamma_2 \setminus \Gamma_1$ is a new symbol standing in for the positions erased by A. We set $\square$ to be the smallest letter in Γ_2 and we add a loop $\delta_2(q, \square) = (q, \square)$ to each state q, ensuring that B performs the same computations as A. □

We now show that accepting modes ET and AS are equivalent.

Lemma 2 (ET simulates AS)**.** *For any Fr1TASS $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ there exists a Fr1TASS $B = (\Sigma, \Gamma_2, Q_1, q_1, F_1, \delta_2)$ such that $L(A)_{AS} = L(B)_{ET}$.*

Proof. In this case the ET machine will be almost identical to the AS one. To get the ET machine accepting the same language, we simply add erasing loops $\delta(f,a) = (f, \lambda)$ to all accepting states f for all tape symbols a. By the definition of AS, when the machine reaches an accepting state, it halts, so we may assume w.l.o.g. that in A there are no outgoing transitions from any accepting state, so adding the aforementioned transitions does not introduce non-determinism. If the AS machine reaches a final state by an input, the same input will take the ET machine into the same state, whereby it will erase all the remaining symbols from the tape. Conversely, by Lemma 1 we may assume that A never erases its tape, therefore the only words leading to an empty tape in B will be the ones accepted by A. □

Lemma 3 (AS simulates ET)**.** *For any Fr1TASS $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ there exists a Fr1TASS $B = (\Sigma, \Gamma_2, Q_1, q_1, F_1, \delta_2)$ such that $L(A)_{ET} = L(B)_{AS}$.*

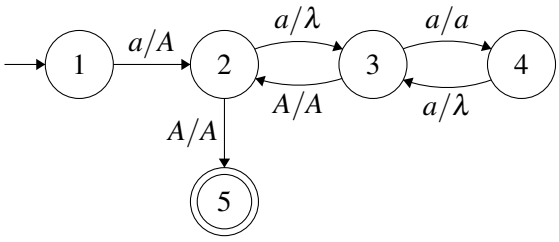


Figure 1: Fr1TASS accepting the language $\{a^{2^n} \mid n \geq 0\}$.

Proof. Similar to the argument in Lemma 1, we can replace erasing transitions $\delta_1(q,a) = (q',\lambda)$ with $\delta_2(q,a) = (q',\square)$ for some newly introduced smallest letter $\square$, and add $\square$-labeled loops to all states. We create two copies of the tape alphabet of A, marked and unmarked. Because of this, when the computation begins, B can mark the first letter to keep track of the start of the input. All operations on that symbol will be done with marked symbols. We duplicate the whole state diagram of A, such that the two copies of the states will 'remember' whether some symbol other than $\square$ was read since last passing the marked start. If the marked start is read twice with no non-$\square$ symbol in between, it means that on the given input A erased the tape, so B will transition to an accepting state. $\square$

2.1 Examples

Example 1. Even over a unary alphabet the ability to repeatedly read the tape allows freezing 1TASS to accept complex languages, such as the well-known non-context-free language $\{a^{2^n} \mid n \geq 0\}$. The tag system in Fig. 1 is intuitive and is essentially the same as for iterated uniform finite state transducers ([8], Lemma 20). The system erases every second occurrence of a which means that in each sweep it halves the length of the remaining tape content. Together with marking the first position with a special symbol at the start this allows Fr1TASS to process correct inputs in logarithmically many sweeps in the length of the input.

Example 2. Our next example is the marked copy language $\{\#w\#w \mid w \in \{a,b\}^*\}$. It is well known that the language is not context-free by a simple application of the Bar-Hillel pumping lemma. A simple Fr1TASS as in Fig. 2 can accept this language by matching and erasing pairs of letters at the same distance from the two special markers # iteratively, accepting the language in linearly many sweeps.

Example 3: accepting nondeterministic context-free languages. As will be detailed later, we were not able to prove that Fr1TASS cannot accept the language of palindromes, even though we conjecture that is the case. Our next idea was that perhaps such systems cannot even detect positions of the input at a certain ratio of the length from the beginning. Somewhat surprisingly, though, this proved to be false. Let us explain how a freezing 1TASS can detect the center of a given input. The idea can be adapted to detect positions at other linearly defined distances from the start. The problem is formalized as follows: modify a given a freezing 1TASS so that, given an input $w = a_0a_1\cdots a_{n-1}$ of length $n \geq 0$, it can mark $a_{\lfloor n/2 \rfloor}$ as a preprocess.

Solving this problem is equivalent to marking the letters in the first half somehow. Let $k = \lfloor n/2 \rfloor$. The following algorithm first marks k letters of w (Step 2), and then move each mark "rightward" across the first letter a_0, which is distinguished from the other letters (Step 1).

1. Mark the first letter as $s_0a_1\cdots a_{n-1}$.
2. Mark every other letter as $s_0\overline{a_1}a_2\overline{a_3}\cdots$. This results in $s_0\overline{a_1}a_2\overline{a_3}\cdots a_{n-2}\overline{a_{n-1}}$ if n is even, or $s_0\overline{a_1}a_2\overline{a_3}\cdots\overline{a_{n-2}}a_{n-1}$ if n is odd.

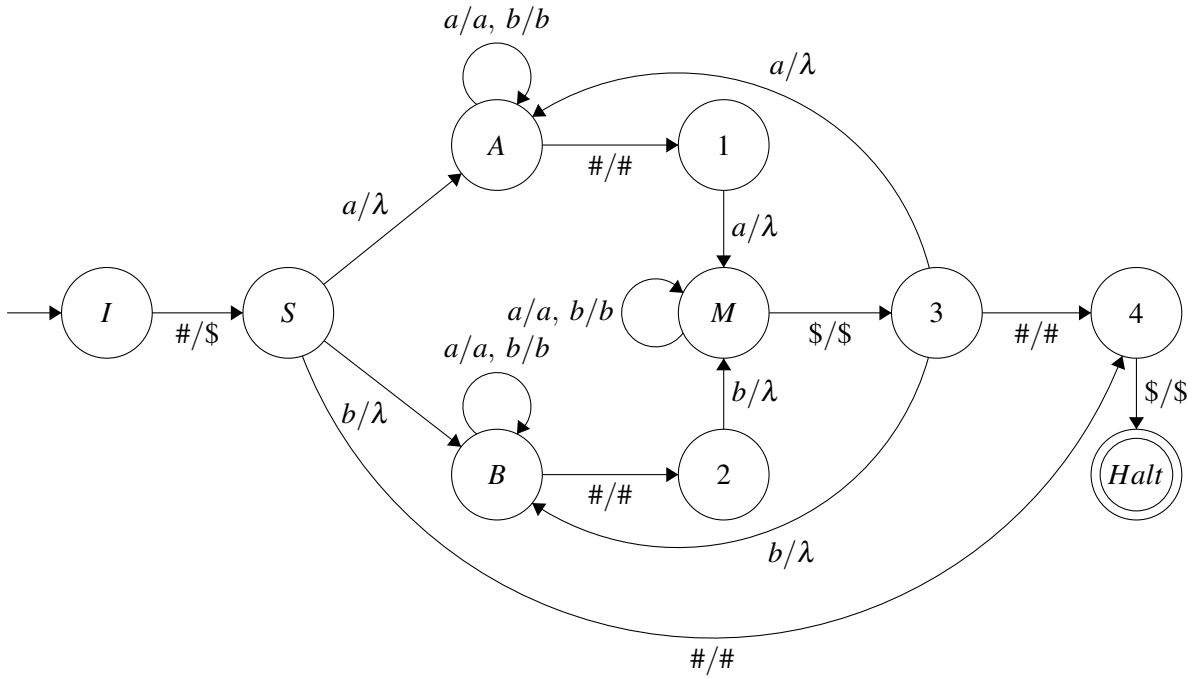


Figure 2: Fr1TASS accepting the language $\{\#w\#w \mid w \in \{a,b\}^*\}$.

The following steps will be repeated until step 3 is unsuccessful, i.e., no overlined position is preceded by an unoverlined one. The read-write head starts at s_0, after finishing steps 1 and 2.

3. Find the first overlined letter, say $\overline{a_j}$, after at least one unoverlined one.
4. Remove the overline as $\overline{a_j} \to a'_j$ (prime is must due to the freezing property).
5. Scan the tape until s_0, which might be overlined.
6. Overline the first unoverlined letter, which might have been primed.

One overline per iteration is shifted to the first half of the word. After repeating Steps 3-6 k times, Step 3 fails to find an unoverlined letter prior to an overlined one; thus the system escapes from this loop. At this point, the k overlines have been all moved to the left half of the word as $\overline{s_0}\overline{a'_1}\overline{a_2}\overline{a'_3}\cdots\overline{a'_{k-1}}a_k\cdots a_{n-1}$. The first unoverlined letter is $a_{\lfloor n/2 \rfloor}$.

3 Power of Fr1TASS

Let $\Sigma = \{1,\ldots,k\}$ be the ordered alphabet of a 1TASS A and consider an input $w = a_1\cdots a_n$, where $a_i \in \Sigma$. We can perceive the computation performed by A as happening in 'sweeps' on a circular tape. A sweep starts at the first position and ends when we reach that position again. Formally, the length of a *sweep* of computation can be defined inductively. Let $r_1 = n$ and then r_i (for $i > 1$) is defined as the length of the remaining input w_i after $i-1$ sweeps, i.e., if $m = \sum_{\ell=1}^{i-1} r_\ell$, then for some $q \in Q$ we have $(q_0, w) \to^m (q, w_i)$ and we set $r_i = |w_i|$. Assuming that the computation stops, the freezing property of A imposes that one of the following is true for each $i > 1$:

1. $r_i < r_{i-1}$.

2. $w_{i-1} = b_1 \cdots b_{r_i}$ and $w_i = c_1 \cdots c_{r_i}$ with $c_j \leq b_j$ for each $j \in \{1,\ldots,r_i\}$, and there exists some $j \in \{1,\ldots,r_i\}$ such that $c_j < b_j$.
3. $w_{i-1} = w_i$, but the sweeps $i-1$ and i start in different states.

From here, we can put an upper bound on the number of sweeps in a computation, and therefore an upper bound on the number of steps. Case 1 can happen at most n times. From Case 3 we get that there can be at most $|Q|$ consecutive sweeps that do not change the tape. The number of times Case 2 occurs is bounded by the total number of possible rewritings. If the tape content is $w = b_1 \cdots b_n$ with $b_i \in \{1,\ldots,k\}$ for all $1 \leq i \leq n$, then each letter can be rewritten to a smaller letter at most $k-1$ times, which means that Case 2 cannot occur more than $\sum_{\ell=1}^{n} b_\ell$ times, and $\sum_{\ell=1}^{n}(b_\ell - 1) \leq n(k-1)$. This means that the number of sweeps performed on an input of length n is $O(n)$. In each sweep we make at most n steps to the right and a Turing machine simulating Fr1TASS would need to make at most n steps to the left at the end of each sweep to return to the beginning. Thus we can conclude that the class of languages accepted by freezing 1TASS is included in DTIME(n^2).

The class of languages accepted by Fr1TASS strictly includes the class of regular languages. We can see that the inclusion is strict (even for unary languages) from the examples of the previous section. In order to show that the inclusion holds, we simulate a deterministic finite automaton by an AS mode Fr1TASS. The construction is simple but we need to bear in mind the fact that if the Fr1TASS reaches a final state reading a word w consisting only of symbols of the input alphabet, that results in the system accepting any word from the language $w\Sigma^*$ according to the definition.

Lemma 4. *For any regular language R there exists a Fr1TASS A such that $L(A)_{AS} = R$.*

Proof. Let $M = (\Sigma, Q, q_0, F, \delta)$ be a deterministic finite automaton accepting R. We construct the Fr1TASS $A = (\Sigma, \Sigma \cup \{\Box\}, Q \cup \{f_A\}, q_0, \{f_A\}, \delta')$, where $\Box \notin \Sigma$, $f_A \notin Q$, as follows. For each transition $\delta(q,a) = q'$ of the DFA, the Fr1TASS has a transition $\delta'(q,a) = (q', \Box)$. For each $f \in F$ we add the transitions $\delta'(f, \Box) = (f_a, \lambda)$. The system A walks the same path in the transition diagram as M does, but it replaces each letter by the $\Box$ symbol to mark it read. If reading the input takes the original system to a final state then the simulating Fr1TASS will have a $\Box$-labeled transition to a final state of its own. $\Box$

Due to the AS and ET modes being equivalent, we can also construct an ET mode Fr1TASS for any regular language. At the other extreme, it is easy to see that linear bounded automata can simulate Fr1TASS, which means that the class of languages accepted by these systems is included in the class of context-sensitive languages. With respect to the other classes of the Chomsky hierarchy, we conjecture that the power of Fr1TASS is incomparable, but we lack the tools to show both sides of such statements. The examples from the previous section demonstrate that not every Fr1TASS language is context-free. We found the ability of Fr1TASS to mark the middle letter of a word counterintuitive, due to the fact that although (even one turn) pushdown automata can accept the related language of words with a in the middle, it needs non-determinism to do so, as we will prove below. As Fr1TASS are deterministic, they cannot guess the middle and match the length of prefixes and suffixes. Nevertheless, nondeterminism is not required for Fr1TASS to mark the middle as shown in Example 3.

Consider $L_a = \{uav \mid |u| = |v|\}$. As we will show now, this language is not a deterministic context-free language. We will use the so-called DCFL pumping lemma below, due to Yu [16].

Lemma 5. *Let L be a deterministic context-free language. Then there exists a constant n for L such that for any pair of words $w, w' \in L$ if*

1. *$w = xy$ and $w' = xz$, $|x| > n$, and*
2. *first letter of y = first letter of z,*

then either 3. or 4. holds:

3. *there is a factorization $x = x_1x_2x_3x_4x_5$, with $|x_2x_4| \geq 1$ and $|x_2x_3x_4| \leq n$, such that for all $i \geq 0$ we have that $x_1{x_2}^i x_3{x_4}^i x_5 y$ and $x_1{x_2}^i x_3{x_4}^i x_5 z$ are in L;*
4. *there exist factorizations $x = x_1x_2x_3, y = y_1y_2y_3$ and $z = z_1z_2z_3$, with $|x_2| \geq 1$ and $|x_2x_3| \leq n$, such that for all $i \geq 0$ we have that $x_1{x_2}^i x_3 y_1{y_2}^i y_3$ and $x_1{x_2}^i x_3 z_1{z_2}^i z_3$ are in L.*

Theorem 6. *$L_a = \{uav \mid |u| = |v|\}$ is not a deterministic context-free language.*

Proof. Suppose that L_a were a DCFL and hence, that Lemma 5 applied. Let n be the constant from the lemma and consider the words $x = b^{n+2}ab^{n+1}$, $y = a$ and $z = ab^{2n+4}$. It is easy to see that both xy and xz are in L_a and long enough to meet the two conditions of the lemma, and the first letter of both y and z is a. Now we will show that assuming either conclusion of the lemma leads to a contradiction. First, suppose conclusion 3. is true. Depending on the factorization of x, we have the following cases:

1. $x_1, x_2, x_4, x_5 \in b^*$, $x_3 \in b^*ab^*$: for $x_1x_2^0x_3x_4^0x_5y$ to be in L_a, we need $|x_2| = |x_4|$ and from the lemma we know they are not empty, so let $x_2 = x_4 = b^k$ for some positive $k \leq \frac{n}{2}$. However, then $x_1x_2^0x_3x_4^0x_5z = x_1x_2^0x_3x_4^0x_5ab^{2n+4}$, where the length of $x_1x_2^0x_3x_4^0x_5 = 2n+4-2k$, so the letter at the middle of the word is b, contradicting $x_1x_2^0x_3x_4^0x_5z \in L_a$.
2. x_1 or x_5 contains a. In both cases $x_1x_2^0x_3x_4^0x_5y$ will result in a word with fewer b's on one side of the first a than the other, a contradiction.
3. x_2 or x_4 has an a. In both cases $x_1x_2^0x_3x_4^0x_5y$ will have only one a, at the end of the word, a contradiction.

Now suppose conclusion 4. is true. Since the factorization $x = x_1x_2x_3$ has the property $|x_2x_3| < n$ and $|x_2| \geq 1$, we know that $x_2 = b^k$, for some positive $k \leq n$. However, this means that for any factorization $y = y_1y_2y_3$, the word $x_1x_2^0x_3y_1y_2^0y_3$ is of the form $b^{n+2}ab^{n+1-k}$ or $b^{n+2}ab^{n+1-k}a$. As neither of those is in L_a, because $k \geq 1$, we arrived at a contradiction again. Consequently, L_a is not a deterministic context-free language. □

4 Closure properties

We can show that the class of languages accepted by Fr1TASS forms a Boolean algebra, i.e., it is closed under union, intersection and complement. For the first two, we can adapt the classical construction used in the case of finite automata: the machine simulating the union/intersection of two others will have pairs of states representing the states of the starting machines and its alphabet will also consist of pairs of letters, to keep track of the tape of both simulated machines.

Theorem 7. *The class of languages accepted by Fr1TASS is closed under union and intersection.*

Proof. Consider two languages, accepted by $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ and $B = (\Sigma, \Gamma_2, Q_2, q_2, F_2, \delta_2)$, respectively. We construct the system

$$C = (\Sigma, \Sigma \cup (\Gamma_1 \times \Gamma_2), Q_1 \times Q_2, (q_1, q_2), F_1 \times F_2, \delta)$$

accepting the language $L(A) \cap L(B)$ as follows. The computation of C will simulate the computations of A and B in parallel, similar to the classical finite automaton construction. By Lemma 1 we may assume that the systems A and B do not erase any symbols, so the length of the word on the tape is the same throughout the computation, making the parallel simulation possible. The difference is that here we have

to observe the freezing property, so the ordering of the tape alphabet Γ and the transition function δ need to be carefully defined. Let the total orderings of Γ_1 and Γ_2 be $\leq_1$ and $\leq_2$, respectively. Those two total orderings naturally define the partial order $\leq_{12}$ on $\Gamma_1 \times \Gamma_2$ as $(a,b) \leq_{12} (c,d)$ if $a \leq_1 c$ and $b \leq_2 d$. By Szpilrajn's theorem [14], every partial order has a linear extension, and we can efficiently construct such a linear order compatible with $\leq_{12}$ by any well-known topological sorting algorithm (e.g. Kahn's [7]), since $\leq_1$ and $\leq_2$ are finite. We define the transition function of C as $\delta((q_1,q_2),a) = ((q_1',q_2'),(b_1,b_2))$ where

$$((q_1',b_1),(q_2',b_2)) = \begin{cases} (\delta_1(q_1,a),\delta_2(q_2,a)) & \text{if } a \in \Sigma \\ (\delta_1(q_1,a_1),\delta_2(q_2,a_2)) & \text{if } a = (a_1,a_2) \notin \Sigma \end{cases}.$$

By the definition of δ we can be certain that the freezing property is preserved, that is, symbols of Γ_2 are rewritten respecting the linear extension of $\leq_{12}$. The proof for the union follows the same line with some small changes. Since the computations are done in parallel, it may happen that one of the machines gets stuck, i.e., it has no outgoing transition from its current state for the current input letter. However, if the other machine accepts, the input should be accepted. To handle such situations we introduce pairs of states $(q_1,\perp)$ and $(\perp,q_2)$ for all $q_1 \in Q_1, q_2 \in Q_2$, where having $\perp$ as one of the state components means the respective machine could not continue its computation. We can reach such states by transitions $\delta((q_1,q_2),a) = ((q_1',\perp),b)$ when $\delta_1(q_1,a) = (q_1',b)$ and $\delta_2(q_2,a)$ is undefined, and then add transitions of the form $\delta((q_1',\perp),a) = ((q_1'',\perp),(b,b))$ if $\delta_1(q_1',a) = (q_1'',(b,b))$ and their counterpart for the $(\perp,q_2)$ states. This way the state component tracking the stuck machine's state will be frozen while the other can continue the computation. □

Theorem 8. *The class of languages accepted by Fr1TASS is closed under complement.*

Proof. For any Fr1TASS that halts on all inputs, it is enough to switch final and non-final states to accept the complement of its language. However, these systems may go into infinite loops, so a system accepting the complement needs to be able to detect that behavior. Each Fr1TASS can be completed with a 'sink state', that is a state from which no other is reachable, and we can direct the transitions for any previously missing state-letter pair into that state. Additionally, we make $n+1$ copies of each state p, say, $p_1,\ldots,p_{n+1}$, where n is the number of states originally. For each i, the states indexed with i are connected among them according to the original transition function, that is $\delta'(p_i,a) = (q_i,b)$ if $\delta(p,a) = (q,b)$, where δ and δ' are the transition functions of the original machine and of the machine for the complement, respectively. In the beginning, we mark the start of the input word with a special symbol to be able to keep track of it. Since the first symbol may need to be changed during the computation of the original machine, we add a marked copy of the original tape alphabet which will only be used to rewrite the first position. Whenever we read the start symbol in some state p_i, we continue on the p_{i+1} states until one of two things happens:

- We change one of the cells on the tape. In this case we continue the computation on the p_1 copies of the states until we reach the start mark.
- We reach the start mark from a p_{n+1} state. This means that the machine made n sweeps without changing any cell on the tape, so the original machine would go into an infinite loop. Instead, here we can simply transition to the sink state defined earlier.

The machine for the complement will have the newly introduced sink state as its only final state. □

Interestingly, the state complexity of intersection and union can be reduced at the expense of time complexity. This is because we can process the input twice instead of simulating both machines in

parallel. First we process it according to the rules of the first machine, and then do so according to the rules of the second one. In order to do this, the number of states in the simulating system only needs to be the sum of the size of state sets of the two starting machines, instead of their product. Moreover, if we 'recycle' the states, the size of the machine for the intersection/union need not increase beyond a constant plus the size of the larger machine participating in the intersection/union. When constructing the machine C to accept $L(A) \cap L(B)$ (or $L(A) \cup L(B)$, respectively), we can reuse the states of the machine by having a tape alphabet with two tracks, say blue and red. Then, we can draw the red transitions completely independent of the blue transitions using the same states as vertices, therefore realizing C on $\max\{|A|,|B|\}+k$ states. The additive constant term k is needed, because after we finish simulating the first machine, we need to freeze the first track of the tape which requires some extra states to cycle through the input and mark each position frozen in the first track. We need to keep track of whether the first machine accepted or rejected the input. We can achieve this without extra states, though, by performing short-circuit evaluation: if the operation is intersection and the first machine rejects then we can reject right away; hence, if the simulation continues to the second machine, we know the first was accepted. The case for union can be treated analogously.

Regarding the regular operations concatenation and Kleene-star, the class of languages accepted by Fr1TASS is probably not closed, but we do not have the tools at present to prove that. In particular, we do not have any necessary conditions for a language to be accepted by Fr1TASS beyond the time complexity bound $O(n^2)$ mentioned earlier, and that bound is not enough to prove negative results regarding closure. The reason we think that the class is not closed under concatenation and Kleene-star is that in general such closure results require either non-deterministically guessing a decomposition of the input into factors of the constituent languages or the possibility of trying all possible decompositions. Neither option seems possible with Fr1TASS.

5 Decision problems and minimal Fr1TASS

Using a construction similar to the freezing 1TASS accepting $\{\#w\#w \mid w \in \Sigma^*\}$, we will show how to reduce the Post Correspondence Problem (PCP) to the emptiness of freezing 1TASS languages. From that we can deduce that emptiness, universality ($= \Sigma^*$) and equivalence are undecidable for this model. We will argue that the undecidability of equivalence also strongly suggests that finding minimal freezing 1TASS for a given language cannot be algorithmically accomplished.

An instance of PCP consists of two sets of words $U = \{u_1,\ldots,u_n\}$ and $V = \{v_1,\ldots,v_n\}$ and the instance is positive if there exists some finite sequence $k_1,\ldots,k_\ell$ (a solution), with $k_i \in \{1,\ldots,n\}$, such that $u_{k_1}\cdots u_{k_\ell} = v_{k_1}\cdots v_{k_\ell}$. It is a well-known fact that it is undecidable whether an instance of PCP is positive ([13]).

Let us fix the alphabet of the PCP instance as Γ, that is, $U,V \subseteq \Gamma^*$, and let $\Gamma' = \{1,\ldots,n\} \cup \Gamma$. The alphabet of the machine will be $\Sigma = \{\#\} \cup \bigcup_{a\in\Gamma'}\{a,\overline{a}\}$. Choose any ordering of the alphabet such that $a < \overline{a}$ for each $a \in \Gamma'$. We construct a freezing 1TASS that accepts the language

$$\{\#k_1\cdots k_\ell\#u_{k_1}\cdots u_{k_\ell}\#v_{k_1}\cdots v_{k_\ell} \mid u_{k_1}\cdots u_{k_\ell} = v_{k_1}\cdots v_{k_\ell}\},$$

where $k_i,u_i,v_i \in (\bigcup_{a\in\Gamma'}\{\overline{a}\})^*$. The machine needs to check whether the input satisfies the following three conditions: (1) the middle part is indeed $u_{k_1}\cdots u_{k_\ell}$, (2) the last part is indeed $v_{k_1}\cdots v_{k_\ell}$ and (3) check whether $u_{k_1}\cdots u_{k_\ell} = v_{k_1}\cdots v_{k_\ell}$. As (2) can be done the same way as (1) and in parallel to it, and (3) has been illustrated before as the machine for $\{\#w\#w\}$, we will only detail (1). Figure 3 illustrates the part of the system for checking (1). Since the factor between the second and third # and the one after the

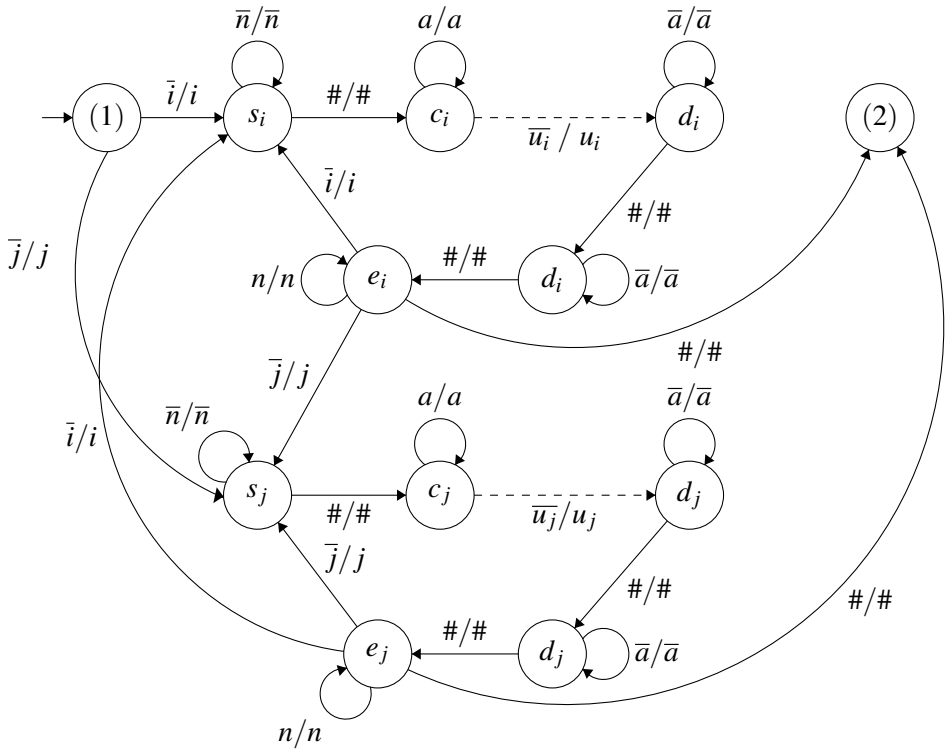


Figure 3: The parts of the 1TASS for matching the first portion of the input containing the numbers, $k_1 \cdots k_\ell$, to the second portion, $u_{k_1} \cdots u_{k_\ell}$. If the number read is i, that is, the symbol $\overline{i}$, then we continue from s_i, if it is $\overline{j}$ then continue from s_j, and so on. Then, the machine looks for the # symbol after which it ignores the already matched parts of $u_{k_1} \cdots u_{k_\ell}$. Finding the first unmatched symbols, it matches them against u_i, after which it returns to the beginning and reads the next number.

third # needs to be checked twice, first for (1) and (2), respectively, then for (3), all the input except the separators # needs to be marked by overlines at the beginning.

1. Checking whether a word equals $u_i = x_1 \cdots x_m$ is easy: we set up $m+1$ states $q_0, \ldots, q_m$ such that $\delta(q_i, \overline{x_i}) = (q_{i+1}, x_i)$. For all $a \neq x_i$, the state q_i has no outgoing transitions, therefore immediately rejecting the input on reading those letters.
2. We read the first unmatched number after the first #, say i. We move without changing over all following symbols until we reach the next #. Then move over all matched symbols, i.e., symbols without overline. From the first symbol with overline, we match u_i to the input, as above. If successful, move over all following symbols until we meet the second #. Move over all symbols without overline and start the process again.

This 1TASS will accept the solutions to the PCP instance, if any. Since PCP is undecidable, deciding whether the language accepted by a freezing 1TASS is empty, is also undecidable. This means that language equivalence is undecidable: if we let freezing 1TASS A and B be such that A does not accept any input, while B accepts the solutions of a PCP instance, then deciding equivalence amounts to deciding whether the PCP instance is positive. Similarly, if we let $L(C) = \Sigma^* \setminus L(B)$, where B accepts the solutions to a PCP instance, then a decision algorithm that could tell whether $L(C) = \Sigma^*$, would decide whether

the PCP instance has solutions, so universality is also undecidable.

Minimization of 1TASS. From the undecidability of the language equivalence, we can draw certain conclusions regarding the minimization of such systems. Say we define minimal 1TASS as ones having the fewest number of states and/or transitions. We instantly get that the following statements cannot both be true, otherwise equivalence would be decidable by the same isomorphism checking method as for DFA:

1. For each freezing 1TASS A there is a unique (up to renaming the states) minimal 1TASS B with $L(A) = L(B)$.
2. There is an algorithm to find for each freezing 1TASS A a minimal freezing 1TASS B with $L(A) = L(B)$.

If 1. holds then we cannot find the unique minimal system. Therefore we could assume that 1. does not hold and try to devise and algorithm for finding *a* minimal system.

Another possibility is to define minimal systems more tightly, in which case minimization algorithms might exist. We suggest the following possible alternative definitions for a freezing 1TASS A to be minimal:
1. A does not contain strongly equivalent states, i.e., states p,q such that $\delta(p,a) = \delta(q,a)$ for all $a \in \Sigma$. This case is straightforward to deal with along with any unreachable states, but yields little information about the similarity of Fr1TASS.
2. No proper subset of the system (removed transitions or states) accepts $L(A)$. Even the question whether minimality is decidable under this definition is nontrivial, let alone finding such a minimal system for a given Fr1TASS.

6 Fr1TASS with no auxiliary symbols

In this section we look at Fr1TASS that cannot have 'auxiliary' symbols (which cannot appear in the input, but can occur on the tape during the computation), that is, $\Sigma = \Gamma$. This type of restriction leads to dramatic changes in computing power even in the case of machines that can rewrite cells arbitrarily many times [6]. For these systems we can show that AS mode is incomparable to ET mode. Simulating AS with ET mode as done in the general case in Lemma 2 does not work. This is because we can no longer assume that the AS mode machines do not erase their tapes, as the technique used in Lemma 1 is not applicable anymore due to the lack of symbols that can stand in for erased ones. In fact, unary languages provide the proof that under the no-auxiliary-symbols restriction, AS and ET are incomparable.

Lemma 9. *For each Fr1TASS $A = (\{a\}, \{a\}, Q, q_0, F, \delta)$, the language $L(A)_{AS}$ accepted with accepting state is of the form $\{a^n \mid n \geq k\}$ for some fixed k, and the language $L(A)_{ET}$ accepted with empty tape is either finite or equal to a^*.*

Proof. Just like in the case of deterministic finite automata, such systems A have a transition diagram of a loop with a 'handle', due to determinism. There are two types of transitions possible: erasing, that is, $\delta(q,a) = (q', \lambda)$ and non-erasing, that is, $\delta(q,a) = (q', a)$. In AS mode the system accepts and halts as soon as it reaches a final state which means that any input longer than the distance from the initial state to the first final state will be accepted. In fact, any input with more letters than the number of erasing transitions on the path from initial to final state will also be accepted. In ET mode for the system to

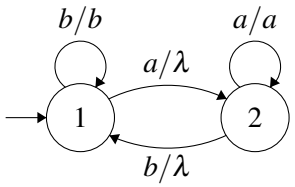


Figure 4: Fr1TASS accepting $\{w \mid |w|_b \leq |w|_a \leq |w|_b + 1\}$ without auxiliary symbols in ET mode.

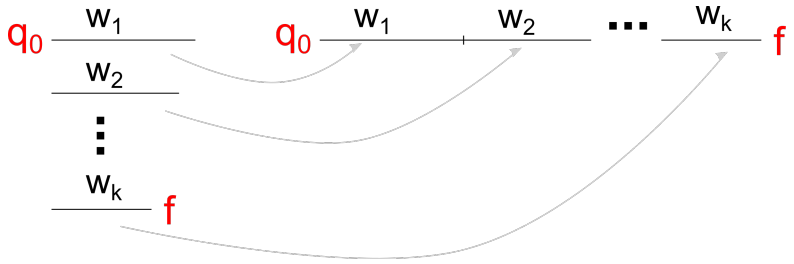


Figure 5: Flattening the computation of an AS mode system with no auxiliary symbols.

accept anything other than the empty word, it needs to have at least one erasing transition. If the only such transitions are on the 'handle', then the accepted language is finite, since the system can only erase finitely many symbols from the tape. If there is an erasing transition in the loop, then all inputs on which the machine reaches the loop will be accepted, since the machine will keep looping until all letters are erased. □

We can also show that AS mode cannot be strictly stronger than ET mode when the tape alphabet is at least binary. Consider the language $L_{ab} = \{w \mid |w|_b \leq |w|_a \leq |w|_b + 1\}$. A machine in ET mode can easily accept this language by reading an a, erasing it, moving to the right until it finds a corresponding b, erasing it and iterating this process (Fig. 4). However, using a 'computation flattening' argument we can prove that a machine in AS mode cannot accept this language.

Lemma 10. *There is no Fr1TASS* $A = (\Sigma, \Sigma, Q, q_0, F, \delta)$ *such that* $L(A)_{AS} = L_{ab}$.

Proof. Assume there is a Fr1TASS A as above that accepts L_{ab}. Take any $w \in L_{ab}$ and let the word on the tape in sweep i of the accepting computation on w be w_i, as defined in the preliminaries. As the word is accepted, there are finitely many, say k, sweeps. If we concatenate the words in the sweeps, we get $w' = w_1 \cdots w_k$. The obtained word w' is a valid input word, because the system can only use the symbols of the input alphabet. On the input w', the system A reaches the same accepting state as on the input w; thus w' is accepted, and the computation requires a single sweep. Moreover, the same final state is reached on input $w'aa$, too. However, this is a contradiction, since $w'aa$ cannot have the required numbers of letter occurrences if w' did. □

Language of palindromes. A very challenging problem is whether the Fr1TASS model can accept the language of palindromes over a non-unary alphabet. Intuitively the model should not be able to accept such a language for the reason described below, but we do not have a proof for this due to the lack of applicable necessary conditions. To verify whether the input is a palindrome, a machine would need to match pairs of letters at the same distance from the middle or from the start and end, respectively. Moreover, the matched pairs would need to be marked to keep track of which parts still need matching. However, in this model, we can only mark symbols *after* a pattern has been identified. This means that if

we start matching letters at the same distance from the middle, then the machine could not guess which is the next unmatched letter in the left half. Conversely, if the machine matches pairs based on their distance from the left and right end, respectively, then it could not guess the next unmatched letter in the right half of the input.

For the restricted model with no auxiliary symbols in AS mode, we can prove that the language of palindromes cannot be accepted, by using the computation flattening argument seen earlier. Let L_{pal} denote the language of palindromes over the binary alphabet $\{a,b\}$, that is, $L_{pal} = \{w \in \{a,b\}^* \mid w = w^R\}$ where w^R is the reverse of w, that is, if $w = a_1 \cdots a_n$ then $w^R = a_n \cdots a_1$.

Theorem 11. *For any Fr1TASS $A = (\Sigma, \Gamma, Q, q, F, \delta)$ we have $L_{pal} \neq L(A)_{AS}$.*

Proof. Suppose that there were a Fr1TASS $A = (\Sigma, \Gamma, Q, q, F, \delta)$ that accepts L_{pal} in accepting state mode. Consider a long palindrome of the form $a^n bwba^n$, where $n > |Q|$, which is accepted by the system in k sweeps. Again, let w_i denote the word on the tape at the beginning of sweep i. Concatenating those words yields the valid input $w' = w_1 \cdots w_k$, which will be accepted with the same transitions as w, but all in one sweep. Since w' is accepted, it must be a palindrome by our assumption, which means that its suffix must be ba^n. Due to the fact that n is larger than the number of states in A, while reading the suffix a^n, the system must enter some state p more than once, reading a^ℓ for some $\ell \geq 1$, between the first two traversals of p. However, this means that the system accepts also words of the form $w'a^{i\ell}$, for all $i \geq 0$. This results in a contradiction for $i = 1$, because the suffix of $w'a^\ell$ is $ba^{n+\ell}$ while its prefix is $a^n b$. Thus, non-palindromes would be also accepted by A. □

7 Concluding remarks

Apart from the decision problems in Section 5, our results have mostly been positive. To establish the limits of the accepting power of Fr1TASS we need negative results separating Fr1TASS languages from other language classes. Although these systems can process symbols in the same position repeatedly, we think that the freezing property allows some form of a pumping lemma, perhaps in combination with a computation flattening argument seen in Section 6. Obtaining such a tool seems quite challenging and will be our main focus in future studies on the topic.

Perhaps with a tool as described above or adapting the Kolmogorov complexity argument of Li et al. [10], one could prove that the language of palindromes is not a Fr1TASS language. This is intuitively a fundamental limitation of such systems with first-in-first-out (FIFO) nature of processing and such questions have proved interesting in their own right with respect to other FIFO style models [11]. If indeed palindromes cannot be accepted with Fr1TASS, then the language class is in some sense a natural counterpart of the class of context-free languages: membership is decidable efficiently and it contains the FIFO-like copy language instead of the LIFO-like palindromes. Interestingly, if one allows Fr1TASS to have non-determinism, then this FIFO limitation seems to vanish: such Fr1TASS could now guess which letters form pairs at equal distance from a reference point (middle or the ends) and verify the guess by marking symbols. The power of nondeterministic Fr1TASS is another topic worth further exploration in our opinion.

Finally, we would like to mention a computational complexity aspect that could be investigated with respect to Fr1TASS. The computation happens in sweeps and those sweeps are a natural resource to measure as the complexity of a given computation. Based on the amount of this resource used, one can define and study asymptotic complexity classes similarly to the case of iterated uniform finite transducers [8] and one-way jumping finite automata [4].

References

[1] Hiroyuki Chigahara, Szilárd Zsolt Fazekas & Akihiro Yamamura (2016): *One-Way Jumping Finite Automata*. *Int. J. Found. Comput. Sci.* 27(3), p. 391, doi:10.1142/S0129054116400165.

[2] John Cocke & Marvin Minsky (1964): *Universality of Tag Systems with $P = 2$*. *Journal of the ACM* 11(1), pp. 15–20, doi:10.1145/321203.321206.

[3] Matthew Cook (2004): *Universality in Elementary Cellular Automata*. *Complex Systems* 15, pp. 1–40, doi:10.25088/ComplexSystems.15.1.1.

[4] Szilárd Zsolt Fazekas, Robert Mercas & Olívia Wu (2022): *Complexities for Jumps and Sweeps*. *J. Autom. Lang. Comb.* 27(1-3), pp. 131–149, doi:10.25596/jalc-2022-131.

[5] Cody Geary, Pierre-Étienne Meunier, Nicolas Robertabanel & Shinnosuke Seki (2018): *Proving the Turing Universality of Oritatami Co-Transcriptional Folding*. In: *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, *LIPIcs* 123, pp. 23:1–23:13, doi:10.4230/LIPIcs.ISAAC.2018.23.

[6] Lane A. Hemaspaandra, Proshanto Mukherji & Till Tantau (2005): *Context-Free Languages Can Be Accepted with Absolutely No Space Overhead*. *Information and Computation* 203(2), pp. 163–180, doi:10.1016/j.ic.2005.05.005.

[7] A. B. Kahn (1962): *Topological Sorting of Large Networks*. *Commun. ACM* 5(11), p. 558–562, doi:10.1145/368996.369025.

[8] Martin Kutrib, Andreas Malcher, Carlo Mereghetti & Beatrice Palano (2022): *Descriptional Complexity of Iterated Uniform Finite-State Transducers*. *Information and Computation* 284, p. 104691, doi:10.1016/j.ic.2021.104691. Selected Papers from DCFS 2019, the 21st International Conference on Descriptional Complexity of Formal Systems.

[9] Martin Kutrib, Andreas Malcher & Matthias Wendlandt (2018): *Queue Automata: Foundations and Developments*, pp. 385–431. Springer International Publishing, Cham, doi:10.1007/978-3-319-73216-9_19.

[10] Ming Li, Luc Longpré & Paul Vitányi (1992): *The Power of the Queue*. *SIAM Journal on Computing* 21(4), pp. 697–712, doi:10.1137/0221042.

[11] J. Andres Montoya (2015): *Open Problems Related to Palindrome Recognition: Are There Open Problems Related to Palindrome Recognition?* *J. Autom. Lang. Comb.* 20(1), p. 5–25, doi:10.25596/jalc-2015-005.

[12] Daria Pchelina, Nicolas Schabanel, Shinnosuke Seki & Guillaume Theyssier (2022): *Oritatami Systems Assemble Shapes No Less Complex Than Tile Assembly Model (aTAM)*. In: *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, *LIPIcs* 219, pp. 51:1–51:23, doi:10.4230/LIPIcs.STACS.2022.51.

[13] Emil L. Post (1946): *A Variant of a Recursively Unsolvable Problem*. *Bull. Amer. Math. Soc.* 52, pp. 264–268, doi:10.1090/S0002-9904-1946-08555-9.

[14] Edward Szpilrajn (1930): *Sur l'extension de l'ordre partiel*. *Fundamenta Mathematicae* 16, pp. 386–389, doi:10.4064/fm-16-1-386-389.

[15] Kyle E. Watters, Eric J. Strobel, Angela M. Yu, John T. Lis & Julius B. Lucks (2016): *Cotranscriptional Folding of a Riboswitch at Nucleotide Resolution*. *Nature Structural and Molecular Biology* 23(12), pp. 1124–1131, doi:10.1038/nsmb.3316.

[16] Sheng Yu (1989): *A Pumping Lemma for Deterministic Context-Free Languages*. *Information Processing Letters* 31(1), pp. 47–51, doi:10.1016/0020-0190(89)90108-7.

[17] Charles Zaiontz (1976): *Circular Automata*. In: *Proceedings of the 14th Annual Southeast Regional Conference (ACM-SE 14)*, pp. 350–354, doi:10.1145/503561.503635.

When Stars Control a Grammar's Work

Henning Fernau
Abteilung Informatikwissenschaften
Universität Trier, Germany
fernau@uni-trier.de

Lakshmanan Kuppusamy
School of Computer Science and Engineering
VIT, Vellore, India
klakshma@vit.ac.in

Indhumathi Raman
Department of Computing Technologies,
School of Computing, SRMIST, Chennai, India
indhumar2@srmist.edu.in

Graph-controlled insertion-deletion (GCID) systems are regulated extensions of insertion-deletion systems. Such a system has several components and each component contains some insertion-deletion rules. The components are the vertices of a directed control graph. A rule is applied to a string in a component and the resultant string is moved to the target component specified in the rule. The language of the system is the set of all terminal strings collected in the final component. We impose the restriction in the structure of the underlying graph to be a star structure where there is a central, control component which acts like a master and transmits a string (after applying one of its rules) to one of the components specified in the (applied) rule. A component which receives the string can process the obtained string with any applicable rule available in it and sends back the resultant string only to the center component. With this restriction, we obtain computational completeness for some descriptional complexity measures.

1 Introduction

Insertion-deletion systems are part of formal languages which are extensively analyzed. The motivation for the systems comes from both linguistics [14, 16] and molecular biology. The action of inserting or deleting some strands do occur often in DNA processing [17] and RNA editing [2]. These two operations together were introduced as a formal languages theory framework in [11] and further studied in [10, 19]. The corresponding grammatical mechanism is called *insertion-deletion system* (abbreviated as ins-del system). The insertion operation means inserting a string η in between the strings w_1 and w_2, whereas the deletion operation is deleting a substring δ from the string $w_1\delta w_2$.

In the literature, several variants of ins-del systems have been considered. We refer to the survey article [20] for details concerning the state-of-the-art around 2010. One of the important variants of ins-del systems is *graph-controlled ins-del systems* (abbreviated as GCID systems), introduced in [6] and further studied in [8]. In such a system, the concept of *components* is introduced, which are associated with insertion or deletion rules. The transition is performed by choosing any applicable rule from the set of rules of the current component and by moving the resultant string to the target component specified in the rule in order to continue processing it. Several restrictions of graph control have been studied, e.g., matrix ins-del systems (see [18, 5] and more papers cited there), time-varying ins-del systems [1], or path-controlled ins-del systems [4]. In this paper, we consider star control (which also has been considered in [9] in an implicit way when dealing with graph-controlled systems with two components. This models a kind of master-slave system in the sense that the central component always dispatches work to the slave components who, after finishing their work, return the result to the master component.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 96–111, doi:10.4204/EPTCS.386.9

k	=	the number of components			
n	=	$\max\{\|\eta\|\colon (i,(u,\eta,v)_I,j)\in R\}$	m	=	$\max\{\|\delta\|\colon (i,(u,\delta,v)_D,j)\in R\}$
i'	=	$\max\{\|u\|\colon (i,(u,\eta,v)_I,j)\in R\}$	j'	=	$\max\{\|u\|\colon (i,(u,\delta,v)_D,j)\in R\}$
i''	=	$\max\{\|v\|\colon (i,(u,\eta,v)_I,j)\in R\}$	j''	=	$max\{\|v\|\colon (i,(u,\delta,v)_D,j)\in R\}$

Table 1: Size $(k;n,i',i'';m,j',j'')$ of a GCID system

Graph-controlled insertion-deletion systems whose underlying control graph is a tree are equivalent to ins-del P systems [13, 8]. Hence, our restriction can be viewed as a special case of ins-del P systems. We want to point to one technicality here: with P systems (and similarly with several restrictions of graph control), there is the possibility to stay in the same membrane with the *here* command (which corresponds to allowing loops in graph control); in the model that we consider in this paper, this is disallowed: when the master saw the 'current work' (and worked on it one step), it has to pass it to some slave immediately, and after the slave performed one step, the work is handed back to the master, etc. Therefore, results in the literature concerning seemingly related models do not always compare well to this star model.

The *descriptional complexity* of a GCID system is measured by its *size* $s=(k;n,i',i'';m,j',j'')$, where the parameters represent resource bounds as given in Table 1. Slightly abusing notation, the language class generated by GCID systems of size s is denoted by $\mathrm{GCID}(s)$. We attach subscripts P and S when referring to path-controlled or star-controlled GCID systems, respectively.
The main results of this paper are the following ones.

1. (Theorem 9) $\mathrm{RE}=\mathrm{GCID}_S(6;1,1,0;2,0,0)=\mathrm{GCID}_S(6;1,0,1;2,0,0)$;
2. (Theorem 11) $\mathrm{RE}=\mathrm{GCID}_S(4;2,1,1;1,0,0)$.

Our proofs are based on the *Special Geffert Normal Form* of type-0 grammars, which characterizes the class RE, the recursively enumerable languages. Formal definitions follow in the next section.

2 Preliminaries

We assume that the readers are familiar with the standard notations used in formal language theory. Here, we recall a few notations for the understanding of the paper. Let $\mathbb{N}$ denote the set of positive integers, and $[\ell\dots k]=\{i\in\mathbb{N}\colon \ell\le i\le k\}$. Given an *alphabet* (finite set) Σ, Σ^* denotes the free monoid generated by Σ. The elements of Σ^* are called *strings* or *words*; λ denotes the empty string. For a string $w\in\Sigma^*$, $|w|$ is the length of w and w^R denotes the reversal (mirror image) of w. L^R and $\mathscr{L}^R$ are also understood for languages L and language families $\mathscr{L}$, collecting all reversals of words from L and all reversals of languages from $\mathscr{L}$, respectively. For the computational completeness results, as our main tool we use the fact that type-0 grammars in Special Geffert Normal Form (SGNF) describe RE.

Definition 1 *A type-*0 *grammar* $G=(N,T,P,S)$ *is said to be in SGNF if*

- *N decomposes as $N=N'\cup N''$, where $N''=\{A,B,C,D\}$ and N' contains at least the two nonterminals S and S';*
- *the only non-context-free rules in P are $AB\to\lambda$ and $CD\to\lambda$;*
- *the context-free rules are of the form (i) $S'\to\lambda$, or (ii) $X\to Y_1Y_2$, where $X\in N'\setminus\{S'\}$ and $Y_1Y_2\in((N'\setminus\{X\})(T\cup\{B,D\}))\cup(\{A,C\}(N'\setminus\{X\}))$.*

The way the normal form is constructed is described in [6], based on [7]. We can assume that S' does not appear on the left-hand side of any non-erasing rule. This also means that the derivation in G undergoes two phases. In phase I, only context-free rules are applied. This phase ends with applying the context-free deletion rule $S' \to \lambda$ (which is the only rule that has S' on its left-hand side). Then only, non-context-free deletion rules $AB \to \lambda$ and $CD \to \lambda$ are applied in phase II. Notice that the symbol from N', as long as present, separates A and C from B and D; this prevents a premature start of phase II. One of the features of SGNF derivations is that any string that can be derived can contain at most one substring AB or CD in its so-called *center*. If such a substring is present, the derivation is in phase II; also, then no nonterminal from N' occurs. In phase I, exactly one such nonterminal is present (in the center). Therefore, we can differentiate two cases within (ii) for $X,Y \in N' \setminus \{S'\}$ with $X \neq Y$: either, we have a rule $X \to bY$, with $b \in \{A,C\}$, or we have a rule $X \to Yb$, with $b \in T \cup \{B,D\}$. This case distinction is often necessary when simulating type-0 grammars in SGNF, as we will see later.

We write $\Rightarrow_r$ to denote a single derivation step using rule r, and $\Rightarrow_G$ (or $\Rightarrow$ if no confusion arises) denotes a single derivation step using any rule of G. Then, $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$, where $\Rightarrow^*$ is the reflexive transitive closure of $\Rightarrow$.

2.1 Graph-Controlled Insertion-Deletion Systems

Definition 2 *A* graph-controlled insertion-deletion system *(GCID for short) with k components is a construct* $\Pi = (k,V,T,A,H,i_0,F,R)$*, where (i) k is the number of components, (ii) V is an alphabet, (iii)* $T \subseteq V$ *is the terminal alphabet, (iv)* $A \subset V^*$ *is a finite set of strings, called axioms, present in the initial component, (v) H is a set of labels associated (in a one-to-one manner) to the rules in R, (vi)* $i_0 \in [1 \ldots k]$ *is the initial component, (vii)* $F \subseteq [1 \ldots k]$ *is the set of final components and (viii) R is a finite set of rules of the form* $l : (i,r,j)$*, where* $l \in H$ *is the label of the rule, r is an insertion rule of the form* $(u,\eta,v)_I$*, with insertion string* η *and context* (u,v)*, or deletion rule of the form* $(u,\delta,v)_D$*, with deletion string* δ *and context* (u,v)*, where* $u,v \in V^*$*,* $\eta,\delta \in V^+$ *and* $i,j \in [1 \ldots k]$.

Often, the component is part of the label name. This will also (implicitly) define H. We shall omit the label l of the rule wherever it is not necessary for the discussion.

We now describe how GCID systems work. Applying an insertion rule of the form $(u,\eta,v)_I$ means that the string η is inserted between u and v; this corresponds to the rewriting rule $uv \to u\eta v$. Similarly, applying a deletion rule of the form $(u,\delta,v)_D$ means that the string δ is deleted between u and v; this corresponds to the rewriting rule $u\delta v \to uv$. A *configuration* of Π is represented by $(w)_i$, where $i \in [1 \ldots k]$ is the number of the current component and $w \in V^*$ is the current string. We also say that w has entered or moved to component Ci. We write $(w)_i \Rightarrow_l (w')_j$ if there is a rule $l : (i,r,j)$ in R, and w' is obtained by applying the insertion or deletion rule r to w. For brevity, we write $(w)_i \Rightarrow (w')_j$ if there is some rule l in R such that $(w)_i \Rightarrow_l (w')_j$. To avoid confusion with traditional grammars, we write $\Rightarrow_*$ for the transitive reflexive closure of $\Rightarrow$ between configurations. The language $L(\Pi)$ generated by Π is defined as $\{w \in T^* \mid (x)_{i_0} \Rightarrow_* (w)_{i_f}$ for some $x \in A$ and some $i_f \in F\}$.

The *underlying control graph* of a GCID system Π with k components is defined to be a graph with k nodes labelled $C1$ through Ck and there exists a directed edge from Ci to Cj if there exists a rule of the form (i,r,j) in R of Π. We also associate an undirected graph on k nodes to a GCID system of k components as follows: there is an undirected edge from a node Ci to Cj if there exists a rule of the form (i,r_1,j) or (j,r_2,i) in R of Π. We call a GCID system with k components *star-controlled* if its underlying undirected control graph has the edge set $\{\{C1,Ci\} \mid i \in [2 \ldots k]\}$. This means that the corresponding directed control graph may contain arcs like $(C1,Ci)$, $(Ci,C1)$, but no loops.

Below, we provide a few examples for a better understanding of how the above-defined system works. As star-controlled systems have to have at least two components to produce anything non-trivial, it is interesting to observe that even with only two components, non-regular languages can be obtained.

Example 3 *The language* $\{a^n b^n \mid n \geq 0\}$ *can be generated by a star-controlled insertion-deletion system with two components, alphabet* $\{a,b,A,B\}$*, the axiom set* $\{AB\}$ *in C1, final component* $\{C1\}$ *and the following rules:* $r1.1 : (1,(A,a,\lambda)_I,2)$*,* $r2.1 : (2,(B,b,\lambda)_I,1)$*,* $r1.2 : (1,(\lambda,A,\lambda)_D,2)$ *and* $r2.2 : (2,(\lambda,B,\lambda)_D,1)$*. A possible derivation of a terminal string is:*

$$(AB)_1 \Rightarrow_{r1.1} (AaB)_2 \Rightarrow_{r2.1} (AaBb)_1 \Rightarrow_{r1.2} (aBb)_2 \Rightarrow_{r2.2} (ab)_1.$$

Observe that $(aBb)_2 \Rightarrow_{r2.1} (aBbb)_1$ *is possible, but now the derivation is stuck, as any rule in* $C1$ *checks for the presence of the nonterminal A. Yet, as the nonterminal B is present, the configuration* $(aBbb)_1$ *cannot lead to a terminal word. The size of this system is* $(2;1,1,0;1,0,0)$*. A very similar system can be given for this language that is of size* $(2;1,0,1;1,0,0)$*. For the very similar language* $\{a^n b^n \mid n \geq 1\}$*, even a system with two rules* $r1 : (1,(a,a,\lambda),2)$ *and* $r2 : (2,(b,b,\lambda),1)$ *would suffice, with axiom ab.*

Recall that the class REG of regular languages is the lowest class of the Chomsky hierarchy. It can be characterized by right-linear grammars whose rules have the form $A \to Ba$ or $A \to \lambda$ for nonterminal symbols A,B and a terminal symbol a. We use this characterization to prove that star-controlled GCID systems can generate all regular languages. The previous example then shows that even non-regular languages can be generated.

Theorem 4 *Each regular language (and also some non-regular languages) can be generated by a* GCID_S *system of size* $(2;3,0,1;2,0,0)$*, where the initial component* $C1$ *is also the only final one.*

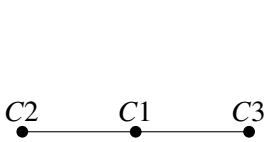
Later, we will see that with both components being final, many more languages can be described.
Proof We only sketch the construction in the following. For each rule of the form $A \to aB$ of a right-linear grammar G that we start with, we introduce the insertion rule $(\lambda, aB\$, A)$ into the first component of the simulating GCID_S system Π. For each erasing rule $A \to \lambda$, we add the insertion rule $(\lambda, a\$, A)$ into the first component of Π. In both cases, \$ is a special marker symbol that is taken care of in the second component that contains all possible deletion rules of the form $(\lambda, \$A, \lambda)$ for any nonterminal A of G. For instance, if G contains the rules $S \to aX$ and $X \to \lambda$, enabling the derivation $S \Rightarrow aX \Rightarrow a$, then the simulation is performed as follows: $(S)_1 \Rightarrow (aX\$S)_2 \Rightarrow (aX)_1 \Rightarrow (a\$X)_2 \Rightarrow (a)_1$. □

By adding more nonterminal symbols, one can also achieve this result with GCID_S systems of size $(2;2,0,1;2,0,0)$. We leave it open if GCID_S systems with only two components and only one final component can generate each recursively enumerable language.

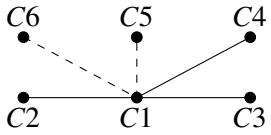
Example 5 *The copy language* $\{ww \mid w \in \{a,b\}^*\}$ *can be generated by a star-controlled insertion-deletion system* $\Pi = (3;\{a,b,A,B\},\{a,b\},\{AB\},H,1,\{1\},R)$*, where* $H = \{r1.1, r1.2, r1.3, r2.1, r2.2, r3.1\}$ *and R is the set of rules depicted in Table 2;* Π *has size* $(3;1,0,1;1,0,0)$*. Starting with the axiom AB in* $C1$*, if we apply* $r1.3$*, then we can apply* $r2.2$ *only in* $C2$ *and that produces* λ *in* $C1$*. The nonterminals A and B serve as markers and if an a is introduced to the left of A in* $C1$ *(by* $r1.1$*), then one a is introduced to the left of B (by* $r2.1$*) in* $C2$*. Similarly, if one b is introduced to the left of A in* $C1$ *(by* $r1.2$*), then a b is introduced to the left of B (by* $r3.1$*) in* $C3$*. This guarantees to have the pattern of the copy language produced by the system* Π*. But, there is a caveat here. If one can apply* $r1.1$ *in* $C1$*, then in* $C2$*,* $r2.2$ *can also be applied and in such a case, the pattern of the copy language is not followed. However, then back in* $C1$*, only* $r1.3$ *can be applied, which means for the string to move to* $C2$ *and there, the derivation stops.*

Component C1	Component C2	Component C3
$r1.1 : (1,(\lambda,a,A)_I,2)$	$r2.1 : (2,(\lambda,a,B)_I,1)$	$r3.1 : (3,(\lambda,b,B)_I,1)$
$r1.2 : (1,(\lambda,b,A)_I,3)$	$r2.2 : (\lambda,A,\lambda)_D,1)$	
$r1.3 : (1,(\lambda,B,\lambda)_D,2)$		

Table 2: Star-controlled ins-del system for generating $\text{Copy}_{a,b} = \{ww \mid w \in \{a,b\}^*\}$.



(a) Star-shaped control graphs underlying the star-controlled systems depicted in Example 5.



(b) Control graphs underlying the star-controlled systems in our main theorems.

Figure 1: Control graphs underlying different GCID systems in this paper.

As C2 is not a final component, by definition the strings over the terminal alphabet $\{a,b\}$ that are not also leading into the final component are not collected into the language generated by Π.

A sample derivation for aabaab is given below.

$$(AB)_1 \Rightarrow_{r1.1} (aAB)_2 \Rightarrow_{r2.1} (aAaB)_1 \Rightarrow_{r1.1} (aaAaB)_2 \Rightarrow_{r2.1} (aaAaaB)_1 \Rightarrow_{r1.2} (aabAaaB)_3$$
$$\Rightarrow_{r3.1} (aabAaabB)_1 \Rightarrow_{r1.3} (aabAaab)_2 \Rightarrow_{r2.2} (aabaab)_1.$$

The control graph underlying the construction is shown in Figure 1.

3 Computational Completeness

In this section, we present the main results of our paper. First, we discuss some limitations for getting computational completeness results and then, we describe two important cases of resource restrictions that characterize RE.

3.1 GCID_S systems with insertion and deletion length one

In [19], it has been proved that ins-del systems with size (1,1,1;1,1,1) characterize RE. Notice that it is proved in [12, 15] that ins-del systems of size $(1,1,1;1,1,0)$ or $(1,1,0;1,1,1)$ cannot characterize RE. It is therefore obvious that we need at least 2 components in a graph-controlled ins-del system of sizes $(1,1,1;1,1,0)$ and $(1,1,0;1,1,1)$ to characterize RE. In [3], we characterized RE by path-controlled GCID systems of size $(k;1,i',i'';1,j',j'')$ for different combinations of $k \geq 1, i', i'', j', j''$.

However, if we impose star structure as the underlying control graph and the resultant string has to move in/move out during every derivation step, then it is interesting to notice that the context-free rules of SGNF, namely $p : X \to bY$, $q : X \to Yb$ and $h : S' \to \lambda$ can never be *directly simulated* by rules of $\text{GCID}_S(k;1,i',i'';1,j',j'')$ for any value of $k \geq 2, i', i'', j', j'' \geq 0$. Here, by a *direct simulation* of a rule r, we mean that, assuming a sentential form w may yield the sentential form v by applying rule r within the original grammar (which is, in our case, in SGNF), then the simulating star-controlled GCID system will

start in the central component $C1$ with the sentential form w and derive after a number of steps, possibly, during the simulation, introducing and deleting symbols specific to r (called *markers* in the following), the sentential form v and moving back to $C1$ to be ready for the next simulation step.

Proposition 6 *The context-free rules of a grammar in SGNF, namely $p : X \to bY$ and $q : X \to Yb$ (with $X \neq Y$), as well as $h : S' \to \lambda$, can never be directly simulated by rules of* $\mathrm{GCID}_S(k;1,i',i'';1,j',j'')$ *for any value of $k \geq 2, i', i'', j', j'' \geq 0$.*

Proof To directly simulate $p : X \to bY$ using insertion-deletion rules, we need two insertion rules (one to insert b and one to insert Y; here we recall that the insertion length is 1) and one deletion rule to delete X. Hence, we need three *basic* insertion-deletion rules. Further, if we need to introduce $r \geq 1$ markers, then we can insert only one marker at a time using an insertion rule which will account for r insertion rules. At the end of the derivation, we need to delete all the r markers using r deletion rules (since we can only delete one symbol at a time). This amounts to having r insertion rules and r deletion rules to deal with the markers and 3 basic insertion-deletion rules to simulate $p : X \to bY$. This sums up to $2r+3$ rules.

We need to distribute these $2r+3$ rules among the k components of the GCID system. Let $C1$ be the central (initial and final) component. As the system is star-structured, in a rule $(i,(x,y,z)_\delta,j),\delta \in \{I,D\}, i,j \in [1\ldots k]$, we have $i \neq j$, as loops are not allowed, and $|\{i,j\} \cap \{1\}| = 1$. Hence, the order of rule applications in any derivation will start at the central node and then alternate between central and non-central nodes. Therefore, (i) the last rule of the simulation should be placed in a non-central component and not in $C1$ and (ii) the total number of rules for simulation is even. Since $2r+3$ is not even, the statement follows. □

By its definition, a derivation of a GCID_S system has to alternate between the central component and any other component. By putting exactly the same rules in two components and declaring one of the two components as being central, while both are final, one obtains:

Proposition 7 $\mathrm{GCID}(1;n,i',i'';m,j',j'') \subseteq \mathrm{GCID}_S(2;n,i',i'';m,j',j'')$ *holds for any value of $i', i'', j', j'' \geq 0$ and $n,m \geq 1$.*

For example, this entails $\mathrm{GCID}_S(2;1,1,1;1,1,1) = \mathrm{RE}$ and similar computational completeness results based on what is known for ins-del systems. By way of contrast, computational completeness results for 2-component graph-controlled systems do not necessarily carry over to our star-controlled systems, as there, 'loops' might be allowed.

3.2 GCID_s systems with insertion or deletion length of more than one

To simplify the presentation and proofs of our further results, the following observations from [3] are used, adapted to our case.

Proposition 8 *[3] Let k,n,i',i'',m,j,j'' be non-negative integers.*

1. $\mathrm{GCID}_S(k;n,i',i'';m,j',j'') = [\mathrm{GCID}_S(k;n,i'',i';m,j'',j')]^R$;
2. $\mathrm{RE} = \mathrm{GCID}_S(k;n,i',i'';m,j',j'')$ *iff* $\mathrm{RE} = \mathrm{GCID}_S(k;n,i'',i';m,j'',j')$.

Theorem 9 $\mathrm{RE} = \mathrm{GCID}_S(6;1,1,0;2,0,0)$ *and* $\mathrm{RE} = \mathrm{GCID}_S(6;1,0,1;2,0,0)$.

Proof Consider a type-0 grammar $G = (N,T,P,S)$ in SGNF as in Definition 1. We construct a GCID system $\Pi = (6,V,T,\{S\},H,1,\{1\},R)$ of size $(6;1,1,0;2,0,0)$ such that $L(\Pi) = L(G)$. The alphabet V contains rule markers, apart from the symbols of G. More specifically, for each rule $\gamma \to \delta \in P$ labeled r, we have $r \in V$. Moreover, if $\gamma \to \delta \neq S' \to \lambda$, we also have single-primed markers $r' \in V$. Finally, for

$C1$	$C2$	$C3$
$p1.1 : (1,(X,p,\lambda)_I,2)$ $p1.2 : (1,(p,p',\lambda)_I,3)$ $p1.3 : (1,(p',b,\lambda)_I,5)$ $p1.4 : (1,(\lambda,pp',\lambda)_D,4)$ $p1.5 : (1,(p''',p^{iv},\lambda)_I,4)$ $p1.6 : (1,(p''',Y,\lambda)_I,2)$	$p2.1 : (2,(\lambda,X,\lambda)_D,1)$ $p2.2 : (2,(\lambda,p''p''',\lambda)_D,1)$	$p3.1 : (3,(p',p^{v},\lambda)_I,1)$
$q1.1 : (1,(X,q,\lambda)_I,2)$ $q1.2 : (1,(q,q',\lambda)_I,3)$ $q1.3 : (1,(q',Y,\lambda)_I,4)$	$q2.1 : (2,(\lambda,X,\lambda)_D,1)$	$q3.1 : (3,(q',b,\lambda)_I,1)$
$h1.1 : (1,(S',h,\lambda)_I,2)$	$h2.1 : (2,(\lambda,S'h,\lambda)_D,1)$	
$f1.1 : (1,(A,f,\lambda)_I,6)$ $f1.2 : (1,(\lambda,fB,\lambda)_D,2)$	$f2.1 : (2,(\lambda,Af',\lambda)_D,1)$	
$g1.1 : (1,(C,g,\lambda)_I,6)$ $g1.2 : (1,(\lambda,gD,\lambda)_D,2)$	$g2.1 : (2,(\lambda,Cg',\lambda)_D,1)$	
$C4$	$C5$	$C6$
$p4.1 : (4,(p'',p''',\lambda)_I,1)$ $p4.2 : (4,(\lambda,p^{iv}p^{v},\lambda)_D,1)$	$p5.1 : (5,(b,p'',\lambda)_I,1)$	
$q4.1 : (4,(\lambda,qq',\lambda)_D,1)$		
		$f6.1 : (6,(B,f',\lambda)_I,1)$
		$g6.1 : (6,(D,g',\lambda)_I,1)$

Table 3: Star-controlled $\mathrm{GCID}_S(6;1,1,0;2,0,0)$ simulating the rules of SGNF.

context-free rules of the form $X \to bY$, even markers $r'', r''', r^{iv}, r^{v} \in V$. We refer to Table 3, showing the simulation of the different rule types of SGNF. The columns of the table correspond to the components of Π. The rows of Table 3 correspond to the rules simulating the 'linear rules' $p : X \to bY$ and $q : X \to Yb$, with $X \in N'$ and $b \in N'' \cup T$, $h : S' \to \lambda$, as well as $f : AB \to \lambda$ and $g : CD \to \lambda$.

We now prove that $L(G) \subseteq L(\Pi)$ as follows. We show that if $w \Rightarrow w'$ in G, then $(w)_1 \Rightarrow_* (w')_1$ according to Π. The claim then follows by induction.

Context-free rule $q : X \to Yb$. Here, $w = \alpha X \beta$ and $w' = \alpha Yb\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$. The simulation performs as follows:

$$\begin{aligned}(\alpha X\beta)_1 &\Rightarrow_{q1.1} (\alpha Xq\beta)_2 \Rightarrow_{q2.1} (\alpha q\beta)_1 \Rightarrow_{q1.2} (\alpha qq'\beta)_3 \Rightarrow_{q3.1} (\alpha qq'b\beta)_1 \\ &\Rightarrow_{q1.3} (\alpha qq'Yb\beta)_4 \Rightarrow_{q4.1} (\alpha Yb\beta)_1 .\end{aligned}$$

Context-free rule $p : X \to bY$. Here, $w = \alpha X \beta$ and $w' = \alpha bY\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$. The simulation performs as follows:

$$\begin{aligned}(\alpha X\beta)_1 &\Rightarrow_{p1.1} (\alpha Xp\beta)_2 \Rightarrow_{p2.1} (\alpha p\beta)_1 \Rightarrow_{p1.2} (\alpha pp'\beta)_3 \Rightarrow_{p3.1} (\alpha pp'p^{v}\beta)_1 \\ &\Rightarrow_{p1.3} (\alpha pp'bp^{v}\beta)_5 \Rightarrow_{p5.1} (\alpha pp'bp''p^{v}\beta)_1 \Rightarrow_{p1.4} (\alpha bp''p^{v}\beta)_4 \Rightarrow_{p4.1} (\alpha bp''p'''p^{v}\beta)_1 \\ &\Rightarrow_{p1.5} (\alpha bp''p'''p^{iv}p^{v}\beta)_4 \Rightarrow_{p4.2} (\alpha bp''p'''\beta)_1 \Rightarrow_{p1.6} (\alpha bp''p'''Y\beta)_2 \Rightarrow_{p2.2} (\alpha bY\beta)_1 .\end{aligned}$$

One might wonder that the $h : S' \to \lambda$ and the $f : AB \to \lambda$ rules could easily be simulated by the rules $(1,(\lambda,S',\lambda)_D,1)$ and $(1,(\lambda,AB,\lambda)_D,1)$, respectively. However, the underlying control graph of our star-controlled GCID forbids loops and hence, we have given a different simulation for these rules. Since the correctness of the h-rule simulation is trivial, it remains to discuss the simulation of the rule $f : AB \to \lambda$. The simulation of $g : CD \to \lambda$ is similar and hence omitted.

Non-context-free rules $f : AB \to \lambda$. This means that we expect $w = \alpha AB\beta$ and $w' = \alpha\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$. Within Π, this can be simulated as follows.

$$(\alpha AB\beta)_1 \Rightarrow_{f1.1} (\alpha AfB\beta)_6 \Rightarrow_{f6.1} (\alpha AfBf'\beta)_1 \Rightarrow_{f1.2} (\alpha Af'\beta)_2 \Rightarrow_{f2.1} (\alpha\beta)_1 .$$

Conversely, a derivation $(w)_1 \Rightarrow^* (w')_1$, with $w \neq w'$ and $w,w' \in \{A,C\}^*(N' \cup \{\lambda\})(\{B,D\} \cup T)^*$ has to start like $(w)_1 \Rightarrow (v)_j$ in Π. If some rule from $C1$ is applied to w, the rule will insert a rule marker into the string w and branch to $C2$ (when simulating context-free rules) or to $C3$ (when simulating non-context-free rules). The introduction of rule markers in $C1$ will take care of the non-interference among the non-context-free and context-free rules. We now discuss the possibilities in detail. In our discussion, we distinguish between sentential forms containing or not containing a symbol from N'.

Our inductive arguments will also show that, in Π, no sentential form is derivable that contains two occurrences of symbols from N'. More generally, we can show the following. Assume that we can derive some configuration $(w)_1$ in Π such that the string v contains no marker symbols, where $v = \mu(w)$ is obtained by applying the morphism μ that acts like the identity on V apart from the letters f, f', g, g' that will be erased. Then the sentential form v is also derivable in G. In particular, if $w \in T^*$, then $v = w$, i.e., each word in $L(\Pi)$ also belongs to $L(G)$.

We will also prove by induction that, if $(w)_1$ is derivable in Π and if w contains at most one occurrence of N' and no markers but possibly f, f', g, g', then also $\mu(w)$ is derivable in Π and then, only using the rules $f1.1$, $f6.1$, $g1.1$ and $g6.1$, we can derive $\mu(w)$ in Π from w. This also means that, in each such string w derivable in Π, the number of occurrences of symbols from $\{f,g\}$ equals the number of occurrences of symbols from $\{f',g'\}$. We also call this the *balance condition*. Therefore, we can start our inductive hypothesis with strings that can be derived both in G and in Π and observe the maintenance of the balance condition along our arguments.

Let us first assume (by induction) that the sentential form $w^1 = \alpha X\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$ and $X \in N'$ is derivable in G and the configuration $(w^1)_1$ is derivable in Π. We will prove (as induction step) that if $(w^1)_1 \Rightarrow_* (u)_1$, $w^1 \neq u$, and $u \in \{A,C\}^*(N' \cup \{\lambda\})(\{B,D\} \cup T)^*$ is the first sentential form from $\{A,C\}^*(N' \cup \{\lambda\})(\{B,D\} \cup T)^*$ that appears in a derivation of Π in $C1$, then $w^1 \Rightarrow u$ holds in G, except from a premature start of simulating non-context-free rules also discuss below and where we argue that the balance condition is maintained.

Applying $f1.1$ to $w^1 = \alpha X\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$ and $X \in N'$, the only nonterminal from N', is possible but unintended. (A similar discussion applies to $g1.1$.) A successful application yields the configuration $(w^2)_6$, with $w^2 = \alpha_1 Af\alpha_2 X\beta$, where $\alpha_1 A\alpha_2 = \alpha$. Now, the only applicable rules are $f6.1$ or $g6.1$.We get a configuration $(w^3)_1$ with $\mu(w^3) = w^1$. It can be observed that on w^3, neither $f1.2$ nor $g1.2$ are applicable, as these rules require AB or CD sitting in the center of w^1, which was not the case by assumption. Therefore, we could either continue our journey with inserting further markers f, f', g, g' (but always maintaining the balance condition) or finally apply $r1.1$, belonging to a context-free rule r. Now, observe that, instead of applying $r1.1$ and then $r2.1$ (because $f2.1$ and $g2.1$ are inapplicable), yielding a configuration $(u)_1$, we could also first apply $r1.1$ and $r2.1$ to w^1, and then the same f- and g-simulation rules as before, arriving at $(u)_1$ in a different way. This proves (here as part of the induction step) that, as claimed, we can exchange the sequence of rule applications in a way that we

apply f- and g-simulation rules after the other rules that are meant to simulate the context-free rules. We also see by induction that the balance condition is always maintained.

Applying $q1.1$ to $w^1 = \alpha X \beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$ and $X \in N'$, the only nonterminal from N', yields the configuration $(w^2)_2$ with $w^2 = \alpha X q \beta$. In $C2$, all rules (except $q2.1$ and $p2.1$) are guarded by markers and the only applicable rule are $q2.1$ or $p2.1$ (in case there is a rule $X \to b'Y'$ in P) which delete X, yielding $(w^3)_1 = (\alpha q \beta)_1$. Due to the rule markers, apart from the intended rule $q1.2$, one could also apply $f1.1$ or $g1.1$. In that case, having moved to $C6$, $f6.1$ or $g6.1$ are applicable, yielding some configuration $(w')_1$ with $\mu(w') = w^3$. As discussed before, we can even continue like this, but in order to make any further progress, we will have to apply $q1.2$ in some configuration $(w'')_1$ with $\mu(w'') = w^3$. As we could also apply the non-context-free simulation rules afterwards, it suffices to discuss what happens if we apply $q1.2$ to w^3 as intended. Hence, we arrive at the configuration $(w^4)_3$ with $w^4 = \alpha q q' \beta$. The required rule markers cause $q3.1$ to be the only applicable rule as desired. Therefore, we arrive at the configuration $(w^5)_1 = (\alpha q q' b \beta)_1$. Clearly, one could now (again) apply $f1.1$ or $g1.1$, but this would only lead to prematurely introducing the markers f, f', g, g' similar as discussed before, again always maintaining the balance condition. Therefore, the only applicable rule that needs to be discussed (apart from the intended one, which is $q1.3$) is $q1.2$ (again). With the string $\alpha q q' q' b \beta$, we are back to $C3$. Now, there are two possible subsequent configurations: (a) $(\alpha q q' b q' b \beta)_1$, or (b) $(\alpha q q' q' b b \beta)_1$. In Case (a), we claim that there is no way to delete the second occurrence of q' in the future. Namely, the only way to delete q' is if left to it, q is sitting. But as now some $b \in \{B,D\} \cup T$ is to the left of q', there is no way to introduce q in this position later, because the marker q always works as a symbol that replaces the former N'-symbol. Therefore, a derivation following (a) cannot terminate. The situation is different in Case (b). For instance, we can apply $q1.3$ to string $\alpha q q' q' b b \beta$, followed by $q4.1$. Again, we have two configurations to study: (A) $(\alpha q' Y b b \beta)_1$ or (B) $(\alpha Y q' b b \beta)_1$. In Case (A), we can argue similarly to Case (a) above to see that this configuration cannot lead to a terminal string: left to q' will sit some symbol A or C. Case (B) is indeed different. Assuming that only rules of the form $Z \to b'Z'$ are simulated subsequently, there may be a derivation $Y \Rightarrow^* \gamma X$ with $\gamma \in \{A,C\}^+$ that is simulated by the GCID system as intended. Hence, we see now a configuration $(\alpha \gamma X q' b b \beta)_1$ and then, after a short excursion into $C2$, we see $(\alpha \gamma q q' b b \beta)_1$. Now, we can actually terminate by using the rules $q1.3$ and $q4.1$, leading to $(\alpha \gamma Y b b \beta)_1$. However, we would arrive at the same string if we had followed our intended plan. Then, we could get from $(\alpha Y b \beta)_1$ via $(\alpha \gamma X b \beta)_1$ to $(\alpha \gamma q b \beta)_1$. Now, after applying $q1.2$ and $q2.2$ as intended, we can also see $(\alpha \gamma q q' b b \beta)_1$ and continue as above. This argument is also valid (by a separate yet straightforward induction) if we happen to produce a string $(\alpha q (q')^k b^k \beta)_1$ for an arbitrary $k > 1$. Therefore, we can avoid this process that we call *rule inversion*, and always follow our standard derivation instead. We can hence assume that we apply $q1.3$ to w^5 as desired. Therefore, we arrive at the configuration $(w^5)_1 = (\alpha q q' Y b \beta)_4$. If we actually apply $q4.1$, then we arrive at $(\alpha Y b \beta)_1$ as intended, proving the inductive step in this case. But by the very structure of this component, no other rules are applicable.

Applying $p1.1$ to $w^1 = \alpha X \beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$ and $X \in N'$, the only nonterminal from N', will insert a marker p to the right of X, yielding $(w^2)_2 = (\alpha X p \beta)_2$. Recall that we are trying to simulate the rule $X \to bY$ for some $X, Y \in N'$ and $b \in \{A,C\}$. In $C2$, all rules (except $p2.1$ and $q2.1$) are guarded by markers and the only applicable rule are $q2.1$ or $p2.1$ (in case there is a rule $X \to b'Y'$ in P) which delete the nonterminal X, yielding a string $w^3 = \alpha p \beta$, i.e., we arrive at the configuration $(w^3)_1$. Since X is deleted in the previous step and there is no p', p'', the only applicable rule is $p1.2$ which inserts a p' after p, yielding the configuration $(w^4)_3 = (\alpha p p' \beta)_3$. In $C3$, guarded by rule markers, we have to apply $p3.1$ as intended. Hence, we arrive at $(w^5)_1 = (\alpha p p' p^v \beta)_1$. If we re-apply $p1.2$, we achieve

an imbalance of the number of occurrences of p and p'. This is problematic insofar, as pp' is deleted together. Also, we would have to then re-apply $p3.1$ again, creating another imbalance. Alternatively, we could apply $f1.1$ or $g1.1$, which introduces a pair of non-primed and primed f/g-markers prematurely, but maintaining their balance. This brings us to the conclusion that we should apply $p1.3$ in $(w^5)_1$.

Hence, we arrive at $(w^6)_5 = (\alpha pp'bp^v\beta)_5$, with $b \in \{A,C\}$. In $C5$, we have to apply a rule that puts some marker r'' to the right of an occurrence of b. As $b \in \{A,C\}$, the b occurring between p' and p^v is the rightmost of all occurrences of A,C within w^6. Let us first discuss what happens if we do apply some $r5.1$ (but possibly $r \neq p$) to this described rightmost occurrence and mark the situation when some b within α is affected as $(*)$, not to forget its discussion. We get to $(w^7)_1 = (\alpha pp'br''p^v\beta)_1$. Apart from now applying $f1.1$ or $g1.1$, we could also apply $p1.2$ or $p1.3$, and finally also $p1.4$ (as intended). The scenario of prematurely introducing the f/g-markers has been sufficiently discussed before. If we apply rule $p1.2$, we again create an imbalance concerning p/p'. Let us defer the discussion of applying $p1.3$ at this configuration $(w^7)_1$ a bit as $(+)$; we rather discuss applying $p1.4$. We arrive at the configuration $(w^8)_4 = (\alpha br''p^v\beta)_4$. Now, only $r4.1$ would be applicable, leading to $(w^9)_1 = (\alpha br''r'''p^v\beta)_1$. We are now in safer waters, as we have to use rule $r1.5$ to get to $(w^{10})_4 = (\alpha br''r'''r^{iv}p^v\beta)_4$, because if we apply $p1.6$ directly, we have no chance to delete p^v in the future. If we apply $r4.1$ again on w^{10}, we create an imbalance between the number of r'' and r''', but this balance is necessary for deleting these markers in $C2$. By using $r4.2$ alternatively on w^{10}, one can see that the only chance to continue the route is when we have $r = p$. In that case, we move to $C1$ with $w^{11} = \alpha bp''p'''\beta$. If we now re-apply $p1.5$, at $C4$, we have to apply $p4.1$ and create an imbalance between p'' and p''', hence preventing us from a terminating derivation. If we introduce f- or g-markers, we are forced to introduce primed versions in $C6$; we have discussed these premature but balanced introductions of these markers before. Hence, we have to discuss applying $p1.6$ as intended. We enter $C2$ with $w^{12} = \alpha bp''p'''Y\beta$. Now, we can either delete Y with some fitting rule $s2.1$ and return to the configuration $(w^{11})_1$, hence making no progress, or we apply $p2.2$ as intended, finally getting to the configuration $(w^{12})_1 = (\alpha bY\beta)_1$ as desired.

In order to conclude that the induction step has been shown, we still have to consider two scenarios, marked as $(*)$ and $(+)$ above. In $(+)$, we look at $(w^7)_1 = (\alpha pp'br''p^v\beta)_1 \Rightarrow_{p1.3} (\alpha pp'bbr''p^v\beta)_5$. Assume we apply a rule $s5.1$ next. As the case when we find $b \in \{A,C\}$ within α is similar to the discussion $(*)$ that is still to come, we focus on two cases of configurations: (1) $(\alpha pp'bs''br''p^v\beta)_1$ or (2) $(\alpha pp'bbs''r''p^v\beta)_1$. In both configurations, we can again apply $p1.3$, but this makes the whole case fail even more. We can now derive (under the conclusion that $r = p$) in the same way as in the main line of derivation, leading to $(\alpha bs''bY\beta)_1$ (Case (1)) or to $(\alpha bbs''Y\beta)_1$ (Case (2)). In both cases, there is no way to make use of s'', because this means we have to move to $C4$, or we mis-use another p-type rule at some point, when $p1.4$ makes us enter $C4$ again, but then continuing with the s-markers (using $s4.1$). Let us clarify this by assuming that we simulate $t : Y \to b'Y'$ next. Following the standard simulation up to $t1.4$, we get $(\alpha bs''bb't''t^v\beta)_4$ (Case (1)) or $(\alpha bbs''b't''t^v\beta)_4$ (Case (2)). We could now use $s4.1$, $s1.6$ and $s2.2$ to introduce another nonterminal from N' at the position of s'', but behold: we have now another left-over double-primed marker t'' whose removal can only be achieved by switching between two rule simulations in the 'next round'. Therefore, we will never be able to terminate this derivation.

For scenario $(*)$, we reconsider $(w^6)_5 = (\alpha pp'bp^v\beta)_5$, with $\alpha = \alpha_1 b\alpha_2$, so that for a suitable rule r that should introduce $Z \in N'$, $(w^7)_1 = (\alpha_1 br''\alpha_2 pp'bp^v\beta)_1$. We could try to continue with $(w^7)_1 \Rightarrow_{p1.4}$

$$(\alpha_1 br''\alpha_2 bp^v\beta)_4 \Rightarrow_{r4.1} (\alpha_1 br''r'''\alpha_2 bp^v\beta)_1 \Rightarrow_{r1.6} (\alpha_1 br''r'''Z\alpha_2 bp^v\beta)_2 \Rightarrow_{r2.2} (\alpha_1 bZ\alpha_2 bp^v\beta)_1$$

but then there is never a chance to lose p^v again. Therefore, also this scenario will never see a derivation producing a terminal string.

Component C1	Component C2	Component C3	Component C4	Component C5
$p1.1 : (1,(X,p,\lambda)_I,2)$	$p2.1 : (2,(\lambda,X,\lambda)_D,1)$	$p3.1 : (3,(\lambda,p'',\lambda)_D,1)$	$p4.1 : (4,(p,p',\lambda)_I,1)$	$p5.1 : (5,(\lambda,pp',\lambda)_D,1)$
$p1.2 : (1,(p,p'',\lambda)_I,4)$				
$p1.3 : (1,(p',b,\lambda)_I,5)$				
$p1.4 : (1,(p'',Y,\lambda)_I,3)$				

Table 4: A direct simulation attempt for a p-rule $p : X \to bY$.

We are now discussing a string w derivable in G and as configuration $(w)_1$ in Π, with $w = \alpha AB\beta$, with $\alpha \in \{A,C\}^*$ and $\beta \in (\{B,D\} \cup T)^*$. The case of a string of the form $\alpha CD\beta$ can be discussed in a very similar fashion. First observe that we cannot apply any rule $p1.x$ or $q1.x$ or $h1.x$ due to the absence of nonterminals from N' or of required markers. We could in fact start with $g1.1$, followed by $g6.1$, and even repeat this, so that some g-markers are attached to C-occurrences. Similarly, we can consider such derivations to occur prematurely, because finally we have to use the f-rule as explained next.

Applying $f1.1$ to $w = \alpha AB\beta$, we get a string w_1 by inserting f anywhere after an A-occurrence within w. Let $\alpha A = \alpha_1 A \alpha_2$ indicate this position, i.e., $w_1 = \alpha_1 A f \alpha_2 B\beta$. w_1 is transferred to component $C6$. So, $f6.1$ is applied and the string, yielding $w_2 = \alpha_1 A f \alpha_2 \beta_1 B f' \beta_2$, which enters $C1$, where $B\beta = \beta_1 B \beta_2$. Notice that the configuration $(w_2)_1$ could have also been created by a premature application of $f1.1$ and $f6.1$ in some earlier phase of the derivation. This explains how a string that satisfies the balance condition could finally yield a terminal string, although it is not following the standard simulation as described in the beginning of the proof. As α_2 cannot contain any B-occurrence, now applying $f1.2$ necessitates $\alpha_2 = \beta_1 = \lambda$, and then, $w_3 = \alpha_1 A f' \beta_2$ is sent to $C2$. There, the only applicable rule is $f2.1$ as intended, producing $w_4 = \alpha_1 \beta_2$, sent to $C1$ as intended. As mentioned at several places, instead of applying $f1.2$ on w_2, one could also possibly apply $f1.1$ again, or also $g1.1$. We can consider all these attempts as premature ones, they only affect the left part of the string and have to be finally successfully matched by using rules $f1.2$ or $g1.2$, followed by executing another deletion rule in $C2$.

It could be that a string $w = \alpha\beta$ was derived in G (and hence possibly the configuration $(w)_1$ in Π by induction) with $\alpha \in \{A,C\}^*$ and $\beta \in (\{B,D\} \cup T)^*$ and neither α ends with A and β starts with B nor α ends with C and β starts with D. We can still apply rules $f1.1$ or $g1.1$, moving the resultant string to $C6$, where $f6.1$ or $g6.1$ are applicable, moving us back to $C1$. Yet, the crucial observation is that neither $f1.2$ nor $g1.2$ are ever applicable now, as they require the presence of the substring AB or CD (within w), respectively. Only then, the substrings fB or gD can be created.

This concludes our argument concerning the inductive step of the correctness proof of our suggested simulation.

Finally, Proposition 8 shows that star-controlled GCID systems of size $(6;1,0,1;2,0,0)$ are computationally complete, as well. □

Remark 10 *The reader might wonder if it would be possible to merge some of the components of the previous construction (Theorem 9), but this will create malicious derivations in each case. Also, the simulation of a p-rule cannot follow a simple pattern as that of the q-rule (see Rem. 10), as we want to avoid the derivation of strings with more than one occurrence of a symbol from N'. Here, we explain why a simple, not complex looking and seeming correct $p : X \to bY$ rule simulation does not work with the size $(5;1,1,0;2,0,0)$. Consider, if we attempt to construct a p-rule simulation for Π as in Table 4.*

A sample derivation of p-rule with the size $(5;1,1,0;2,0,0)$ is as follows.

$$(\alpha X\beta)_1 \Rightarrow_{p1.1} (\alpha Xp\beta)_2 \Rightarrow_{p2.1} (\alpha p\beta)_1 \Rightarrow_{p1.2} (\alpha pp''\beta)_4 \Rightarrow_{p4.1} (\alpha pp'p''\beta)_1 \Rightarrow_{p1.3} (\alpha pp'bp''\beta)_5$$
$$\Rightarrow_{p5.1} (\alpha bp''\beta)_1 \Rightarrow_{p4} (\alpha bp''Y\beta)_3 \Rightarrow_{p1.4} (\alpha bY\beta)_1.$$

*However, the simulation does not always work in the intended way as one need not apply p1.3 and instead p*1.4 *can be applied first. Therefore, the corresponding b is not inserted, however the Y has been inserted. With suitable a Y-rule that (finally) creates X again, later one could eliminate the markers pp' together and that will be a problem as a malicious string could be generated. For example, consider the grammar G contains the rules $p : X \to bY$ and $u : Y \to b'X$, $b,b' \in \{A,C\}$, $b \neq b'$, besides some other rules. Then, with the rules of Table 4, we can have the following derivation.*

$$(\alpha X\beta)_1 \Rightarrow_{p1.1,p2.1,p1.2,p4.1,p1.4,p3.1} (\alpha pp'Y\beta)_1 \Rightarrow^{*\,simulating}_{Y\to b'X} (\alpha pp'b'X\beta)_1 \Rightarrow^{*}_{as\ earlier\ for\ X} (\alpha pp'b'pp'Y\beta)_1$$
$$\Rightarrow_{p1.3} (\alpha pp'b'pp'bY\beta)_5 \Rightarrow_{p5.1} (\alpha b'pp'bY\beta)_1 \Rightarrow_{p1.3} (\alpha b'pp'bbY\beta)_5 \Rightarrow_{p5.1} (\alpha b'bbY\beta)_1.$$

We are supposed to get $\alpha bb'bY\beta$ with G, but we could derive $\alpha b'bbY\beta$ with Π, but not in G.

Our next computational completeness result even further reduces the deletion complexity, making it context-free.

Theorem 11 $\mathrm{RE} = \mathrm{GCID}_S(4;2,1,1;1,0,0)$.

Component C1	Component C2	Component C3	Component C4
$p1.1 : (1,(\lambda,p,X)_I,2)$ $p1.2 : (1,(p,bY,\lambda)_I,3)$	$p2.1 : (2,(\lambda,X,\lambda)_D,1)$	$p3.1 : (2,(\lambda,p,\lambda)_D,1)$	
$q1.1 : (1,(\lambda,q,X)_I,2)$ $q1.2 : (1,(q,Yb,\lambda)_I,3)$	$q2.1 : (2,(\lambda,X,\lambda)_D,1)$	$q3.1 : (2,(\lambda,q,\lambda)_D,1)$	
$h1.1 : (1,(\lambda,hh',S')_I,2)$ $h1.2 : (1,(\lambda,h',\lambda)_D,3)$	$h2.1 : (3,(\lambda,S',\lambda)_D,1)$	$h3.1 : (3,(\lambda,h,\lambda)_D,1)$	
$f1.1 : (1,(\lambda,f',A)_I,2)$ $f1.2 : (1,(\lambda,A,\lambda)_D,4)$ $f1.3 : (1,(\lambda,f,\lambda)_D,2)$ $f1.4 : (1,(\lambda,B,\lambda)_D,4)$ $f1.5 : (1,(\lambda,f^3,\lambda)_D,3)$ $f1.6 : (1,(\lambda,f''',\lambda)_D,3)$ $f1.7 : (1,(\lambda,f^4,\lambda)_D,2)$	$f2.1 : (2,(A,f,B)_I,1)$ $f2.2 : (2,(B,f^4,\lambda)_I,1)$ $f2.3 : (\lambda,f^2,\lambda)_D,1)$	$f3.1 : (3,(3,(\lambda,f'',\lambda)_D,1)$ $f3.2 : (3,(\lambda,f',\lambda)_D,1)$	$f4.1 : (4,(f',f''f^2,f)_I,1)$ $f4.2 : (4,(f^2,f'''f^3,f^4)_I,1)$

Table 5: Star-controlled GCID of size $(4;2,1,1;1,0,0)$ simulating rules of SGNF.

Proof Consider a type-0 grammar $G = (N,T,P,S)$ in SGNF as in Definition 1. We construct a GCID system $\Pi = (4,V,T,\{S\},H,1,\{1\},R)$ of size $(4;2,1,1;1,0,0)$ such that $L(\Pi) = L(G)$. The set V contains the symbols of G as well as some rule markers. We refer to Table 5 for the direct simulation of SGNF. The rules simulating $g : CD \to \lambda$ are similar to the ones simulating the $f : AB \to \lambda$ rule and hence omitted.

We now prove that $L(G) \subseteq L(\Pi)$. We show that if $w \Rightarrow w'$ in G, then $(w)_1 \Rightarrow_* (w')_1$ according to Π. From this fact, the claim follows by a simple induction, split into different cases, as discussed now.

Context-free rule $p : X \to bY$. Here, $w = \alpha X\beta$ and $w' = \alpha bY\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$. The simulation performs as follows:

$$(w)_1 = (\alpha X\beta)_1 \Rightarrow_{p1.1} (\alpha pX\beta)_2 \Rightarrow_{p2.1} (\alpha p\beta)_1 \Rightarrow_{p1.2} (\alpha pbY\beta)_3 \Rightarrow_{p3.1} (\alpha bY\beta)_1 = w'.$$

Context-free rule $q : X \to Yb$ is simulated in a similar fashion to the simulation of the p-rule.

Context-free rule $h : S' \to \lambda$ is simulated as follows for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$:

$$(w)_1 = (\alpha S'\beta)_1 \Rightarrow_{h1.1} (\alpha hh'S'\beta)_2 \Rightarrow_{h2.1} (\alpha hh'\beta)_1 \Rightarrow_{h1.2} (\alpha h\beta)_3 \Rightarrow_{h3.1} (\alpha\beta)_1 = w'.$$

Non-context-free rule $f : AB \to \lambda$. This means that we expect $w = \alpha AB\beta$ and $w' = \alpha\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$. Within Π, this can be simulated as follows.

$$\begin{aligned}(\alpha AB\beta)_1 &\Rightarrow_{f1.1} (\alpha f' AB\beta)_2 \Rightarrow_{f2.1} (\alpha f' AfB\beta)_1 \Rightarrow_{f1.2} (\alpha f' fB\beta)_4 \Rightarrow_{f4.1} (\alpha f' f'' f^2 fB\beta)_1 \Rightarrow_{f1.3} (\alpha f' f'' f^2 B\beta)_2 \\ &\Rightarrow_{f2.2} (\alpha f' f'' f^2 Bf^4\beta)_1 \Rightarrow_{f1.4} (\alpha f' f'' f^2 f^4\beta)_4 \Rightarrow_{f4.2} (\alpha f' f'' f^2 f''' f^3 f^4\beta)_1 \Rightarrow_{f1.5} (\alpha f' f'' f^2 f''' f^4\beta)_3 \\ &\Rightarrow_{f3.1} (\alpha f' f^2 f''' f^4\beta)_1 \Rightarrow_{f1.6} (\alpha f' f^2 f^4\beta)_3 \Rightarrow_{f3.2} (\alpha f^2 f^4\beta)_1 \Rightarrow_{f1.7} (\alpha f^2\beta)_2 \Rightarrow_{f2.3} (\alpha\beta)_1.\end{aligned}$$

We next prove that $L(\Pi) \subseteq L(G)$. More precisely, we show the following (by induction). If $(u)_1$ is a configuration that is derivable in Π such that u contains the same number of markers from $\{f', g'\}$ as from $\{f^4, g^4\}$ (we call this property of the symbols from $\{f', g', f^4, g^4\}$ also a balanced situation) and such that the word u' that is obtained from u by deleting all symbols from $\{f', g', f^4, g^4\}$ belongs to $(N \cup T)^*$, then u is derivable in G. In particular, by induction we can assume that any such u that we discussed for proving the inductive step satisfies $u' \in \{A,C\}^*(N' \cup \{\lambda\})(\{B,D\} \cup T)^*$. To avoid clumsy formulations, we will discuss the markers from $\{f', g', f^4, g^4\}$ only in particular situations and argue why we maintain the property of being in a balanced situation. Hence, consider a derivation $(w)_1 \Rightarrow^* (w')_1$, with $w \neq w'$ and $w, w' \in \{A,C\}^*(N' \cup \{\lambda\})(\{B,D\} \cup T)^*$; it has to start like $(w)_1 \Rightarrow (v)_j$ in Π. If some rule from $C1$ is applied to w, the rule will insert a rule marker into the string w and move to $C2$, or an A or B is deleted and the string moves to $C4$. The introduction of rule markers in $C1$ will take care of the non-interference among the non-context-free and context-free rules. We will now discuss more details.

Let us first assume (by induction) that the sentential form $w^1 = \alpha X\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$ and $X \in N'$ is derivable in G and the configuration $(w^1)_1$ is derivable in Π. We prove (as induction step) that if $(w^1)_1 \Rightarrow_* (u)_1$, $w^1 \neq u$, and $u \in \{A,C\}^*(N' \cup \{\lambda\})(\{B,D\} \cup T)^*$ is the first sentential form in $\{A,C\}^*(N' \cup \{\lambda\})(\{B,D\} \cup T)^*$ that appears in a Π-derivation, then $w^1 \Rightarrow u$ in G.

Caveat: The reader might wonder why a context-free deletion of A or B as in rules $f1.2$ or $f1.4$ could work at all. But notice that in $C4$, which is the target component of these rules, the presence of f-style markers is checked in each of the rules. This prevents any successful derivation that interferes with, say, a p-rule derivation by deleting an A or a B in the first component, simply because there are no p-rule simulation rules in $C4$. We will hence tacitly assume that $f1.2$ etc. are not applied within p-simulations.

Applying $p1.1$ to $w^1 = \alpha X\beta$ for some $\alpha \in \{A,C\}^*$, $\beta \in (\{B,D\} \cup T)^*$ and as $X \in N'$ being the only nonterminal of N' that is in w^1, the marker p will be inserted to the left of X, yielding $(w^2)_2 = (\alpha pX\beta)_2$. In $C2$, the only available and applicable rule among the rules meant to simulate context-free rules deletes the nonterminal X (which is not S') of N'. This results in $(w^3)_1 = (\alpha p\beta)_1$. Alternatively, rule $f2.2$ (or $g2.2$) would be applicable in $C2$, bringing the string back to $C1$. There, one could return to $C2$ by using $f1.1$, $g1.1$, $f1.7$ or $g1.7$. The last two possibilities (if applicable at all) would just undo the previous step and hence offer no progress in the derivation, or they would exchange an occurrence of g^4 with an occurrence of f^4 or vice versa. The first two mentioned rules would add a symbol from $\{f', g'\}$ to the string obtained from w^3 by previously adding a symbol from $\{f^4, g^4\}$. Hence, if we think of the statement that our inductive argument should prove, we keep up a balanced situation as required. Now, in $C1$, the rule $p1.1$ cannot be applied again, as no $X \in N'$ is present. So, the only applicable rule (within the rules in $C1$ that are meant to deal with simulating context-free rules) is $p1.2$ which results in $(w^3)_3 = (\alpha pbY\beta)_3$. In $C3$, the introduced marker p is deleted and the resultant string $w^4 = \alpha bY\beta$ is sent to $C1$, thus the intended and desired derivation is correctly simulated. Notice that we could also apply $f3.2$ or $g3.2$ instead of $p3.1$, which would delete some f' or g' that might have been introduced earlier. However, this would then create an imbalanced situation, with less occurrences of symbols from $\{f', g'\}$ than from $\{f^4, g^4\}$ within a string in $C1$. As we will see, such an imbalance can never be resolved, so

that such a string will not yield a terminal string. There is one possibility after applying $p1.1$ and $p2.1$ that still needs to be discussed. It might be possible to apply $f1.1$ on $w^3 = \alpha p \beta$. This is only possible if $\alpha = \alpha_1 A \alpha_2$, so that we arrive at $(u^4)_2 = (\alpha_1 f' A \alpha_2 p \beta)_2$. Now, $f2.2$ or $g2.2$ may be applicable, bringing us back to $C1$. However, this will maintain a balanced situation as claimed. Moreover, as the reader can check, this is indeed the only possible continuation, keeping in mind that AB will not occur as a substring in the present string due to SGNF. So, in various ways, along with a simulation of a context-free rule, we may add symbols from $\{f', g', f^4, g^4\}$, but this always happens in a balanced way if it might be fruitful.

The correctness of the simulation of the h-rule is easily seen. Notice that there are again possibilities to introduce or delete symbols from $\{f', g', f^4, g^4\}$ similar as discussed above for simulating p-rules.

Applying $f1.1$ to $w^1 = \alpha X \beta$ for some $\alpha \in \{A, C\}^*$, $\beta \in (\{B, D\} \cup T)^*$ and $X \in N'$, the marker f' will be inserted to the left of some occurrence of A, yielding $(w^2)_2 = (\alpha_1 f' A \alpha_2 X \beta)_2$, with $\alpha = \alpha_1 A \alpha_2$. As $w^1 = \alpha X \beta$ indicates that we are simulating phase I of the work of the SGNF grammar G, the substring AB is absent, preventing us from applying $f2.1$. We might continue with $f2.2$ or $g2.2$, though, which introduces an occurrence of f^4 or g^4, respectively. This is one possibility how we can obtain 'pairs' of occurrences of symbols from $\{f', g'\}$ and $\{f^4, g^4\}$, but we clearly maintain a balanced situation. Still, we might delete X with $p2.1$, say, but then we arrive at a typical imbalanced situation. We would have to apply $f1.1$ or $f1.2$ or $f1.4$ next, as no markers from the context-free rule simulations are present. In the first case, we see that we maintain an imbalanced situation of which we can never get rid, while in the second and third case, the derivation is blocked in $C4$ because of the lack of appropriate f-markers.

We could lead a similar discussion for applying $f1.7$ to a string that contains one occurrence of N'-symbols and at least one occurrence of f^4; in particular, the arguments concerning (im)balance remain the same, because for this condition, it does not matter whether we add f' or delete f^4.

We shall discuss the cases for the f-rule simulation next, including that the rules $f1.i$ were applied in a wrong manner. Recall the discussion of the *Caveat* above which ruled out a premature application of $f1.2$ or of $f1.4$. We cannot start with any rules that require the presence of f-markers, apart from those stemming from a balanced situation that we will discuss below. We therefore discuss a derivation starting with $f1.1$ on w^1 as intended. Again by the absence of the marker f'', we have to apply $f2.1$ next, shifting the discussion of a balanced situation as created after applying $f2.2$ to what we say further down. The role of rule $f2.1$ is crucial insofar as it checks that the substring AB is present in the current string. It is one of the important properties of SGNF that this substring can only occur once in a derived string, and this also means that we are in phase II of the SGNF derivation. By induction, we can assume this property also to hold for the string w^1 that is under discussion. In other words, we can assume that $w^1 = \alpha AB \beta$. Then, after applying $f1.1$ and $f2.1$, we are in the configuration $(w^2)_1 = (\alpha' f' \alpha'' A f B \beta)_1$ with $\alpha = \alpha' \alpha''$, and α'' being a string that is either empty or it starts with an A. The intention would be to apply $f1.2$ on w^2. As the marker f^2 is absent, we have to apply $f4.1$ now. This is only possible if α'' is empty. Hence, $(w^2)_1 = (\alpha f' A f B \beta)_1$, and after applying $f1.2$ and $f4.1$, we necessarily arrive at the configuration $(w^3)_1 = (\alpha f' f'' f^2 f B \beta)_1$. We now check the other possibilities in configuration $(w^2)_1$. Due to the absence of f^3, f''' and f^4, only $f1.1$, $f1.3$, or $f1.4$ are applicable. If we apply $f1.1$, then as there is no second substring AB nor the f-marker f^2 present, we have to introduce f^4 next to obtain a configuration $(u)_1$ where u contains two occurrences of f' and one occurrence of f and one of f^4. So, on $(u)_1$, we might apply $f1.2$, followed by $f4.1$. As argued above for the main line of derivation, this means that we arrive at a configuration $(v)_1$ with $v = \alpha_1 f' \alpha_2 f' f'' f^2 f \beta_1 B f^4 \beta_2$, with $\alpha_1 \alpha_2 = \alpha$ and $\beta_1 B \beta_2 = \beta$. In fact, we could continue now with the derivation, closely following the main line, the only difference being an additional f' and f^4 being present in the string. This indicates that balanced situations are not necessarily a problem and also shows how they can arise. On applying $f1.3$, we can delete f, but this

takes us back to $(w^2)_2$. This analysis tells us that, in configuration $(w^2)_1$, we have to apply $f1.2$.

Now, we study the configuration $(w^3)_1 = (\alpha f' f'' f^2 f B \beta)_1$. If we apply $f1.1$, we again have to introduce an occurrence of f^4 and can then follow the main line of derivation, which will finally lead to a balanced situation. If we delete A or B, the derivation is stuck in $C4$. Hence, the only applicable, promising rule is $f1.3$, deleting f and moving to $C2$. After deleting f^2, we get back to $C1$ with nearly the same (im)possibilities as just discussed, except that $f1.3$ is no longer available. Therefore, such a derivation cannot lead to a terminal string. Hence, in $C2$ the rule $f2.2$ must be applied, giving, with $\beta'\beta'' = \beta$ and β' being empty or ending with B, $(w^3)_1 = (\alpha f' f'' f^2 f B \beta' f^4 \beta'')_1$. Again, we might add a second f' with no fruitful continuation now, except for possibly creating other balanced situations finally. If we delete A, then the derivation is stuck in $C4$. If we delete f^4 with $f1.7$, we have to delete f^2 in $C2$ (unless we want to un-do $f1.7$ by applying $f2.2$) and then, whatever we apply next ($f1.1$ or $f1.2$ or $f1.4$), the derivation is stuck, ignoring the possibility to add more and more occurrences of f' and f^4. Therefore, on w^3, we have to apply $f1.4$ which deletes one B. In $C4$, we can only apply any rule if β' is empty, so that $f4.2$ is applicable, resulting in $(w^4)_1 = (\alpha f' f'' f^2 f''' f^3 f^4 \beta)_1$. The next six rule applications will delete all six f-markers; several ways to do this are possible. What could go wrong? Introducing a second f'-occurrence is again not interesting: it has to be matched by adding a f^4-occurrence; if we delete f^2 instead, then the addition of f^4 is indirect insofar, as $f1.7$ need not be executed to delete f^2. After applying $f1.5$, $f3.1$, $f1.6$ and $f3.2$, we end up with a string from which one could delete any A- or B-occurrence and try to restart with $f4.2$. Yet, now the symbols f''' and f^3 can only be deleted if one introduces two additional occurrences of $\{f', f''\}$, requiring a complete re-start of the simulation; it is not possible to get rid of f''' and f^3 otherwise. Such a re-start would require the substring AB which is currently not present. Hence, $(\alpha f^2 f^4 \beta)_1$ can only be continued as intended, applying $f1.7$ and $f2.3$.

More on balanced situations. As we have been mentioning these over and over again in the previous arguments, let us briefly discuss possibilities when we do have some balanced occurrences from $\{f', g', f^4, g^4\}$ in configuration $(w^1)_1$, e.g., $w^1 = \alpha AB\beta$ where $\alpha \in \{A, C\}^*\{f'\}$ and β contains exactly one occurrence from $\{f^4, g^4\}$. Then, instead of applying $f1.1$, we could start the derivation with $f1.7$ or $g1.7$. Notice that this yields a configuration $(w^2)_2$ that could have also been obtained when starting from $(\alpha' AB\beta')_1$ and applying $f1.1$, where α' is obtained from α by deleting f' and β' is obtained from β by deleting the unique occurrence of either f^4 or g^4. This shows that balanced situations can lead to terminal strings finally, as they may converge again to the main line of derivation. Importantly, no new terminal strings can be derived that are not following the possibilities given by G.

This concludes the main arguments concerning the inductive step and hence the claim follows. □

4 Summary and Open Problems

In this paper, we focused on examining the computational power of graph-controlled ins-del systems with a star as a control graph. We lowered the resource requirements to describe RE, all recursively enumerable languages. We leave it open to explore the following possibilities.

1. $\mathrm{GCID}_S(k; 2, i', i''; 1, j', j'') \stackrel{?}{=} \mathrm{RE}$ for $i' + i'' \leq 1$ and $j' + j'' \leq 1$ and some k as small as possible,
2. $\mathrm{GCID}_S(k'; 2, 0, 0; 1, i', i'') \stackrel{?}{=} \mathrm{RE}$ for $i' + i'' \leq 1$ and some k' as small as possible.

Here we only considered GCID systems where the underlying graph is star-controlled and does not contain loops. One may also consider a tree structure and / or the possibility to allow loops (i.e., rules have option 'here' and the resultant string can stay back in the same component if such rules are applied), which may give additional power and connect closer to ins-del P systems and also to the results of [9].

References

[1] A. Alhazov, R. Freund, S. Ivanov & S. Verlan (2022): *Regulated Insertion-Deletion Systems*. Journal of Automata, Languages and Combinatorics 27(1-3), pp. 15–45, doi:10.25596/jalc-2022-015.

[2] R. Benne, editor (1993): *RNA Editing: The Alteration of Protein Coding Sequences of RNA*. Ellis Horwood.

[3] H. Fernau, L. Kuppusamy & I. Raman (2017): *On the computational completeness of graph-controlled insertion-deletion systems with binary sizes*. Theoretical Computer Science 682, pp. 100–121, doi:10.1016/j.tcs.2017.01.019.

[4] H. Fernau, L. Kuppusamy & I. Raman (2019): *On path-controlled insertion-deletion systems*. Acta Informatica 56(1), pp. 35–59, doi:10.1007/s00236-018-0312-2.

[5] H. Fernau, L. Kuppusamy & I. Raman (2021): *On the generative capacity of matrix insertion-deletion systems of small sum-norm*. Natural Computing 20(4), pp. 671–689, doi:10.1007/s11047-021-09866-y.

[6] R. Freund, M. Kogler, Y. Rogozhin & S. Verlan (2010): *Graph-Controlled Insertion-Deletion Systems*. In I. McQuillan & G. Pighizzini, editors: Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, *DCFS*, *EPTCS* 31, pp. 88–98, doi:10.4204/EPTCS.31.11.

[7] V. Geffert (1991): *Normal forms for phrase-structure grammars*. RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications 25, pp. 473–498, doi:10.1051/ita/1991250504731.

[8] S. Ivanov & S. Verlan (2014): *About One-Sided One-Symbol Insertion-Deletion P Systems*. In A. Alhazov, S. Cojocaru, M. Gheorghe, Y. Rogozhin, G. Rozenberg & A. Salomaa, editors: Membrane Computing - 14th Int. Conf., *CMC 2013*, *LNCS* 8340, Springer, pp. 225–237, doi:10.1007/978-3-642-54239-8_16.

[9] S. Ivanov & S. Verlan (2017): *Universality and Computational Completeness of Controlled Leftist Insertion-Deletion Systems*. Fundamenta Informaticae 155(1-2), pp. 163–185, doi:10.3233/FI-2017-1580.

[10] L. Kari, Gh. Păun, G. Thierrin & S. Yu (1999): *At the crossroads of DNA computing and formal languages: Characterizing recursively enumerable languages using insertion-deletion systems*. In: Discrete Mathematics and Theretical Computer Science, *DIMACS* 48, AMS, pp. 329–338, doi:10.1090/dimacs/048/23.

[11] L. Kari & G. Thierrin (1996): *Contextual Insertions/Deletions and Computability*. Information and Computation 131(1), pp. 47–61, doi:10.1006/inco.1996.0091.

[12] A. Krassovitskiy, Y. Rogozhin & S. Verlan (2008): *Further Results on Insertion-Deletion Systems with One-Sided Contexts*. In C. Martín-Vide, F. Otto & H. Fernau, editors: Language & Automata Theory & Applications, *LATA*, *LNCS* 5196, Springer, pp. 333–344, doi:10.1007/978-3-540-88282-4_31.

[13] S. N. Krishna & R. Rama (2002): *Insertion-Deletion P Systems*. In N. Jonoska & N. C. Seeman, editors: DNA Computing, 7th Int. Workshop on DNA-Based Computers, *2001*, *LNCS* 2340, Springer, pp. 360–370, doi:10.1007/3-540-48017-X_34.

[14] S. Marcus (1969): *Contextual grammars*. Revue Roumaine de Mathématiques Pures et Appliquées 14, pp. 1525–1534.

[15] A. Matveevici, Y. Rogozhin & S. Verlan (2007): *Insertion-Deletion Systems with One-Sided Contexts*. In: *MCU*, *LNCS* 4664, Springer, pp. 205–217, doi:10.1007/978-3-540-74593-8_18.

[16] Gh. Păun (1997): *Marcus Contextual Grammars*. Studies in Linguistics and Philosophy 67, Kluwer, doi:10.1007/978-94-015-8969-7_4.

[17] Gh. Păun, G. Rozenberg & A. Salomaa (1998): *DNA Computing: New Computing Paradigms*. Springer, doi:10.1007/978-3-662-03563-4.

[18] I. Petre & S. Verlan (2012): *Matrix insertion-deletion systems*. Theoretical Computer Science 456, pp. 80–88, doi:10.1016/j.tcs.2012.07.002.

[19] A. Takahara & T. Yokomori (2003): *On the computational power of insertion-deletion systems*. Natural Computing 2(4), pp. 321–336, doi:10.1023/B:NACO.0000006769.27984.23.

[20] S. Verlan (2010): *Recent Developments on Insertion-Deletion Systems*. The Computer Science Journal of Moldova 18(2), pp. 210–245.

Comparative Transition System Semantics for Cause-Respecting Reversible Prime Event Structures

Nataliya Gribovskaya Irina Virbitskaite
A.P. Ershov Institute of Informatics Systems
the Siberian Branch of the Russian Academy of Sciences
6, Acad. Lavrentiev avenue, 630090, Novosibirsk, Russia
{natamosk,virbitskaite}@gmail.com

Reversible computing is a new paradigm that has emerged recently and extends the traditional forwards-only computing mode with the ability to execute in backwards, so that computation can run in reverse as easily as in forward. Two approaches to developing transition system (automaton-like) semantics for event structure models are distinguished in the literature. In the first case, states are considered as configurations (sets of already executed events), and transitions between states are built by starting from the initial configuration and repeatedly adding executable events. In the second approach, states are understood as residuals (model fragments that have not yet been executed), and transitions are constructed by starting from the given event structure as the initial state and deleting already executed (and conflicting) parts thereof during execution. The present paper focuses on an investigation of how the two approaches are interrelated for the model of prime event structures extended with cause-respecting reversibility. The bisimilarity of the resulting transition systems is proved, taking into account step semantics of the model under consideration.

1 Introduction

Reversible computations, extensively studied during in recent years, is an unconventional form of computations that can be performed in the forward direction as easily as in the reverse direction. Any sequence of actions executed by the system can subsequently be canceled for some reason (for example, in case of an error), which allows the system to restore previous consistent states, as if these canceled actions were not executed at all. Reversible computing is attracting interest for its applications in many fields including program analysis and debugging [20], programming abstractions for reliable systems [11, 23], modelling biochemical reactions [18], hardware design and quantum computing [12], and etc.

Despite the fact that reversing computations in concurrent/distributed systems has many promising applications, it also involves many technical and conceptual challenges. One of the most essential issues that arise concerns the techniques that should be applied when moving backwards. Several different styles of the undoing of computation have been identified recently. The most prominent of these are backtracking [26], causal reversibility [25, 26], and out-of-causal-order reversibility [26, 28], that differ in the order of executing actions in backward direction. Backtracking is generally understood as the ability to execute past actions in the exact reverse order in which they were executed. Causal reversibility in concurrent systems means that actions that cause others can only be undone after the caused actions are undone first, and that actions which are independent of each other can be reversed in an arbitrary order. Out-of-causal reversibility, a form of reversal most characteristic of biochemical systems, does not preserve causes. The interplay between reversibility and concurrency has been widely studied in various models: parallel rewriting systems [1], cellular automata [16], process calculi [11, 19], Petri nets [6, 13, 26], event structures [24, 27, 30], membrane systems [29], and etc.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 112–126, doi:10.4204/EPTCS.386.10

Event structures are a well-established model of concurrency. They were originally proposed by Winskel in his PhD dissertation [32] and were considered as an intermediate abstraction between Scott domains (i.e., a denotational model) and Petri nets (i.e., an operational model). Basically, event structures are collections of possible events, some of which are conflicting (i.e., the execution of an event forbids the execution of other events), while others are causally dependent (i.e., an event cannot be executed if it has not been preceded by other ones), and events that are neither in causal dependency nor in conflict are treated as concurrent. Events are often labelled with actions, to represent different occurrences of the same action. Prime Event Structures (written PESs) are the earliest and simplest form of event structures, where causality is a partial order and conflict between events is inherited by their causal successors. The association of transition system (automaton-like) models with event structures has proved to contribute to studying and solving various problems in the analysis and verification of concurrent systems. It is distinguished two methods of providing transition system semantics for event structures: a configuration-based and a residual-based method. In the first case (see [2, 3, 31, 15, 17, 32, 33] among others), states are understood as sets of events, called configurations, and state transitions are built by starting with the initial configuration and enlarging configurations by already executed events. In the second more 'structural' method (see [5, 9, 10, 17, 21] among others), states are understood as event structures, and transitions are built by starting with the given event structure as an initial state and removing already executed (and conflicting) parts thereof in the course of execution. In the literature, configuration-based transition systems seem to be predominantly used as the semantics of event structures, and residual-based transition systems are actively used in providing operational semantics of process calculi and in demonstrating the consistency of operational and denotational semantics. The two kinds of transition systems have occasionally been treated alongside each other (see [17] as an example), but their general relationship has not been studied for a wide range of existing models. In a seminal paper, viz. [22], bisimulations between configuration-based and residual-based transition systems have been proved to exist for prime event structures [33]. The result of [22] has been extended in [7] to more complex event structure models with asymmetric conflict. The paper [8] demonstrated that when using non-executable events, the removal operators defined in [22, 7] to obtain residuals can be tightened in such a way that isomorphisms, rather than just bisimulations, between the two types of transition systems belonging to a single event structure can be obtained, for a full spectrum of semantics (interleaving, step, pomset, multiset).

Reversible event structures extend event structures to represent reversible computational processes, capable of undoing executed actions by allowing configurations to evolve by eliminating events. In [27, 30], Phillips et. al. determined causal and out-of-causal reversible forms of prime, asymmetric and general event structures and showed the correspondence between their configurations and traditional ones when there are no reversible events. In [4], Aubert and Cristescu have provided a true concurrent semantics of a reversible extension of CCS, RCCS (without auto-concurrency, auto-conflict, or recursion), in terms of configuration structures. In [14], Graversen et. al. have developed a category of reversible bundle event structures with symmetric conflict and used the causal subcategory to model semantics of another reversible extension of CCS, CCSK. They also modified CCSK to control reversibility with a rollback primitive, and gave, by exploiting the capacity for out-of-causal reversibility, semantics of this kind of CCSK in terms of reversible bundle event structures with asymmetric conflict. Constructions associating causal reversible prime event structures to reversible occurrence nets and vice versa have been proposed within causal reversibility in [25], as well as within out-of-causal reversibility in [24].

The aim of this paper is to identify two (configuration-based and residual-based) types of transition system semantics for cause-respecting reversible prime event structures and to understand how these types relate to each other, which can assist in the construction of algebraic calculi to describe and verify

reversible concurrent processes.

This paper is structured as follows. In Section 2, we start with recalling the syntax of prime and reversible prime event structures and their (step) semantics in terms of configurations and traces. In Section 3, we define a removal operator, which is useful for constructing model residuals, and demonstrate the correctness of the operator. In Section 4, we develop two types of transition system semantics for cause-respecting reversible prime event structures and establish bisimulation results between the semantics. In Section 5, we provide some concluding remarks. The proofs of the propositions presented here can be found at www.iis.nsk.su/virb/proofs-AFL-2023.

2 Reversing in Prime Event Structures

In this section, we first recall the notion of prime event structures (PESs) [32] labeled over the set $L = \{a, b, c, \ldots\}$ of actions, and then formulate the concept of reversible prime event structures (RPESs) [27] and consider their (step) semantics and properties.

The behavior of concurrent systems is formally modelled by event structure models where units of the behavior are represented by events. There are different ways to relate events. In prime event structures (PESs), the dependency between events, called causality, is given by a partial order, and the incompatibility is determined by a conflict relation. Two events which are neither in causal dependency nor in conflict are considered independent (concurrent).

Definition 1. *A (labeled)* prime event structure (PES) *(over the set L of actions) is a tuple* $\mathscr{E} = (E, <, \sharp, l, C_0)$*, where*

- E *is a countable set of* events*;*
- $< \subseteq E \times E$ *is an irreflexive partial order (the* causality relation*) satisfying the* principle of finite causes*:* $\forall e \in E \circ \lfloor e \rfloor = \{e' \in E \mid e' < e\}$ *is finite;*
- $\sharp \subseteq E \times E$ *is an irreflexive and symmetric relation (the* conflict relation*) satisfying the* principle of hereditary conflict*:* $\forall e, e', e'' \in E \circ e < e'$ *and* $e \sharp e''$ *then* $e' \sharp e''$*;*
- $l : E \to L$ *is a* labeling function*;*
- $C_0 = \emptyset$ *is the* initial configuration[(1)]*.*

So, the PES is a simple event-based model of concurrent and nondeterministic computations where events labeled over the set L of actions are considered as atomic, indivisible and instantaneous action occurrences, some of which can only be executed after another (i.e. there is a causal dependency represented by a partial order $\leq$ between the events) and some of which might not be executed together (i.e. there is a binary conflict $\sharp$ between the events). In addition, the principle of finite causes and the principle of conflict inheritance are required.

The PES progresses by executing events, thus moving from one state to another, starting from the initial state, which is an empty set. A state called a configuration is a set of events that have occurred. A subset of events $X \subseteq E$ is *left-closed under* $<$ iff for all $e \in X$ it holds that $\lfloor e \rfloor \subseteq X$; is *conflict-free* iff for all $e, e' \in X$ it holds that $\neg(e \sharp e')$, and we denote it with $CF(X)$. A subset $C \subseteq E$ is a *configuration* of $\mathscr{E}$ iff C is finite, left-closed under $<$ and conflict-free.

Reversible prime event structures (RPESs) [27, 30] are based on a weaker form of PESs because conflict inheritance may not hold when adding reversibility to PESs. Also, in RPESs, some events are

(1) We add the initial configuration as an empty set to the classical PES definition, but this does not affect the behavior of the structure in any way, because the PES progresses by moving from one configuration to another and starting from an empty set.

categorised as reversible, and two relations are added: the reverse causality relation and the prevention relation. The first one is a dependency relation in the backward direction: to reverse an event in the current configuration there must be other events on which the event reversibly depends. The second relation, on the contrary, identifies those events whose presence in the current configuration prevents the event being reversed.

Definition 2. *A (labeled)* reversible prime event structure (RPES) *(over L) is a tuple* $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$*, where*

- E *is a countable set of* events*;*
- $\sharp \subseteq E \times E$ *is an irreflexive and symmetric relation (the* conflict relation*);*
- $< \subseteq E \times E$ *is an irreflexive partial order (the* causality relation*) satisfying:* $\lfloor e \rfloor$ *is finite and conflict-free, for every* $e \in E$*;*
- $l : E \to L$ *is a* labeling function*;*
- $F \subseteq E$ *are* reversible events *being denoted by the set* $\underline{F} = \{\underline{e} \mid e \in F\}$ *such that* $\underline{F} \cap E = \emptyset$*;*
- $\prec \subseteq E \times \underline{F}$ *is the* reverse causality relation *satisfying:* $a \prec \underline{a}$ *and* $\{e \in E \mid e \prec \underline{a}\}$ *is finite and conflict-free, for every* $a \in F$*;*
- $\rhd \subseteq E \times \underline{F}$ *is the* prevention relation *such that* $\rhd \cap \prec = \emptyset$*;*
- $\ll$ *is the transitive* sustained causation relation*:* $a \ll b$ *is defined to mean that* $a < b$ *and if* $a \in F$ *then* $b \rhd \underline{a}$*.* $\sharp$ *is* hereditary w.r.t. the sustained causation $\ll$*: if* $a \sharp b \ll c$ *then* $a \sharp c$*;*
- $C_0 \subseteq E$ *is the* initial configuration *which is finite, left-closed under* $<$ *and conflict-free.*

It is straightforward to check that any PES is also an RPES with $F = \emptyset$ and $C_0 = \emptyset$. Then, any concept defined for RPESs applies to PESs as well.

Example 1. Consider the structure $\mathscr{E}_0 = (E_0, <_0, \sharp_0, l_0, F_0, \prec_0, \rhd_0, C_0^0)$, where $E_0 = \{a,b,c,d,e\}$; $<_0 = \{(b,d),(c,e)\}$; $\sharp_0 = \{(a,b),(b,a),(b,c),(c,b)\}$; l_0 is the identical function; $F_0 = \{b,c\}$; $\prec_0 = \{(b,\underline{b}),(c,\underline{c})\}$; $\rhd_0 = \emptyset$; $C_0^0 = \emptyset$. It is easy to make sure that the components of the structure $\mathscr{E}_0$ meet the requirements of the corresponding items of Definition 2. In particular, we see that $\prec_0 = \{(b,\underline{b}),(c,\underline{c})\}$ and $(b,\underline{b}),(c,\underline{c}) \notin \rhd_0$. Notice that $\sharp_0$ is not hereditary w.r.t. $<_0$ because $a \sharp_0 b <_0 d$ and $\neg(a \sharp_0 d)$, $b \sharp_0 c <_0 e$ and $\neg(b \sharp_0 e)$, $c \sharp_0 b <_0 d$ and $\neg(c \sharp_0 d)$. From Definition 2, we know that x and y are in the sustained causation relation iff x causes y, and x cannot be reversed as long as y is present. In $\mathscr{E}_0$, the pairs (b,d) and (c,e) are in the causality relation $<_0$, and the prevention relation $\rhd_0$ is empty. Therefore, the sustained causation relation $\ll_0$ is empty. It is easy to see that $\sharp_0$ is hereditary w.r.t. $\ll_0$. So, the structure $\mathscr{E}_0$ is indeed an RPES. ◇

The RPES progresses by executing events and/or by undoing previously executed events, thus moving from one configuration to another. The act of moving is a computation step. Reachable configurations are subsets of events which can be reached from the initial configuration by executing computation steps. A sequence of computation steps is a trace of the RPES.

Definition 3. *Given an RPES* $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$*, and* $C \subseteq E$ *such that* $CF(C)$*,*

- *for* $A \subseteq E$ *and* $B \subseteq F$*, we say that* $A \cup \underline{B}$ *is* enabled at C *if*
  - *a)* $A \cap C = \emptyset$*,* $B \subseteq C$*,* $CF(C \cup A)$*;*
  - *b)* $\forall e \in A, \forall e' \in E$ *:* *if* $e' < e$ *then* $e' \in (C \setminus B)$*;*
  - *c)* $\forall e \in B, \forall e' \in E$ *:* *if* $e' \prec \underline{e}$ *then* $e' \in (C \setminus (B \setminus \{e\}))$*;*

d) $\forall e \in B, \forall e' \in E$: *if* $e' \rhd \underline{e}$ *then* $e' \notin (C \cup A)$.

If $A \cup \underline{B}$ *is enabled at* C *then* $C \xrightarrow{A \cup \underline{B}} C' = (C \setminus B) \cup A$. *We shall write* $l(A \cup \underline{B}) = M$ *iff* M *is a multiset over the set* L *of actions, defined as follows:* $M(a) = \mid \{e \in (A \cup B) \mid l(e) = a\} \mid$ *for all* $a \in L$.

- C *is a* forwards reachable configuration of $\mathscr{E}$ *(from* C_0*) iff for all* $i = 1, \ldots, n$ $(n \geq 0)$, *there exists a finite set* $A_i \subseteq E$ *such that* $C_{i-1} \xrightarrow{A_i \cup \underline{\emptyset}_i} C_i$ *and* $C_n = C$.
- C *is a* (reachable) configuration of $\mathscr{E}$ *(from* C_0*) iff for all* $i = 1, \ldots, n$ $(n \geq 0)$, *there exist finite sets* $A_i \subseteq E$ *and* $B_i \subseteq F$ *such that* $C_{i-1} \xrightarrow{A_i \cup \underline{B}_i} C_i$ *and* $C_n = C$. *In this case,* $t = (A_1 \cup \underline{B}_1) \ldots (A_n \cup \underline{B}_n)$ $(n \geq 0)$ *is a trace of* $\mathscr{E}$ *and* $last(t) = C_n$. *The set of (reachable) configurations of* $\mathscr{E}$ *is denoted by* $Conf(\mathscr{E})$, *and the set of traces of* $\mathscr{E}$ *— by* $Traces(\mathscr{E})$. *Clearly, any configuration* $C \in Conf(\mathscr{E})$ *is conflict-free, and any prefix of any trace* $t \in Traces(\mathscr{E})$ *belongs to* $Traces(\mathscr{E})$.
- *Two traces* $t = (A_1 \cup \underline{B}_1) \ldots (A_n \cup \underline{B}_n)$ $(n \geq 0)$ *and* $t' = (A'_1 \cup \underline{B'}_1) \ldots (A'_m \cup \underline{B'}_m)$ $(m \geq 0)$ *of* $\mathscr{E}$ *are called to be* equivalent w.r.t. $\sim$ *(denoted* $t \sim t'$*) iff* $last(t) = last(t')$.

The last two items of Definition 3 lead to the following auxiliary

Lemma 1. *Given an RPES* $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$, *it holds:*

(i) $\{last(t) \mid t \in Traces(\mathscr{E})\} = Conf(\mathscr{E})$;

(ii) for any $t \in Traces(\mathscr{E})$, *if* $t(A \cup \underline{B}) \in Traces(\mathscr{E})$ *then* $last(t) \xrightarrow{A \cup \underline{B}} last(t(A \cup \underline{B}))$;

(iii) for any $t, t' \in Traces(\mathscr{E})$, *if* $last(t) \xrightarrow{A \cup \underline{B}} last(t')$ *then* $t(A \cup \underline{B}) \in Traces(\mathscr{E})$ *and* $t(A \cup \underline{B}) \sim t'$.

Example 2. First, recall the RPES $\mathscr{E}_0 = (E_0, <_0, \sharp_0, l_0, F_0, \prec_0, \rhd_0, C_0^0)$ (see Example 1) with the components: $E_0 = \{a,b,c,d,e\}$; $<_0 = \{(b,d),(c,e)\}$; $\sharp_0 = \{(a,b),(b,a),(b,c),(c,b)\}$; l_0 is the identical function; $F_0 = \{b,c\}$; $\prec_0 = \{(b,\underline{b}),(c,\underline{c})\}$; $\rhd_0 = \emptyset$; $C_0^0 = \emptyset$. We shall check if the sequence $t = (\{b\} \cup \emptyset)(\{d\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})(\{c\} \cup \emptyset)(\{e\} \cup \emptyset)(\emptyset \cup \{\underline{c}\})$ is a trace of $\mathscr{E}_0$, using Definition 3. First, we need to show that $((A_1 = \{b\}) \cup (\underline{B}_1 = \emptyset))$ is enabled at C_0^0. Item a) is true because $(A_1 = \{b\}) \cap (C_0^0 = \emptyset) = \emptyset$, $B_1 = \emptyset \subseteq C_0^0$, and $(\emptyset \cup \{b\})$ is conflict-free. Item b) is correct, since the event b has no causes, i.e. there is no $e' \in E_0$ such that $e' <_0 b$. As $B_1 = \emptyset$, items c) and d) are met. Then, we have $C_0^0 \xrightarrow{\{b\} \cup \emptyset} C_1^0 = \{b\}$. Second, verify if $((A_2 = \{d\}) \cup (\underline{B}_2 = \emptyset))$ is enabled at C_1^0. We see that $(A_2 = \{d\}) \cap (C_1^0 = \{b\}) = \emptyset$, $B_2 = \emptyset \subseteq C_1^0$, and $\{b,d\}$ is conflict-free. Hence, item a) is correct. Item b) is met because d has the only cause b belonging to $C_1^0 \setminus B_2$. Due to $B_2 = \emptyset$, items c) and d) are true. So, we get $C_1^0 \xrightarrow{\{d\} \cup \emptyset} C_2^0 = \{b,d\}$. Third, make sure that $((A_3 = \emptyset) \cup (\underline{B}_3 = \{\underline{b}\}))$ is enabled at C_2^0. Items a) is fulfilled thanks to $A_3 = \emptyset$, $B_3 = \{b\} \subseteq C_2^0$, and C_2^0 is conflict-free. Clearly, item b) is true. Item c) holds because the only reverse cause for the event b is the event itself, which is in $\{b,d\} = (C_2^0 \setminus (B_3 \setminus \{b\}))$. As $\rhd_0 = \emptyset$, item d) is correct. Hence, we obtain $C_2^0 \xrightarrow{\emptyset \cup \{\underline{b}\}} C_3^0 = \{d\}$. Fourth, demonstrate that $((A_4 = \{c\}) \cup (\underline{B}_4 = \emptyset))$ is enabled at C_3^0. We see that $(A_4 = \{c\}) \cap (C_3^0 = \{d\}) = \emptyset$, $B_4 = \emptyset \subseteq C_3^0$, and $C_3^0 \cup A_4 = \{c,d\}$ is conflict-free. This means that item a) is correct. Item b) is met thanks to the fact that c has no causes. Because of $B_4 = \emptyset$, items c) and d) are met. Therefore, $C_3^0 \xrightarrow{\{c\} \cup \emptyset} C_4^0 = \{c,d\}$ is true. Fifth, check that $((A_5 = \{e\}) \cup (\underline{B}_5 = \emptyset))$ is enabled at C_4^0. Since $(A_5 = \{e\}) \cap (C_4^0 = \{c,d\}) = \emptyset$, $B_5 = \emptyset \subseteq C_4^0$, $(C_4^0 \cup A_5) = \{c,d,e\}$ is conflict-free, item a) is correct. As e has the only cause c belonging $C_4^0 \setminus B_5$, item b) is met. Due to $B_5 = \emptyset$, items c) and d) are true. Hence, we get $C_4^0 \xrightarrow{\{e\} \cup \emptyset} C_5^0 = \{c,d,e\}$. Finally, we examine if $((A_6 = \emptyset) \cup (\underline{B}_6 = \{\underline{c}\}))$ is enabled at C_5^0. Item a) is fulfilled thanks to $A_6 = \emptyset$, $B_6 = \{c\} \subseteq C_5^0$, and C_5^0 is conflict-free. Obviously, item b) is true. Item c) holds because the only reverse cause for the event c is the event itself, which is in

$\{c,d,e\} = (C_5^0 \setminus (B_6 \setminus \{c\}))$. Because of $\rhd_0 = \emptyset$, item d) is correct. So, we obtain $C_5^0 \stackrel{\emptyset\cup\{\underline{c}\}}{\longrightarrow} C_6^0 = \{d,e\}$. Thus, t is indeed a trace of $\mathscr{E}_0$.

Reasoning analogously, we get the following configurations of $\mathscr{E}_0$: $\emptyset$, $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$, $\{e\}$, $\{b,d\}$, $\{c,e\}$, $\{c,d\}$, $\{b,e\}$, $\{d,e\}$, $\{b,d,e\}$, $\{c,d,e\}$. Since the event a is independent with each of the events c, d, e, we get the additional configurations: $\{a,c\}$, $\{a,d\}$, $\{a,e\}$, $\{a,c,e\}$, $\{a,c,d\}$, $\{a,d,e\}$, $\{a,c,d,e\}$. Since the pair (a,b) ((b,c)) is in the conflict relation $\sharp_0$, the events a and b (b and c) cannot occur together in any configuration. Therefore, all the configurations of $\mathscr{E}_0$ are listed above. Some of the maximal traces of $\mathscr{E}_0$ are: $(t_1t_2)^*(\{a\}\cup\emptyset)t_2t_4t_2$, $(t_1t_2)^*t_4(t_1t_2)^*(\{a,c\}\cup\emptyset)t_5$, $(t_1t_2)^*t_3(t_1t_2)^*t_4(t_1t_2)^*(\{a,c\}\cup\emptyset)t_5$, $(t_1t_2)^*t_3(t_1t_2)^*(\{c\}\cup\emptyset)(\{a,e\}\cup\emptyset)t_5$, where $t_1 = (((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*$, $t_2 = (((\{c\}\cup\emptyset)(\emptyset\cup\{\underline{c}\}))^*$, $t_3 = (\{b\}\cup\emptyset)(\{d\}\cup\emptyset)(\emptyset\cup\{\underline{b}\})$, $t_4 = (\{c\}\cup\emptyset)(\{e\}\cup\emptyset)(\emptyset\cup\{\underline{c}\})$, $t_5 = (\emptyset\cup\{\underline{c}\})(\{c\}\cup\emptyset)$.

Second, consider the structure $\mathscr{E}_1 = (E_1, <_1, \sharp_1, l_1, F_1, \prec_1, \rhd_1, C_0^1)$, where $E_1 = \{a,b\}$; $<_1 = \{(a,b)\}$; $\sharp_1 = \emptyset$; l_1 is the identical function; $F_1 = \{a\}$; $\prec_1 = \{(a,\underline{a})\}$; $\rhd_1 = \emptyset$; $C_0^1 = \emptyset$. It is easy to see that $\mathscr{E}_1$ is an RPES. As the only pair (a,b) is in the causality relation $<_1$, i.e., the event a has no cause and it causes the event b, the event a can occur first and only after that b can happen. Then, we obtain the forward steps: $\emptyset \stackrel{(\{a\}\cup\emptyset)}{\rightarrow} \{a\} \stackrel{(\{b\}\cup\emptyset)}{\rightarrow} \{a,b\}$. The intended meaning of $a \prec_1 \underline{a}$ is that the event a can be undone if it has occurred in a configuration. In this regard, the reverse step $\{a\} \stackrel{(\emptyset\cup\{\underline{a}\})}{\rightarrow} \emptyset$ is possible, thanks to $(b,\underline{a}) \notin \prec_1$ and $\rhd_1 = \emptyset$. Moreover, the event a can be undone in the configuration $\{a,b\}$ even though the event b is present because $(b,\underline{a}) \notin \rhd_1$. This means that we can move backwards from $\{a,b\}$ to $\{b\}$ by executing the step $(\emptyset\cup\{\underline{a}\})$. Therefore, the configurations of $\mathscr{E}_1$ are $\emptyset$, $\{a\}$, $\{b\}$, $\{a,b\}$, and the traces of $\mathscr{E}_1$ are all prefixes of the trace $(((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a\}\cup\emptyset)(\{b\}\cup\emptyset)((\emptyset\cup\{\underline{a}\})(\{a\}\cup\emptyset))^*(\emptyset\cup\{\underline{a}\})$.

Third, examine the structure RPES $\mathscr{E}_2 = (E_2, <_2, \sharp_2, l_2, F_2, \prec_2, \rhd_2, C_0^2)$, where $E_2 = \{a,b\}$; $<_2 = \emptyset$; $\sharp_2 = \emptyset$; l_2 is the identical function; $F_2 = \{a\}$; $\prec_2 = \{(a,\underline{a})\}$; $\rhd_2 = \{(b,\underline{a})\}$; $C_0^2 = \emptyset$. It is not difficult to check that $\mathscr{E}_2$ is an RPES. As the causality relation $<_2$ and the conflict relation $\sharp_2$ are empty, the events a and b are independent, and, therefore, they can take place in any order. This leads to the following forward steps: $\emptyset \stackrel{(\{a\}\cup\emptyset)}{\rightarrow} \{a\} \stackrel{(\{b\}\cup\emptyset)}{\rightarrow} \{a,b\}$ and $\emptyset \stackrel{(\{b\}\cup\emptyset)}{\rightarrow} \{b\} \stackrel{(\{a\}\cup\emptyset)}{\rightarrow} \{a,b\}$. Since $b \rhd_2 \underline{a}$, we conclude that b prevents the undoing of a, i.e. a cannot be undone if b is present. So, we can go back from $\{a\}$ to $\emptyset$ by executing the step $(\emptyset\cup\{\underline{a}\})$ and cannot move backwards from $\{a,b\}$. The configurations of $\mathscr{E}_2$ are $\emptyset$, $\{a\}$, $\{b\}$, $\{a,b\}$, and the traces of $\mathscr{E}_2$ are all prefixes of the traces $(((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a\}\cup\emptyset)(\{b\}\cup\emptyset)$, $(((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a,b\}\cup\emptyset)$, $(((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{b\}\cup\emptyset)(\{a\}\cup\emptyset)$.

It is not difficult to verify the truth of Lemma 1 for all the RPESs discussed above. ◇

RPESs are able to model such a peculiarity of reversible computation as causal-consistent reversibility which relates reversibility with causality: an event can be undone provided that all of its effects have been undone. This allows the system to get back to a past state, which could only be reached by forward computation. This notion of reversibility is natural in reliable concurrent systems since when an error occurs the system tries to go back to a past consistent state.

Definition 4. *An RPES $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ is called*

- cause-respecting *if for any $e,e' \in E$, if $e < e'$ then $e \ll e'$;*
- causal *if for any $e \in E$ and $u \in F$ it holds: $e \prec \underline{u}$ iff $e = u$, and $e \rhd \underline{u}$ iff $u < e$.*

Informally, in the cause-respecting and causal RPES, causes can be only undone if their effects are not present in the current configuration. Clearly, if the RPES is causal, then it is cause-respecting as well.

Example 3. First, recall the RPES $\mathscr{E}_0$ (see Examples 1 and 2) with the components: $E_0 = \{a,b,c,d,e\}$; $<_0 = \{(b,d),(c,e)\}$; $\sharp_0 = \{(a,b),(b,a),(b,c),(c,b)\}$; l_0 is the identical function; $F_0 = \{b,c\}$; $\prec_0 =$

$\{(b,\underline{b}),(c,\underline{c})\}$; $\rhd_0=\emptyset$; $C_0^0=\emptyset$. We know from Example 1 that the sustained causation relation $\ll_0$ is empty, because the causality relation $<_0$ contains the pairs (b,d) and (c,e) and the prevention relation $\rhd_0$ is empty. Since $\ll_0 \neq <_0$, we have that this RPES is neither cause-respecting nor causal.

Second, consider the RPES $\mathscr{E}_1$ (see Example 2) with the components: $E_1=\{a,b\}$; $<_1=\{(a,b)\}$; $\sharp_1=\emptyset$; l_1 is the identical function; $F_1=\{a\}$; $\prec_1=\{(a,\underline{a})\}$; $\rhd_1=\emptyset$; $C_0^1=\emptyset$. It is easy to see that $\ll_1=\emptyset$, since $<_1=\{(a,b)\}$ and $(b,\underline{a})\notin\rhd_1$. Then, we obtain $<_1\neq\ll_1$. So, this RPES is neither cause-respecting nor causal.

Third, examine the RPES $\mathscr{E}_2=(E_2,<_2,\sharp_2,l_2,F_2,\prec_2,\rhd_2,C_0^2)$ (see Example 2) with the components: $E_2=\{a,b\}$; $<_2=\emptyset$; $\sharp_2=\emptyset$; l_2 is the identical function; $F_2=\{a\}$; $\prec_2=\{(a,\underline{a})\}$; $\rhd_2=\{(b,\underline{a})\}$; $C_0^2=\emptyset$. The RPES is cause-respecting, because the causality relation $<_2$ is empty, and, hence, for the only reversible event a of $\mathscr{E}_3$, the set of its effects is empty, which implies $<_2=\ll_2=\emptyset$. On the other hand, $\mathscr{E}_2$ is not causal, because there are the events a and b such that $b\rhd_2\underline{a}$ and $a\not<_2 b$.

Fourth, treat the RPES $\mathscr{E}_3=(E_3,<_3,\sharp_3,l_3,F_3,\prec_3,\rhd_3,C_0^3)$, where $E_3=\{a,b,c,d\}$; $<_3=\{(b,d),(c,d)\}$; $\sharp_3=\{(a,c),(c,a),(a,d),(d,a)\}$; l_3 is the identical function; $F_3=\{b\}$; $\prec_3=\{(a,\underline{b}),(b,\underline{b})\}$; $\rhd_3=\{(d,\underline{b})\}$ and $C_0^3=\{b\}$. Since for the only reversible event b, the set of its effects is equal to $\{d\}$ and $d\rhd_3\underline{b}$ is true, we conclude that the RPES is cause-respecting, whereas it is not causal because $(a,\underline{b})\in\prec_3$ and $a\neq b$.

Finally, consider the RPES $\mathscr{E}_4=(E_4,<_4,\sharp_4,l_4,F_4,\prec_4,\rhd_4,C_0^4)$, where $E_4=\{a,b,c,d\}$; $<_4=\{(c,d)\}$; $\sharp_4=\{(a,c),(c,a),(a,d),(d,a)\}$; l_4 is the identical function; $F_4=\{c,b\}$; $\prec_4=\{(b,\underline{b}),(c,\underline{c})\}$; $\rhd_4=\{(d,\underline{c})\}$, $C_0^4=\{b,c\}$. The RPES is causal and therefore cause-respecting. This is because $<_4=\{(c,d)\}$ and $\rhd_4=\{(d,\underline{c})\}$, and the reverse cause for the undoing of the only reversible event is the event itself, since we have $F_4=\{b,c\}$ and $\prec_4=\{(b,\underline{b}),(c,\underline{c})\}$. $\diamond$

Any cause-respecting RPES with the empty initial configuration can be presented as a PES. On the other hand, any PES can be converted into a causal and therefore cause-respecting RPES with the empty initial configuration, once we specify which events are to be reversible. The following facts are slight modifications of Propositions 3.36 and 3.37 from [27].

Proposition 1.

(i) If $\mathscr{E}=(E,<,\sharp,l,F,\prec,\rhd,\emptyset)$ is a cause-respecting RPES then $\phi(\mathscr{E})=(E,<,\sharp,l,\emptyset)$ is a PES.

(ii) If $\mathscr{E}=(E,<,\sharp,l,\emptyset)$ is a PES and $F\subseteq E$ then $\varphi(\mathscr{E},F)=(E,<,\sharp,l,F,\prec,\rhd,\emptyset)$ is a causal RPES, where $e\prec\underline{e}$ for any $e\in F$, and $e\rhd\underline{e'}$ for any $e\in E$ and $e'\in F$ such that $e'<e$. Moreover, $\phi(\varphi(\mathscr{E},F))=\mathscr{E}$.

The following lemma states specific features of the configurations of the cause-respecting RPES, which are left-closed w.r.t. causality and forwards reachable. Thanks to Definitions 2 and 3, the truth of item (i) follows from Proposition 3.38(1) [27], and the truth of item (ii) — from Proposition 3.40(2) [27].

Lemma 2. *Given a cause-respecting $\mathscr{E}$ and its configuration $C\in Conf(\mathscr{E})$, it holds:*

(i) C is left-closed under $<$;

(ii) if C is reachable, then C is forwards reachable.

The below example explains the above lemma.

Example 4. Recall the non-cause-respecting RPES $\mathscr{E}_0$ (with $<_0=\{(b,d),(c,e)\}$) from Examples 1–3. We know that $\{d\}$, $\{e\}$, $\{b,e\}$, $\{c,d\}$, $\{d,e\}$, $\{b,d,e\}$, $\{c,d,e\}$, $\{a,d\}$, $\{a,e\}$, $\{a,c,d\}$, $\{a,d,e\}$, $\{a,c,e,d\}$ are configurations of $\mathscr{E}_0$. Clearly, these configurations are not left-closed under $<_0$. Also,

we can reach the configurations only by using a combination of forward and reverse steps, i.e. the configurations are reachable but not forwards reachable.

Consider the non-cause-respecting RPES $\mathscr{E}_1$ (with $<_1=\{(a,b)\}$) from Examples 2–3. The configurations of $\mathscr{E}_1$ are $\emptyset$, $\{a\}$, $\{b\}$, $\{a,b\}$. We see that the configuration $\{b\}$ is not left-closed under $<_1$. In addition, the configuration $\{b\}$ can only be reached with a combination of forward and reverse steps, but this is not possible when doing only forward steps.

It is easy to check that in the cause-respecting RPES $\mathscr{E}_2$ from Examples 2–3, all its configurations are left-closed under its causality relation and, moreover, forwards reachable. $\diamond$

3 Residuals

The removal operator, the concept of which is based on deleting already executed configurations (traces) and events that conflict with the events presenting in the configurations (traces), is necessary for residual semantics.

Introduce the definition of the removal operator for RPESs by using their traces.

Definition 5. *For an RPES* $\mathscr{E}=(E,<,\sharp,l,F,\prec,\rhd,C_0)$ *and its trace* $t=(A_1\cup\underline{B}_1)\ \ldots\ (A_n\cup\underline{B}_n)\in Traces(\mathscr{E})$ *(*$n\geq 0$*), the residual* $\mathscr{E}\setminus t$ *of* $\mathscr{E}$ *after* t *under the removal operator* $\setminus$ *is defined by induction on* $0\leq i\leq n$ *as follows:*

$i=0$. $\mathscr{E}\setminus(t_0=\varepsilon)=\mathscr{E}$.

$i>0$. $\mathscr{E}\setminus t_i=(E^i,\ <^i=<^{i-1}\cap(E^i\times E^i),\ \sharp^i=\sharp^{i-1}\cap(E^i\times E^i),\ l^i=l^{i-1}\mid_{E^i},\ F^i,\ \prec^i=\prec^{i-1}\cap(E^i\times\underline{F}^i),$
$\rhd^i=\rhd^{i-1}\cap\ (E^i\times\underline{F}^i),\ C_0^i)$, *with*

- $E^i=E^{i-1}\setminus(\widetilde{A}_i\cup\sharp^{i-1}(\widetilde{A}_i))$, *where*
  $\widetilde{A}_i=(A_i\setminus F^{i-1})\ \cup\ (\lfloor(A_i\setminus F^{i-1})\rfloor\cap F^{i-1}=\{\widetilde{a}\in F^{i-1}\mid\exists a\in A_i\setminus F^{i-1}\colon\widetilde{a}<^{i-1}a\})$,
  $\sharp^{i-1}(\widetilde{A}_i)=\{a\in E^{i-1}\mid\exists\widetilde{a}\in\widetilde{A}_i\ :\ a\,\sharp^{i-1}\,\widetilde{a}\}$;
- $F^i=(F^{i-1}\cap E^i)\setminus\big(\hat{A}_i\cup\hat{\hat{A}}_i\big)$, *where*
  $\hat{A}_i=\{e\in F^{i-1}\mid\exists a\in\sharp^{i-1}(\widetilde{A}_i)\colon a\prec^{i-1}\underline{e}\}$,
  $\hat{\hat{A}}_i=\{e\in F^{i-1}\mid\exists a\in\widetilde{A}_i\colon a\rhd^{i-1}\underline{e}\}$;
- $C_0^i=((C_0^{i-1}\setminus B_i)\cup A_i)\cap E^i$.

$\mathscr{E}\setminus t=\mathscr{E}\setminus t_n$.

The intuitive interpretation of the above definition is as follows. In the process of constructing the residual of the RPES after a trace, all the irreversible events occurred in the current computation step, their reversible causes and conflicting events thereof are removed, yielding a reduction of all the relations, the labelling function and the initial configuration in the residual. This is due to the fact that all these removed events will never be able to occur in any subsequent step. In addition, reversible events become irreversible, whenever at least one of their reverse causes and/or at least one of the events preventing their undoing are eliminated because the reversible events can never be undone afterwards. At the same time, the other reversible events presented in the current step are retained, since they can be reversed in next steps.

It should be emphasized that for any trace t of the RPES $\varphi(\mathscr{E},\emptyset)^{(2)}$, where $\mathscr{E}$ is a PES, the residual $\varphi(\mathscr{E},\emptyset)\setminus t$ coincides with the residual $\mathscr{E}\ \setminus'\ last(t)$, where $\setminus'$ is the removal operator defined in [22]$^{(3)}$.

(2) See Proposition 1(ii).

(3) In [22], for the PES $\mathscr{E}=(E,<,\sharp,l)$ and its configuration $C\in Conf(\mathscr{E})$, the residual $\mathscr{E}\ \setminus'\ C$ is defined as follows: $\mathscr{E}\ \setminus'\ C=(E'=E\setminus(C\cup\sharp(C)),\leq\cap(E'\times E'),\sharp\cap(E'\times E'),l\mid_{E'})$, where $\sharp(C)$ denotes the events conflicting with the events in C.

We illustrate the application of the above removal operator with

Example 5. Consider the RPES $\mathscr{E}_2 = (E_2, <_2, \sharp_2, l_2, F_2, \prec_2, \rhd_2, C_0^2)$ (see Examples 2–4) with the components: $E_2 = \{a,b\}$; $<_2 = \emptyset$; $\sharp_2 = \emptyset$; l_2 is the identical function; $F_2 = \{a\}$; $\prec_2 = \{(a,\underline{a})\}$; $\rhd_2 = \{(b,\underline{a})\}$; $C_0^2 = \emptyset$. From Example 2 we know that the traces of $\mathscr{E}_2$ are $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*$, $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a\}\cup\emptyset)$, $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a\}\cup\emptyset)(\{b\}\cup\emptyset)$, $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a,b\}\cup\emptyset)$, $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{b\}\cup\emptyset)$, $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{b\}\cup\emptyset)(\{a\}\cup\emptyset)$.

Applying the removal operator to the RPES $\mathscr{E}_2$ and its traces, we obtain the following structures:

- $\tilde{\mathscr{E}}_2 = \mathscr{E}_2 \setminus (A_1 = \{a\} \cup \underline{B}_1 = \emptyset) = (\tilde{E} = E_2, \tilde{<} = <_2, \tilde{\sharp} = \sharp_2, \tilde{l} = l_2, \tilde{F} = F_2, \tilde{\prec} = \prec_2, \tilde{\rhd} = \rhd_2, \tilde{C}_0 = \{a\})$, because $(\widetilde{A}_1 \cup \sharp_2(\widetilde{A}_1)) = \emptyset$, due to $a \in F_2$, and $\tilde{C}_0 = ((C_0^2 = \emptyset) \cup (A_1 = \{a\})) \cap (\tilde{E} = \{a,b\}) = \{a\}$;
- $\hat{\mathscr{E}}_2 = \mathscr{E}_2 \setminus (A_1 = \{a\} \cup \underline{B}_1 = \emptyset)(A_2 = \emptyset \cup \underline{B}_2 = \{\underline{a}\}) = \mathscr{E}_2$, since $(\widetilde{A}_2 \cup \dot{\sharp}(\widetilde{A}_2)) = \emptyset$, thanks to $a \in \tilde{F}$, and $((\tilde{C}_0 = \{a\}) \setminus B_2 = \{a\}) \cap \hat{E}_2 = \emptyset$;
- $\check{\mathscr{E}}_2 = \mathscr{E}_2 \setminus (A_1 = \{a\} \cup \underline{B}_1 = \emptyset)(A_2 = \{b\} \cup \underline{B}_2 = \emptyset) = (\check{E} = \{a\}, \check{<} = \emptyset, \check{\sharp} = \emptyset; \check{l} = l_2|_{\{a\}}; \check{F} = \emptyset; \check{\prec} = \emptyset; \check{\rhd} = \emptyset, \check{C}_0 = \{a\})$, because $\widetilde{A}_2 = \{b\}$, due to $b \in A_2 \setminus \tilde{F}$, $a \notin \check{F}$, due to $(b,\underline{a}) \in \tilde{\rhd}$, and $\check{C}_0 = ((\tilde{C}_0 = \{a\}) \cup (A_2 = \{b\})) \cap (\check{E} = \{a\}) = \{a\}$;
- $\check{\mathscr{E}}_2 = \mathscr{E}_2 \setminus (A_1 = \{a,b\} \cup \underline{B}_1 = \emptyset) = \check{\mathscr{E}}_2$, since $\widetilde{A}_1 = \{b\}$, due to $b \in A_1 \setminus F_2$, $a \notin \check{F}$, due to $(b,\underline{a}) \in \rhd_2$, and $\check{C}_0 = ((C_2^0 = \emptyset) \cup (A_1 = \{a,b\})) \cap (\check{E} = \{a\}) = \{a\} = \check{C}_0$;
- $\dot{\mathscr{E}}_2 = \mathscr{E}_2 \setminus (A_1 = \{b\} \cup \underline{B}_1 = \emptyset) = (\dot{E} = \{a\}, \dot{<} = \emptyset, \dot{\sharp} = \emptyset, \dot{l} = l_2|_{\{a\}}, \dot{F} = \emptyset, \dot{\prec} = \emptyset, \dot{\rhd} = \emptyset, \dot{C}_0 = \emptyset)$, because $\widetilde{A}_1 = \{b\}$, due to $b \in A_1 \setminus F_2$, $a \notin \dot{F}$, due to $(b,\underline{a}) \in \rhd_2$, and $\dot{C}_0 = ((C_2^0 = \emptyset) \cup (A_1 = \{b\})) \cap (\dot{E} = \{a\}) = \emptyset$;
- $\ddot{\mathscr{E}}_2 = \mathscr{E}_2 \setminus (A_1 = \{b\} \cup \underline{B}_1 = \emptyset)(A_2 = \{a\} \cup \underline{B}_2 = \emptyset) = (\ddot{E} = \emptyset, \ddot{<} = \emptyset, \ddot{\sharp} = \emptyset, \ddot{l} = \emptyset, \ddot{F} = \emptyset, \ddot{\prec} = \emptyset, \ddot{\rhd} = \emptyset, \ddot{C}_0 = \emptyset)$, since $\widetilde{A}_2 = \{a\}$, due to $a \in A_2 \setminus \dot{F}$, and $\ddot{C}_0 = ((\dot{C}_0 = \emptyset) \cup (A_2 = \{a\})) \cap (\dot{E} = \emptyset) = \emptyset$.

Notice that the removal operator produces the same residuals after the different traces. For example, it is easy to see that:

$\mathscr{E}_2 \setminus (\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}) = \mathscr{E}_2 \setminus ((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*$,
$\mathscr{E}_2 \setminus (\{a\}\cup\emptyset) = \mathscr{E}_2 \setminus ((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a\}\cup\emptyset)$,
$\mathscr{E}_2 \setminus (\{a\}\cup\emptyset)(\{b\}\cup\emptyset) = \mathscr{E}_2 \setminus ((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a\}\cup\emptyset)(\{b\}\cup\emptyset)$,
$\mathscr{E}_2 \setminus (\{a,b\}\cup\emptyset) = \mathscr{E}_2 \setminus ((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a,b\}\cup\emptyset)$,
$\mathscr{E}_2 \setminus (\{b\}\cup\emptyset) = \mathscr{E}_2 \setminus ((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{b\}\cup\emptyset)$,
$\mathscr{E}_2 \setminus (\{b\}\cup\emptyset)(\{a\}\cup\emptyset) = \mathscr{E}_2 \setminus ((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{b\}\cup\emptyset)(\{a\}\cup\emptyset)$. ◇

Below are some technical facts specific to the removal operator for RPESs.

Lemma 3. *Given a cause-respecting RPES $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$, a trace $t = (A_1 \cup \underline{B_1}) \dots (A_n \cup \underline{B_n})$ $(C_0 \overset{A_1\cup B_1}{\rightarrow} C_1 \dots C_{n-1} \overset{A_n\cup B_n}{\rightarrow} C_n)$ $(n \geq 0)$ of $\mathscr{E}$, and $\mathscr{E} \setminus t = (E^n, <^n, \sharp^n, l^n, F^n, \prec^n, \rhd^n, C_0^n)$, it holds:*

(i) $E^j \subseteq E^i$, $F^j \subseteq F^i$, $l^j \subseteq l^i$, $\nabla^j \subseteq \nabla^i$ $(\nabla \in \{<, \sharp, \prec, \rhd\})$, *for any* $0 \leq i \leq j \leq n$;

(ii) $\mathscr{E} \setminus t_i$ *is a cause-respecting RPES, for any* $0 \leq i \leq n$;

(iii) $B_i \subseteq F^{i-1}$, *for any* $1 \leq i \leq n$;

(iv) $A_i \subseteq E^{i-1}$, *for any* $1 \leq i \leq n$;

(v) $\widetilde{A}_i \subseteq C_n$, *for any* $1 \leq i \leq n$;

(vi) $C_0^n = C_n \cap E^n$.

The following two statements demonstrate compositional properties of the residual operator for cause-respecting RPESs.

Proposition 2. *Given a cause-respecting RPES $\mathscr{E}$ with a trace $t \in Traces(\mathscr{E})$ and its residual $\mathscr{E}' = \mathscr{E} \setminus t$ with a trace $t' \in Traces(\mathscr{E}')$, it holds that $tt' \in Traces(\mathscr{E})$, and, moreover, $\mathscr{E} \setminus tt' = \mathscr{E}' \setminus t'$.*

So, it turned out that the concatenation of any trace t of the cause-respecting RPES $\mathscr{E}$ and any trace t' of the residual $\mathscr{E} \setminus t$ is a trace of $\mathscr{E}$, and, moreover, the residuals $\mathscr{E} \setminus tt'$ and $\mathscr{E} \setminus t \setminus t'$ coincide.

Example 6. First, consider the non-cause-respecting $\mathscr{E}_0 = (E_0, <_0, \sharp_0, l_0, F_0, \prec_0, \rhd_0, C_0^0)$ from Examples 1–4, where $E_0 = \{a,b,c,d,e\}$; $<_0 = \{(b,d),(c,e)\}$; $\sharp_0 = \{(a,b),(b,a),(b,c),(c,b)\}$; l_0 is the identical function; $F_0 = \{b,c\}$; $\prec_0 = \{(b,\underline{b}),(c,\underline{c})\}$; $\rhd_0 = \emptyset$; $C_0^0 = \emptyset$. As was demonstrated in Example 2, the sequences $(\{b\} \cup \emptyset)$, $(\{b\} \cup \emptyset)(\{d\} \cup \emptyset)$ are traces of $\mathscr{E}_0$. Construct the following residuals of $\mathscr{E}_0$:

- $\dot{\mathscr{E}}_0 = \mathscr{E}_0 \setminus (A_1 = \{b\} \cup \underline{B}_1 = \emptyset) = (\dot{E}_0 = E_0, \dot{<}_0 = <_0, \dot{\sharp}_0 = \sharp_0, \dot{l}_0 = l_0, \dot{F}_0 = F_0, \dot{\prec}_0 = \prec_0, \dot{\rhd}_0 = \rhd_0, \dot{C}_0 = \{b\})$, because $(\widetilde{A}_1 \cup \sharp_0(\widetilde{A}_1)) = \emptyset$, due to $b \in F_0$, and, moreover, $\dot{C}_0 = ((C_0^0 = \emptyset) \cup (A_1 = \{b\})) \cap (\dot{E}_0 = \{a,b,c,d,e\}) = \{b\}$;
- $\ddot{\mathscr{E}}_0 = \mathscr{E}_0 \setminus (A_1 = \{b\} \cup \underline{B}_1 = \emptyset)(A_2 = \{d\} \cup \underline{B}_2 = \emptyset) = \ddot{E}_0 = \{e\}$, $\ddot{<}_0 = \emptyset$, $\ddot{\sharp}_0 = \emptyset$, $\ddot{l}_0 = \dot{l}_0|_{\{e\}}$, $\ddot{F}_0 = \emptyset$, $\ddot{\prec}_0 = \emptyset$, $\ddot{\rhd}_0 = \emptyset$, $\ddot{C}_0 = \emptyset$), because $\widetilde{A}_2 = \{b,d\}$ thanks to $d \in A_2 \setminus \dot{F}_0$, $(b,d) \in \dot{<}_0$ and $b \in \dot{F}_0$, and $\dot{\sharp}_0(\widetilde{A}_2) = \{a,c\}$, due to $(a,b),(b,c) \in \dot{\sharp}_0$, and, moreover, $\dot{C}_0 = ((\dot{C}_0 = \{b\}) \cup (A_2 = \{d\})) \cap (\dot{E}_0 = \{e\}) = \emptyset$.

It is easy to see that $(\{e\} \cup \emptyset)$ is a trace of $\ddot{\mathscr{E}}_0$, whereas the sequence $(\{b\} \cup \emptyset)(\{d\} \cup \emptyset)(\{e\} \cup \emptyset)$ is not a trace of $\mathscr{E}_0$.

Using Examples 2–5, it is not difficult to make sure that Proposition 2 holds for the cause-respecting RPES $\mathscr{E}_2$. ◇

It is stated below that any suffix t' of any trace tt' of the cause-respecting RPES $\mathscr{E}$ is a trace of the residual $\mathscr{E} \setminus t$.

Proposition 3. *Given a cause-respecting RPES $\mathscr{E}$ with traces $t', t't'' \in Traces(\mathscr{E})$, $t'' \in Traces(\mathscr{E} \setminus t')$ holds.*

Example 7. Examine the non-cause-respecting RPES $\mathscr{E}_1$ from Examples 2–4, with the components: $E_1 = \{a,b\}$; $<_1 = \{(a,b)\}$; $\sharp_1 = \emptyset$; l_1 is the identical function; $F_1 = \{a\}$; $\prec_1 = \{(a,\underline{a})\}$; $\rhd_1 = \emptyset$; $C_0^1 = \emptyset$. We know that $t' = (\{a\} \cup \emptyset)(\{b\} \cup \emptyset)$ and $t = (\{a\} \cup \emptyset)(\{b\} \cup \emptyset)(\emptyset \cup \{\underline{a}\})(\{a\} \cup \emptyset)$ are traces of $\mathscr{E}_1$. Let $t'' = (\emptyset \cup \{\underline{a}\})(\{a\} \cup \emptyset)$. Using Definition 5, we obtain the RPES $\mathscr{E}_1 \setminus t' = (E_1' = \emptyset, <_1' = \emptyset, \sharp_1' = \emptyset, l_1' = \emptyset, F_1' = \emptyset, \prec_1' = \emptyset, \rhd_1' = \emptyset, C_0'^1 = \emptyset)$. It is clear that $Traces(\mathscr{E}_1 \setminus t') = \emptyset$. Therefore, we get $t'' \notin Traces(\mathscr{E}_1 \setminus t')$.

Using Examples 2–5, it is not difficult to check that Proposition 3 holds for the cause-respecting RPES $\mathscr{E}_2$. ◇

4 Transition System Semantics for Cause-Respecting RPESs

In this section, we first give some basic definitions concerning labeled transition systems. Then, we define the mappings $TC(\mathscr{E})$ and $TR(\mathscr{E})$, which associate two distinct kinds of transition systems – one whose states are configurations and one whose states are residuals – with the RPES $\mathscr{E}$ labeled over the set L of actions.

A transition system $T = (S, \rightarrow, i)$ labeled over a set $\mathscr{L}$ of labels consists of a set of states S, a transition relation $\rightarrow \subseteq S \times \mathscr{L} \times S$, and an initial state $i \in S$. Two transition systems labeled over $\mathscr{L}$ are

isomorphic if their states can be mapped one-to-one to each other, preserving transitions and initial states. We call a relation $R \subseteq S \times S'$ a *bisimulation* between transition systems $T = (S, \rightarrow, i)$ and $T' = (S', \rightarrow', i')$ over $\mathscr{L}$ iff $(i, i') \in R$, and for all $(s, s') \in R$ and $l \in \mathscr{L}$: if $(s, l, s_1) \in \rightarrow$ then $(s', l, s'_1) \in \rightarrow'$ and $(s_1, s'_1) \in R$, for some $s'_1 \in S'$; and if $(s', l, s'_1) \in \rightarrow'$ then $(s, l, s_1) \in \rightarrow$ and $(s_1, s'_1) \in R$, for some $s_1 \in S$. Two transition systems over $\mathscr{L}$ are *bisimilar* if there is a bisimulation between them.

For a fixed set L of actions in RPESs, define the set $\mathbb{L} := \mathbb{N}_0^L$ (the set of multisets over L, or functions from L to the non-negative integers). The set $\mathbb{L}$ will be used as the set of labels in transition systems.

We are ready to define transition systems (labeled over $\mathbb{L}$) with configurations as states.

Definition 6. *For an RPES $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ over L,*

$$TC(\mathscr{E}) \text{ is a transition system } (Conf(\mathscr{E}), \multimap, C_0) \text{ over } \mathbb{L},$$

where $C \xrightarrow{M} C'$ iff $C \overset{(A \cup \underline{B})}{\rightarrow} C'$ in $\mathscr{E}$ and $M = l(A \cup \underline{B})$[4].

Let us explain the above definition with

Example 8. Consider the cause-respecting RPES $\mathscr{E}_2$ from Examples 2–5. In Example 2, we can see that $C_0^2 = \emptyset$ and $Conf(\mathscr{E}_2) = \{\emptyset, \{a\}, \{b\}, \{a,b\}\}$. Using Definition 6, we obtain $\multimap = \{(\emptyset, (\{a\} \cup \emptyset), \{a\}), (\{a\}, (\emptyset \cup \{a\}), \emptyset), (\{a\}, (\{b\} \cup \emptyset), \{a,b\}), (\emptyset, (\{b\} \cup \emptyset), \{b\}), (\{b\}, (\{a\} \cup \emptyset), \{a,b\}), (\emptyset, (\{a,b\} \cup \emptyset), \{a,b\})\}$. A graphical representation of the configuration transition system $TC(\mathscr{E}_2)$ is shown in Fig. 1. $\diamond$

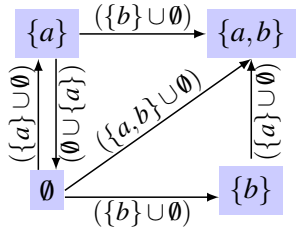


Figure 1: The configuration transition system $TC(\mathscr{E}_2)$

We next propose the definition of labeled transition systems over $\mathbb{L}$ with RPESs as states.

Definition 7. *For an RPES $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ over L,*

$$TR(\mathscr{E}) \text{ is a transition system } (Reach(\mathscr{E}), \rightharpoonup, \mathscr{E}) \text{ over } \mathbb{L},$$

where $\mathscr{F} \overset{M}{\rightharpoonup} \mathscr{F}'$ iff $\mathscr{F}' = \mathscr{F} \setminus (A \cup \underline{B})$ and $M = l(A \cup \underline{B})$, and $Reach(\mathscr{E}) = \{\mathscr{F} \mid \exists \mathscr{E}_0, \ldots, \mathscr{E}_k \ (k \geq 0)$ s.t. $\mathscr{E}_0 = \mathscr{E} \setminus \varepsilon$, $\mathscr{E}_k = \mathscr{F}$, and $\mathscr{E}_i \overset{l(A \cup \underline{B})}{\rightharpoonup} \mathscr{E}_{i+1} \ (0 \leq i < k)\}$.

We illustrate the above definition with

Example 9. Consider the RPES $\mathscr{E}_2$ from Examples 2–5. Using Definitions 5 and 7, we construct the residual transition system $TR(\mathscr{E}_2)$ which is depicted in Fig. 2. It is easy to check that the configuration transition system $TC(\mathscr{E}_2)$ (see Fig. 1) and the residual transition system $TR(\mathscr{E}_2)$ are bisimilar but not isomorphic. $\diamond$

We establish the relationships between the states and transitions of the configuration-based and residual-based transition systems of the RPES.

[4] See Definition 3.

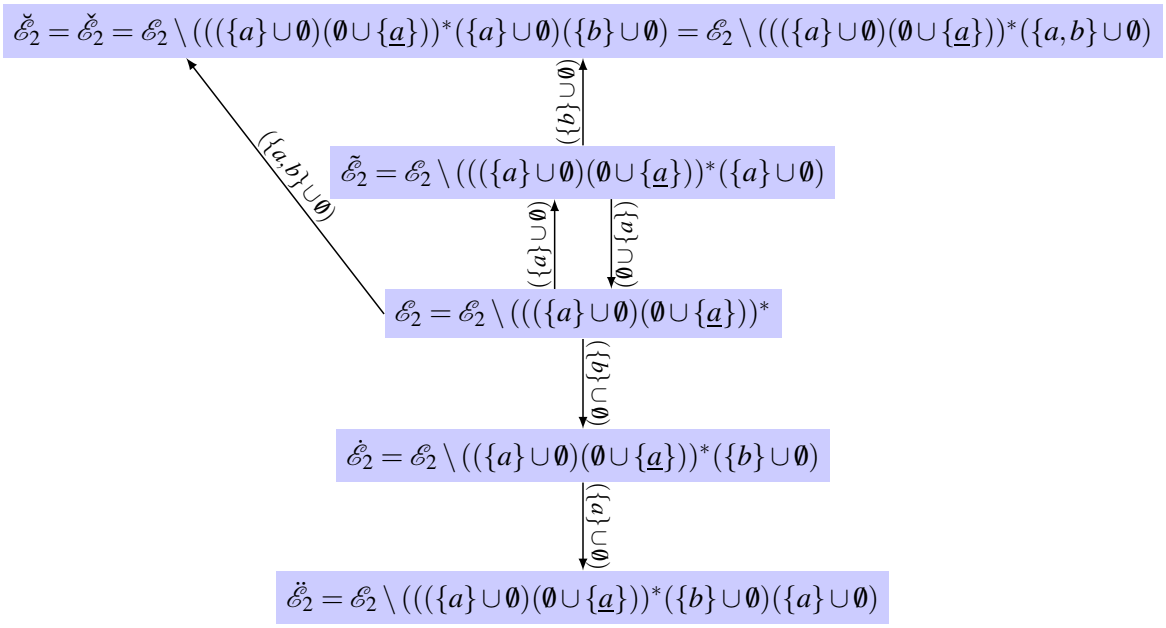


Figure 2: The residual transition system $TR(\mathscr{E}_2)$

Proposition 4. *Given a cause-respecting RPES $\mathscr{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ over L,*

(i) for any $last(t) \in Conf(\mathscr{E})$, $\mathscr{E} \setminus t \in Reach(\mathscr{E})$;

(ii) for any $\mathscr{E}' \in Reach(\mathscr{E})$, there is $last(t) \in Conf(\mathscr{E})$ such that $\mathscr{E}' = \mathscr{E} \setminus t$;

(iii) for any $last(t)$, $last(t') \in Conf(\mathscr{E})$, if $last(t) \overset{l(A \cup \underline{B})}{\rightharpoonup} last(t')$ then $\mathscr{E} \setminus t \overset{l(A \cup \underline{B})}{\rightharpoonup} \mathscr{E} \setminus t(A \cup \underline{B})$ and $last(t(A \cup \underline{B})) = last(t')$;

(iv) for any $\mathscr{E}', \mathscr{E}'' \in Reach(\mathscr{E})$, if $\mathscr{E}' \overset{l(A \cup \underline{B})}{\rightharpoonup} \mathscr{E}''$ then, for any $last(t) \in Conf(\mathscr{E})$ such that $\mathscr{E}' = \mathscr{E} \setminus t$, there is $last(t') \in Conf(\mathscr{E})$ such that $\mathscr{E}'' = \mathscr{E} \setminus t'$ and $last(t) \overset{l(A \cup \underline{B})}{\rightharpoonup} last(t')$.

Theorem 1. *Given a cause-respecting RPES $\mathscr{E}$ over L, $TC(\mathscr{E})$ and $TR(\mathscr{E})$ are bisimilar and in general not isomorphic.*

Proof. From Example 9 we know that, for the cause-respecting RPES $\mathscr{E}_2$, $TC(\mathscr{E}_2)$ and $TR(\mathscr{E}_2)$ are not isomorphic.

We shall check that $TC(\mathscr{E})$ and $TR(\mathscr{E})$ are bisimilar for an arbitrary cause-respecting RPES $\mathscr{E} = (E, <, \sharp, L, l, F, \prec, \rhd, C_0)$. Due to Lemma 1(i) and Propositions 4(i), we can define a relation $R \subseteq Conf(\mathscr{E}) \times Reach(\mathscr{E})$ as follows: $R = \{(last(t), \mathscr{E} \setminus t) \mid t \in Traces(\mathscr{E})\}$.

We need to show that R is a bisimulation between $TC(\mathscr{E})$ and $TR(\mathscr{E})$. Clearly, we have that $\varepsilon \in Traces(\mathscr{E})$, and, moreover, $C_0 = last(\varepsilon) \in Conf(\mathscr{E})$ and $\mathscr{E} = \mathscr{E} \setminus \varepsilon \in Reach(\mathscr{E})$. So, $(C_0, \mathscr{E}) \in R$ holds. Take an arbitrary $(last(t), \mathscr{E} \setminus t) \in R$. Suppose that $last(t) \overset{l(A \cup \underline{B})}{\rightharpoonup} C'$ in $TC(\mathscr{E})$ for some $C' \in Conf(\mathscr{E})$. By Lemma 1(i), there is $t' \in Traces(\mathscr{E})$ such that $C' = last(t')$. According to Proposition 4(iii), it is true that $\mathscr{E} \setminus t \overset{l(A \cup \underline{B})}{\rightharpoonup} \mathscr{E} \setminus t(A \cup \underline{B})$ and $last(t(A \cup \underline{B})) = last(t')$. Thanks to Lemma 1(i), we have $t(A \cup \underline{B}) \in Traces(\mathscr{E})$. Hence, $(C' = last(t(A \cup \underline{B})), \mathscr{E} \setminus t(A \cup \underline{B})) \in R$ holds. In the opposite direction, assume that

$\mathscr{E} \setminus t \stackrel{l(A \cup \underline{B})}{\longrightarrow} \mathscr{E}'$ in $TR(\mathscr{E})$ for some $\mathscr{E}' \in Reach(\mathscr{E})$. Due to Propositions 4(iv), for $last(t) \in Conf(\mathscr{E})$, there is $last(t') \in Conf(\mathscr{E})$ such that $\mathscr{E}' = \mathscr{E} \setminus t'$ and $last(t) \stackrel{l(A \cup \underline{B})}{\longrightarrow} last(t')$. Due to Lemma 1(i), $t' \in Traces(\mathscr{E})$ is true. This implies that $(last(t'), \mathscr{E} \setminus t' = \mathscr{E}') \in R$ holds. Hence, R is indeed a bisimulation. □

5 Concluding Remarks

In this paper, we dealt with two different – configuration-based and residual-based – ways of giving (step) transition system semantics for cause-respecting reversible prime event structures which encompass prime event structures. For this purpose, we firstly defined (step) semantics from [27], which is based on configurations/traces obtained by starting with the initial configuration and by executing events and/or undoing previously executed events, and, secondly, developed a removal operator which is useful for constructing residuals (model fragments) by retaining an appropriate amount of structure during the execution of the model. We also stated some correctness criteria for the removal operator. The meaning of the correctness properties is that the obtained residuals do not allow configurations/traces that are disallowed by the original structure. Also, in some sense, this signifies some compositionality properties of the removal operator. It turned out that in the context of PESs, the removal operator developed here produces the same residuals as the removal operator proposed in [22]. As our main result, we have obtained a (step) bisimulation between configuration-based and residual-based transition systems of the models under consideration. The configuration-based method discussed here can be useful in analyzing the state space of reversible concurrent systems whose behavior is represented as RPESs, and the proposed residual-based method can be suitable for specification and visualization of changes in the structures of reversible concurrent processes during their simulation in tools. Due to the good compositionality properties of the residual-based transition systems of RPESs and their complementarity and consistency with the configuration-based ones, it is hoped that the results obtained here may be helpful in demonstrating the correspondence between operational and denotational semantics of algebraic calculi of reversible concurrent processes, similar to how the results from [5, 9, 17] have found their application in traditional (irreversible) process algebras.

As for future work, we plan to broaden the list of studied models by adding flow/bundle/general event structures with symmetric and asymmetric conflict. Work on extending our approach to out-of-causal reversible prime event structures is under way and has yielded promising intermediate results. Another future line of our research is to generalize the model of reversible prime event structures with non-executable (impossible) events (for example, by dropping the transitivity/acyclicity of causality, as well as the principles of finite causes) in order to obtain isomorphisms between the two types of transition systems of the models, as was done for the corresponding extension of PESs in the paper [8]. There, the authors have been able to argue that non-executable events are useful in comparative semantics, facilitating the elimination of non-fundamental inconsistencies between models. Furthermore, isomorphisms between the transition system semantics are expected to allow one to relate those constructed on configurations and those derived from denotational semantics of process calculi in a tight way.

References

[1] Bogdan Aman & Gabriel Ciobanu (2018): *Controlled Reversibility in Reaction Systems*. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa & Claudio Zandron, editors: *Membrane Computing*, Springer International Publishing, Cham, pp. 40–53, doi:10.1007/978-3-319-73359-3_3.

[2] Youssef Arbach, David Karcher, Kirstin Peters & Uwe Nestmann (2015): *Dynamic Causality in Event Structures*. In Susanne Graf & Mahesh Viswanathan, editors: *Formal Techniques for Distributed Objects, Components, and Systems*, Springer International Publishing, Cham, pp. 83–97, doi:10.1007/978-3-319-19195-9_6.

[3] Abel Armas-Cervantes, Paolo Baldan & Luciano Garcia-Banuelos (2016): *Reduction of event structures under history preserving bisimulation*. *Journal of Logical and Algebraic Methods in Programming* 85(6), pp. 1110–1130, doi:10.1016/j.jlamp.2015.10.004.

[4] Clement Aubert & Ioana Cristescu (2017): *Contextual equivalences in configuration structures and reversibility*. *Journal of Logical and Algebraic Methods in Programming* 86(1), pp. 77–106, doi:10.1016/j.jlamp.2016.08.004.

[5] Christel Baier & Mila Majster-Cederbaum (1994): *The connection between an event structure semantics and an operational semantics forTCSP*. *Acta Informatica* 31(1), doi:10.1007/BF01178923.

[6] Kamila Barylska, Anna Gogolinska, Lukasz Mikulski, Anna Philippou, Marcin Piatkowski & Kyriaki Psara (2022): *Formal Translation from Reversing Petri Nets to Coloured Petri Nets*. In Claudio Antares Mezzina & Krzysztof Podlaski, editors: *Reversible Computation*, Springer International Publishing, Cham, pp. 172–186, doi:10.1007/978-3-031-09005-9_12.

[7] Eike Best, Nataliya Gribovskaya & Irina Virbitskaite (2017): *Configuration- and Residual-Based Transition Systems for Event Structures with Asymmetric Conflict*. In Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey & Tiziana Margaria, editors: *SOFSEM 2017: Theory and Practice of Computer Science*, Springer International Publishing, Cham, pp. 132–146, doi:10.1007/978-3-319-51963-0_11.

[8] Eike Best, Nataliya Gribovskaya & Irina Virbitskaite (2018): *From Event-Oriented Models to Transition Systems*. In Victor Khomenko & Olivier H. Roux, editors: *Application and Theory of Petri Nets and Concurrency*, Springer International Publishing, Cham, pp. 117–139, doi:10.1007/978-3-319-91268-4_7.

[9] Gérard Boudol (1990): *Flow event structures and flow nets*. In Irène Guessarian, editor: *Semantics of Systems of Concurrent Processes*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 62–95, doi:10.1007/3-540-53479-2_4.

[10] Silvia Crafa, Daniele Varacca & Nobuko Yoshida (2012): *Event Structure Semantics of Parallel Extrusion in the π-Calculus*. In Lars Birkedal, editor: *Foundations of Software Science and Computational Structures*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 225–239, doi:10.1007/978-3-642-28729-9_15.

[11] Vincent Danos & Jean Krivine (2005): *Transactions in RCCS*. In Martín Abadi & Luca de Alfaro, editors: *CONCUR 2005 – Concurrency Theory*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 398–412, doi:10.1007/11539452_31.

[12] Alexis De Vos, Stijn De Baerdemacker & Yvan Van Rentergem (2018): *Synthesis of Quantum Circuits vs. Synthesis of Classical Reversible Circuits*, first edition. Springer Cham, doi:10.1007/978-3-031-79895-5.

[13] David de Frutos Escrig, Maciej Koutny & Łukasz Mikulski (2019): *Reversing Steps in Petri Nets*. In Susanna Donatelli & Stefan Haar, editors: *Application and Theory of Petri Nets and Concurrency*, Springer International Publishing, Cham, pp. 171–191, doi:10.1007/978-3-030-21571-2_11.

[14] Eva Graversen, Iain Phillips & Nobuko Yoshida (2021): *Event structure semantics of (controlled) reversible CCS*. *Journal of Logical and Algebraic Methods in Programming* 121, p. 100686, doi:10.1016/j.jlamp.2021.100686.

[15] P.W. Hoogers, H.C.M. Kleijn & P.S. Thiagarajan (1996): *An event structure semantics for general Petri nets*. *Theoretical Computer Science* 153(1), pp. 129–170, doi:10.1016/0304-3975(95)00120-4.

[16] Jarkko Kari (2018): *Reversible Cellular Automata: From Fundamental Classical Results to Recent Developments*. *New Generation Computing* 36(3), pp. 145–172, doi:10.1007/s00354-018-0034-6.

[17] Joost-Pieter Katoen (1996): *Quantitative and Qualitative Extensions of Event Structures*. Ph.D. thesis, University of Twente, Netherlands.

[18] Stefan Kuhn, Bogdan Aman, Gabriel Ciobanu, Anna Philippou, Kyriaki Psara & Irek Ulidowski (2020): *Reversibility in Chemical Reactions*, pp. 151–176. Springer International Publishing, Cham, doi:10.1007/978-3-030-47361-7_7.

[19] Ivan Lanese, Claudio Antares Mezzina & Jean-Bernard Stefani (2016): *Reversibility in the higher-order π-calculus*. *Theoretical Computer Science* 625, pp. 25–84, doi:10.1016/j.tcs.2016.02.019.

[20] Ivan Lanese, Adrián Palacios & Germán Vidal (2019): *Causal-Consistent Replay Debugging for Message Passing Programs*. In Jorge A. Pérez & Nobuko Yoshida, editors: *Formal Techniques for Distributed Objects, Components, and Systems*, Springer International Publishing, Cham, pp. 167–184, doi:10.1007/978-3-030-21759-4_10.

[21] Rom Langerak (1991): *Bundle event structures: a non-interleaving semantics for LOTOS*. In Michel Diaz & Roland Groz, editors: *Formal Description Techniques V*, IFIP transactions C, Communication systems, North Holland, Netherlands, pp. 331–346. 5th International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols, FORTE 1992.

[22] Mila Majster-Cederbaum & Markus Roggenbach (1998): *Transition systems from event structures revisited*. *Information Processing Letters* 67(3), pp. 119–124, doi:10.1016/S0020-0190(98)00105-7.

[23] Doriana Medić, Claudio Antares Mezzina, Iain Phillips & Nobuko Yoshida (2020): *Towards a Formal Account for Software Transactional Memory*. In Ivan Lanese & Mariusz Rawski, editors: *Reversible Computation*, Springer International Publishing, Cham, pp. 255–263, doi:10.1007/978-3-030-52482-1_16.

[24] Hernan Melgratti, Claudio Antares Mezzina & G. Michele Pinna (2021): *A distributed operational view of Reversible Prime Event Structures*. In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–13, doi:10.1109/LICS52264.2021.9470623.

[25] Hernan Melgratti, Claudio Antares Mezzina, Iain Phillips, G. Michele Pinna & Irek Ulidowski (2020): *Reversible Occurrence Nets and Causal Reversible Prime Event Structures*. In Ivan Lanese & Mariusz Rawski, editors: *Reversible Computation*, Springer International Publishing, Cham, pp. 35–53, doi:10.1007/978-3-030-52482-1_2.

[26] Anna Philippou & Kyriaki Psara (2020): *Reversible Computation in Cyclic Petri Nets*. *CoRR* abs/2010.04000, doi:10.48550/arXiv.2010.04000.

[27] Iain Phillips & Irek Ulidowski (2015): *Reversibility and asymmetric conflict in event structures*. *Journal of Logical and Algebraic Methods in Programming* 84(6), pp. 781–805, doi:10.1016/j.jlamp.2015.07.004.

[28] Iain Phillips, Irek Ulidowski & Shoji Yuen (2013): *A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway*. In Robert Glück & Tetsuo Yokoyama, editors: *Reversible Computation*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 218–232, doi:10.1007/978-3-642-36315-3_18.

[29] Giovanni Michele Pinna (2017): *Reversing steps in membrane systems computations*. In M. Gheorghe, G. Rozenberg, A. Salomaa & C. Zandron, editors: *Membrane Computing*, 10725, Springer International Publishing, Cham, pp. 245–261, doi:10.1007/978-3-319-73359-3_16.

[30] Irek Ulidowski, Iain Phillips & Shoji Yuen (2018): *Reversing event structures*. *New Generation Computing* 36(3), pp. 281–306, doi:10.1007/s00354-018-0040-8.

[31] R.J. van Glabbeek & G.D. Plotkin (2009): *Configuration structures, event structures and Petri nets*. *Theoretical Computer Science* 410(41), pp. 4111–4159, doi:10.1016/j.tcs.2009.06.014.

[32] Glynn Winskel (1980): *Events in computation*. Ph.D. thesis, University of Edinburgh.

[33] Glynn Winskel (1989): *An introduction to event structures*. In J. W. de Bakker, W. P. de Roever & G. Rozenberg, editors: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 364–397, doi:10.1007/BFb0013026.

On Minimal Pumping Constants for Regular Languages

Markus Holzer Christian Rauch
Institut für Informatik, Universität Giessen, Arndstr. 2, 35392 Giessen, Germany
holzer@informatik.uni-giessen.de christian.rauch@informatik.uni-giessen.de

The study of the operational complexity of minimal pumping constants started in [J. DASSOW and I. JECKER. Operational complexity and pumping lemmas. *Acta Inform.*, 59:337–355, 2022], where an almost complete picture of the operational complexity of minimal pumping constants for two different variants of pumping lemmata from the literature was given. We continue this research by considering a pumping lemma for regular languages that allows pumping of sub-words at any position of the considered word, if the sub-word is long enough [S. J. SAVITCH. Abstract Machines and Grammars. 1982]. First we improve on the simultaneous regulation of minimal pumping constants induced by different pumping lemmata including Savitch's pumping lemma. In this way we are able to simultaneously regulate four different minimal pumping constants. This is a novel result in the field of descriptional complexity. Moreover, for Savitch's pumping lemma we are able to completely classify the range of the minimal pumping constant for the operations Kleene star, reversal, complement, prefix- and suffix-closure, union, set-subtraction, concatenation, intersection, and symmetric difference. In this way, we also solve some of the open problems from the paper that initiated the study of the operational complexity of minimal pumping constants mentioned above.

1 Introduction

Pumping lemmata are fundamental to the study of formal languages. An annotated bibliography on variants of pumping lemmata for regular and context-free languages is given in [9]. One variant of the pumping lemma states that for any regular language L, there exists a constant p (depending on L) such that any word w in the language of length at least p can be split into three parts $w = xyz$, where y is non-empty, and xy^tz is also in the language, for every $t \geq 0$—see Lemma 1. By the contrapositive one can prove that certain languages are not regular. Since the aforementioned pumping lemma is only a necessary condition, it may happen that such a proof fails for a particular language such as, e.g., $\{ a^m b^n c^n \mid m \geq 1 \text{ and } n \geq 0 \} \cup \{ b^m c^n \mid m,n \geq 0 \}$. The application of pumping lemmata is not limited to prove non-regularity. For instance, they also imply an algorithm that decides whether a regular language is finite or not. A regular language L is *infinite* if and only if there is a word of length at least p, where p is the aforementioned constant of the pumping lemma.[1] Here a small p is beneficial. Thus, for instance, the question arises on how to determine a small or smallest value for the constant p such that the pumping lemma is still satisfied.

For a regular language L the value of p in the above-mentioned pumping lemma can always be chosen to be the number of states of a finite automaton, regardless whether it is deterministic or nondeterministic, accepting L. Consider the unary language $a^n a^*$, where all values p with $0 \leq p \leq n$ do *not* satisfy the property of the pumping lemma, but $p = n+1$ does. A closer look on some example languages reveals that sometimes a much smaller value suffices. For instance, consider the language

$$L = a^* + a^*bb^* + a^*bb^*aa^* + a^*bb^*aa^*bb^*,$$

[1] For the other pumping lemma constants p considered in this paper, the statement on infiniteness can be strengthened to: a regular language L is *infinite* if and only if there is a word of length ℓ with $p < \ell \leq 2p$. This also holds true if p refers to the deterministic state complexity of a language.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 127–141, doi:10.4204/EPTCS.386.11

which is accepted by a (minimal) deterministic finite automaton with five states, the sink state included, but already for $p = 1$ the statement of the pumping lemma is satisfied. It is easy to see that regardless whether the considered word starts with a or b, this letter can be readily pumped. Thus, the minimal pumping constant satisfying the statement of pumping lemma for the language L is 1, because the case $p = 0$ is equivalent to $L = \emptyset$. This leads to the notation of a minimal pumping constant for a language L w.r.t. a particular pumping lemma, which is the smallest number p such that the pumping lemma under consideration for the language L is satisfied.

Recently minimal pumping lemmata constants were investigated from a descriptional complexity perspective in [2]. Besides basic facts on these constants for two specific pumping lemmata [1, 6, 8, 10] their relation to each other and their behaviour under regularity preserving operations was studied in detail. In fact, it was proven that for three natural numbers p_1, p_2, and p_3 with $1 \leq p_1 \leq p_2 \leq p_3$, there is a regular language L over a growing size alphabet such that $\mathtt{mpc}(L) = p_1$, $\mathtt{mpl}(L) = p_2$, and $\mathtt{sc}(L) = p_3$, where `mpc` (`mpl`, respectively) refers to the minimal pumping constant induced by the pumping lemma from [8] (from [1, 6, 10], respectively) and `sc` is the abbreviation of the deterministic state complexity. This simultaneous regulation of three measures is novel in descriptional complexity theory. For the exact statements of the pumping lemmata mentioned above we refer to Lemma 1 and its following paragraph. The *operational complexity of pumping* or *pumping lemmata* for an n-ary regularity preserving operation $\circ$ undertaken in [2] is in line with other studies on the operational complexity of other measures for regular languages such as the state complexity or the accepting state complexity to mention a few. The operational complexity of pumping is the study of the set $g_\circ(k_1,k_2,\ldots,k_n)$ of all numbers k such that there are regular languages $L_1,L_2,\ldots,L_n$ with minimal pumping complexity $k_1,k_2,\ldots,k_n$, respectively, and the language $L_1 \circ L_2 \circ \cdots \circ L_n$ has minimal pumping complexity k. In [2] a complete picture for the operational complexity w.r.t. the pumping lemma from [8] (measure `mpc`) for the operations Kleene closure, complement, reversal, prefix and suffix-closure, circular shift, union, intersection, set-subtraction, symmetric difference, and concatenation was given—see Table 1 on page 138. However, for the pumping lemma from [1, 6, 10] (measure `mpl`) some results from [2] are only partial (set-subtraction and symmetric difference) and others even remained open (circular shift and intersection); for comparison see the table mentioned above. The behaviour of these measures differ with respect to finiteness/infinity of ranges, due to the fact that for the pumping lemma from [1, 6, 10] the pumping has to be done within a prefix of bounded length.

This is the starting point of our investigation. As a first step we improve on the above mentioned result on the simultaneous regulation of minimal pumping constants showing that already a binary language suffices. If we additionally also consider a fourth measure (`mps`) induced by the pumping lemma from [11], we obtain a similar result for a quinary language. Thus, we are able to regulate four descriptional complexity measures simultaneously on a single regular language. Savitch's pumping lemma allows pumping of sub-words at any position of the considered word, if the sub-word is long enough—see Lemma 3. Moreover, the outcome of our study on the operational complexity of pumping presents a comprehensive view for the previously mentioned operations. In passing, we can also solve all the partial and open problems from [2], completing the overall picture for the three pumping lemmata in question—see the gray shaded entries in Table 1 on page 138. This provides a full understanding of the operational complexity of these pumping lemmata. it is worth mentioning that the obtained result are very specific to the considered pumping lemmata—compare with [3, 5] where descriptional and computational complexity aspects of Jaffe's pumping lemma [7] are considered. For instance, the simultaneous regulation of pumping constants involving those satisfying Jaffe's pumping lemma seems to be much more complicated, since only the deterministic state complexity can serve as an upper bound, while the nondeterministic state complexity becomes incomparable. Due to space constraints almost all proofs are

omitted; they can be found in the full version of this paper.

2 Preliminaries

We recall some definitions on finite automata as contained in [4]. Let Σ be an alphabet. Then, as usual Σ^* refers to the set of all words over the alphabet Σ, including the empty word λ, and $\Sigma^{\leq k}$ denotes the set of all words of length at most k. For a word $w = a_1 a_2 \dots a_n \in \Sigma^*$ and a natural number $k \geq 1$ we refer to the word $a_1 a_2 \dots a_k$, if $k \leq n$, and $a_1 a_2 \dots a_n$, otherwise, as the *k-prefix of w*. If $k = 0$, then λ is the unique 0-prefix of any word. Analogously one can define the *k-suffix of a word w*.

A *deterministic finite automaton* (DFA) is a quintuple $A = (Q, \Sigma, \cdot, q_0, F)$, where Q is the finite set of *states*, Σ is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and the *transition function* $\cdot$ maps $Q \times \Sigma$ to Q. The *language accepted* by the DFA A is defined as $L(A) = \{ w \in \Sigma^* \mid q_0 \cdot w \in F \}$, where the transition function is recursively extended to a mapping $Q \times \Sigma^* \to Q$ in the usual way. Finally, a finite automaton is *unary* if the input alphabet Σ is a singleton set, that is, $\Sigma = \{a\}$, for some input symbol a. The *deterministic state complexity of a finite automaton A* with state set Q is referred to as $\mathtt{sc}(A) := |Q|$ and the *deterministic state complexity of a regular language L* is defined as

$$\mathtt{sc}(L) = \min\{ \mathtt{sc}(A) \mid A \text{ is a DFA accepting } L, \text{ i.e., } L = L(A) \}.$$

A finite automaton is *minimal* if its number of states is minimal with respect to the accepted language. It is well known that each minimal DFA is isomorphic to the DFA induced by the Myhill-Nerode equivalence relation. The *Myhill-Nerode* equivalence relation $\sim_L$ for a language $L \subseteq \Sigma^*$ is defined as follows: for $u, v \in \Sigma^*$ let $u \sim_L v$ if and only if $uw \in L \iff vw \in L$, for all $w \in \Sigma^*$. The equivalence class of u is referred to as $[u]_L$ or simply $[u]$ if the language is clear from the context and it is the set of all words that are equivalent to u w.r.t. the relation $\sim_L$, i.e., $[u]_L = \{ v \mid u \sim_L v \}$.

Regular languages satisfy a variety of different pumping lemmata—for a comprehensive list of pumping or iteration lemmata we refer to [9]. A well known pumping lemma variant can be found in [8, page 70, Theorem 11.1]:

Lemma 1. *Let L be a regular language over Σ. Then, there is a constant p (depending on L) such that the following holds: If $w \in L$ and $|w| \geq p$, then there are words $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ such that $w = xyz$ and $xy^t z \in L$ for $t \geq 0$—it is then said that y can be* pumped *in w. Let $\mathtt{mpc}(L)$ denote the smallest number p satisfying the aforementioned statement.*

The above lemma can be slightly modified with the condition $|xy| \leq p$, which can be found in [10, page 119, Lemma 8], [1, page 252, Folgerung 5.4.10], and [6, page 56, Lemma 3.1]. Analogously, to $\mathtt{mpc}$ one defines $\mathtt{mpl}(L)$, as the smallest number p satisfying the statement of the modified pumping lemma.

Recently, pumping lemmata were considered in [2], where besides some simple facts such as

1. $\mathtt{mpc}(L) = 0$ if and only if $\mathtt{mpl}(L) = 0$ if and only if $L = \emptyset$,
2. for every non-empty finite language L we have $\mathtt{mpc}(L) = \mathtt{mpl}(L) = 1 + \max\{ |w| \mid w \in L \}$,
3. $\mathtt{mpc}(L) = 1$ implies $\lambda \in L$, and
4. if $\mathtt{mpl}(L) = 1$, then L is suffix closed,[2]

[2] A language $L \subseteq \Sigma^*$ is *suffix closed* if $L = \{ x \mid yx \in L, \text{ for some } y \in \Sigma^* \}$, i.e., the word x is a member of L whenever yx is in L, for some $y \in \Sigma^*$.

also the inequalities

$$\mathtt{mpc}(L) \leq \mathtt{mpl}(L) \leq \mathtt{sc}(L)$$

and results on the operational complexity w.r.t. these minimal pumping constants were shown. The upper bound on the minimal pumping constants by the deterministic state complexity is obvious. Moreover, in [2] it was also proven that for three natural numbers p_1, p_2, and p_3 with $1 \leq p_1 \leq p_2 \leq p_3$, there is a regular language L such that $\mathtt{mpc}(L) = p_1$, $\mathtt{mpl}(L) = p_2$, and $\mathtt{sc}(L) = p_3$. The witness language to prove this result is in almost all cases, except for $p_2 = p_3$,

$$L = b^{p_1-1}(a^{p_2-p_1+1})^* + c_1^* + c_2^* + \cdots + c_{p_3-p_2-1}^*,$$

while for the remaining case a unary language is given. Hence, L is a language over an alphabet of *growing size*. We improve on this result, showing that already a *binary* language can be used. Moreover, we also fix a simple flaw[3] on the size of the automaton in case $p_1 = p_2 = 1$ and $p_2 < p_3$ in the original proof given in [2].

Theorem 2. *Let p_1, p_2, and p_3 be three natural numbers with $1 \leq p_1 \leq p_2 \leq p_3$. Then, there is a regular language L over a* binary alphabet *such that $\mathtt{mpc}(L) = p_1$, $\mathtt{mpl}(L) = p_2$, and $\mathtt{sc}(L) = p_3$.*

Proof. First we define some useful languages. For $k \geq 1$ let

$$B_k^{(+)} = \begin{cases} b^+(a^*b^*)^{(k-1)/2}, & \text{if } k \text{ is odd,} \\ b^+(a^*b^*)^{(k-2)/2}a^*, & \text{if } k \text{ is even,} \end{cases}$$

and

$$B_k^{(*)} = \begin{cases} b^*(a^*b^*)^{(k-1)/2}, & \text{if } k \text{ is odd,} \\ b^*(a^*b^*)^{(k-2)/2}a^*, & \text{if } k \text{ is even,} \end{cases}$$

be languages over the alphabet $\Sigma = \{a,b\}$. Observe that in all cases there are $k-1$ alternations between the blocks. Thus, e.g., $B_3^{(*)} = b^*a^*b^*$ and $B_4^{(+)} = b^+a^*b^*a^*$. In case $k = 0$ the languages $B_k^{(+)}$ and $B_k^{(*)}$ are set to $\emptyset$. Observe that $B_k^{(+)} + \lambda$ is not equal to $B_k^{(*)}$.

Now we are ready for the proof. We distinguish whether $p_2 = 1$ (this implies that $p_1 = p_2 = 1$) or $p_2 = p_3$ (which implies $p_1 \leq p_2 = p_3$) or $p_2 \notin \{1, p_3\}$.

1. Case $p_1 = p_2 = 1$. For $p_3 = 1,2$ we simply use the DFAs accepting the languages Σ^*, a^*, respectively, for $\Sigma = \{a,b\}$ being the input alphabet of those automata. For $p_3 \geq 3$ we observe that the languages $B_{p_3-1}^{(*)}$ fulfill $\mathtt{mpc}(B_{p_3-1}^{(*)}) = \mathtt{mpl}(B_{p_3-1}^{(*)}) = p_1 = p_2 = 1$ since each accepted word can be pumped by its first letter. Additionally those languages are accepted by the DFA A shown in Figure 1—the non-accepting sink state is not shown. It is not hard to see that for each state of A

[3] For $1 \leq p_1 \leq p_2 \leq p_3$ let

$$L = b^{p_1-1}(a^{p_2-p_1+1})^* + c_1^* + c_2^* + \cdots c_{p_3-p_2-1}^*,$$

over the alphabet $\{a,b\} \cup \{c_i \mid 1 \leq i \leq p_3 - p_2 - 1\}$. For $p_1 = p_2 = 1$ consider the above given language. In case $p_3 = 2$ we get the language $L = a^*$ over the alphabet $\{a,b\}$, which requires a minimal DFA with 2 states and in case $p_3 \geq 3$ we have $L = a^* + c_1^* + c_2^* + \cdots + c_{p_3-2}^*$ over the alphabet $\{a,b\} \cup \{c_i \mid 1 \leq i \leq p_3 - 2\}$. Note that $p_3 - 2 \geq 1$ since $p_3 \geq 3$ and therefore the latter set in the union of the alphabet letters is non-empty. Thus, the minimal DFA accepting the language L has $p_3 + 1$ states, which are responsible for the Myhill-Nerode equivalence classes $[\lambda] = \{\lambda\}$, $[a] = a^+$, $[c_1] = c_1^+$, $[c_2] = c_2^+$, $\ldots$, $[c_{p_3-2}] = c_{p_3-2}^+$, and finally the equivalence class $[b] = \{w \mid w \in b^+ \text{ or } w \text{ contains at least two different letters}\}$. Observe, that all equivalence classes are accepting, except the class $[b]$, which represents the non-accepting sink state. Hence in case $p_1 = p_2 = 1$ and $p_2 < p_3$ the statement on the number of states of the minimal DFA accepting the language L presented in [2] is off by one state. The claims on the minimal pumping constants mpc and mpl for L are correct. Note that the case $p_3 = 1$ is shown in [2] with the help of a unary language.

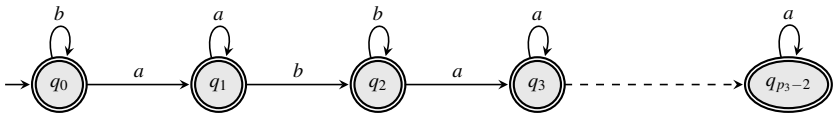


Figure 1: The automaton A for $p_1 = p_2 = 1$ and $p_3 - 1$ even, where the non-accepting sink state q_{p_3-1} and all transitions to it are not shown. Recall, that the letter on the transition to q_{p_3-2} depend on the parity of $p_3 - 2$.

there is a unique shortest word that maps the state onto the non-accepting state. Therefore we have that A is minimal and $\mathtt{sc}(B^{(*)}_{p_3-1}) = \mathtt{sc}(A) = p_3$.

2. Case $p_1 \leq p_2 = p_3$. In this case we define the unary DFA

$$A = (\{q_0, q_1, \ldots, q_{p_3-1}\}, \{a\}, \cdot_A, q_0, \{q_{p_1-1}\}),$$

with $q_i \cdot_A a = q_{i+1 \bmod p_3}$, for $0 \leq i \leq p_3 - 1$. By inspecting Figure 2 which shows A it is not hard to see that $L(A) = \{ a^{p_2 \cdot i + p_1 - 1} \mid i \geq 0 \}$ and that A is already minimal; thus $\mathtt{sc}(L(A)) = p_3$. So every

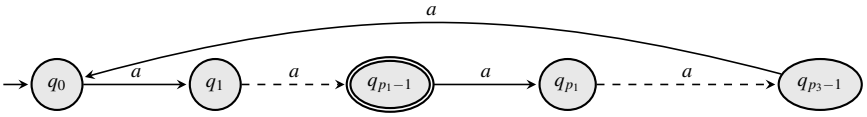


Figure 2: The unary automaton A for $p_1 < p_2 = p_3$.

word in the language $L(A)$ that has length greater or equal p_1 contains the sub-word a^{p_2} which implies that it is pumpable. On the other hand the word a^{p_1-1} cannot be pumped since it is the shortest accepting word; hence it cannot be shortened by pumping. Therefore $\mathtt{mpc}(L(A)) = p_1$ and $\mathtt{mpl}(L(A)) = p_2$.

3. Case $p_2 \notin \{1, p_3\}$. We define the language

$$L = b^{p_1-1}(a^{p_2-p_1+1})^*(B^{(+)}_{p_3-p_2-1} + \lambda).$$

This language is accepted by the DFA shown in Figure 3; again the non-accepting sink state is not shown. Observe that each state q_i, for $i \in \{0, 1, \ldots, p_2 - 1\} \setminus \{p_1 - 1\}$, is only mapped by one letter onto a state that is unequal to the sink state while this is *not* true for each state q_i, for $i \in \{p_2, p_2 + 1, \ldots, p_3 - 3, p_1 - 1\}$. Then one can easily prove that this DFA is minimal. Thus, the automaton A has p_3 states. Further we observe that the word b^{p_1-1} is in L but it cannot be pumped since no shorter word is in L. Therefore, $\mathtt{mpc}(L) \geq p_1$. Additionally we observe that $w \in b^{p_1-1}(a^{p_2-p_1+1})^+$ is a word in L which is only pumpable by $a^{p_2-p_1+1}$. Since the shortest prefix of w that ends with $a^{p_2-p_1+1}$ has length p_2 we obtain that $\mathtt{mpl}(L) \geq p_2$. Clearly we can pump all words in $b^{p_1-1}(a^{p_2-p_1+1})^+ B^{(+)}_{p_3-p_2-1}$ in the same way which implies that none of these words has an impact on $\mathtt{mpc}(L)$ and $\mathtt{mpl}(L)$. Last we see that all words in $b^{p_1-1}B^{(+)}_{p_3-p_2-1}$ can be pumped by their first letter or by their (p_1+1)th letter, respectively, for $p_1 = p_2$ and $p_1 < p_2$. So we obtain that all words in L which have length at least p_1 can be pumped by a sub-word in their prefix of length at most p_2. Thus, we have $\mathtt{mpc}(L) = p_1$ and $\mathtt{mpl}(L) = p_2$.

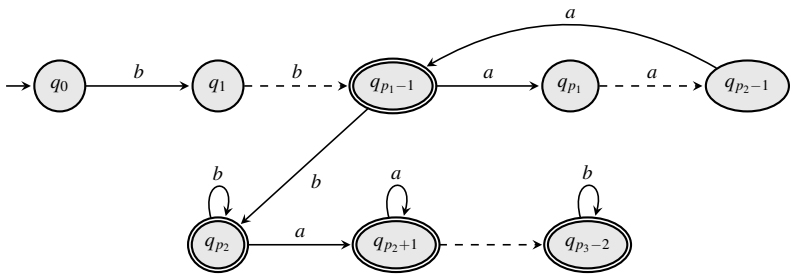


Figure 3: The automaton A for the language L in case $p_3 - p_2 - 1$ is odd, where the non-accepting sink state q_{p_3-1} and all transitions to it are not shown. In case $p_3 - p_2 - 1$ is even the lower sub-chain of states looks similar by alternatively reading a'a and b's, has appropriate self-loops on the states, and end with the letter a.

This completes the construction and proves the stated claim for languages over a binary alphabet. □

The previous theorem is best possible w.r.t. the alphabet size, because for unary languages there are infinitely many combinations of minimal pumping constants like, e.g., $\mathtt{mpc}(L) = \mathtt{mpl}(L) = 1$ and $\mathtt{sc}(L) \geq 2$, which cannot be achieved by any unary language L. This is due to the fact that if $\mathtt{mpl}(L) = 1$, then the language L is suffix-closed, and $\{a\}^*$ is the only suffix-closed unary language. It is not hard to prove that Theorem 2 is also valid if the nondeterministic state complexity instead of the deterministic state complexity is considered.

3 Results on Sub-Word Pumping

Let us first introduce a pumping lemma which is a straight forward generalization of Lemma 1 with the additional $|xy| \leq p$ condition. The lemma can be found in [11, page 49, Theorem 3.10] and reads as follows— roughly speaking, this pumping lemma allows pumping of sub-words, whose length is large enough, at any position of the considered word; hence we sometimes speak of *sub-word pumping*.

Lemma 3. *Let L be a regular language over Σ. Then there is a constant p (depending on L) such that the following holds: If $\tilde{w} = uwv \in L$ and $|w| \geq p$, where u and v are any (possibly empty) words, then there are words $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ such that $w = xyz$, $|xy| \leq p$, and $uxy^tzv \in L$ for $t \geq 0$.*

Similarly as for the aforementioned pumping lemmata, one can define the minimal pumping constant $\mathtt{mps}(L)$, for a regular language, as the smallest number p that satisfies the condition of Lemma 3 when considering L. Observe, that the condition of the lemma requires that any sub-word that is long enough can be pumped.

3.1 Comparing mps to Other Minimal Pumping Constants

We first prove some basic properties:

Lemma 4. *Let L be a regular language over Σ. Then*

- $\mathtt{mps}(L) = 0$ *if and only if* $L = \emptyset$, *and*
- $\mathtt{mps}(L) = 1$, *implies that L is prefix- and suffix-closed.*[4]

[4]Moreover, $\mathtt{mps}(L) = 1$, also implies that L is factor-closed. A regular language L is *factor-closed* if L contains all factors of all words $w \in L$. We call $w_{i_1}w_{i_2}\ldots w_{i_k}$ a *factor* of the word $w_1w_2\ldots w_n$ if $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ are natural numbers.

Proof. First we observe that there are no words $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ such that $|xy| \leq 0$. This implies directly that the statement of Lemma 3 is fulfilled for $p = 0$ and the language L if and only if $L = \emptyset$. Next we have that $\mathtt{mps}(L) = 1$ implies that for all w with $|w| \geq 1$ and all words $u, v \in \Sigma^*$ such that $\tilde{w} = uwv \in L$ there are words $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ such that $w = xyz$, $|xy| \leq 1$, and $uxy^t zv \in L$ for $t \geq 0$. In especially this holds for $w \in \Sigma$ which implies that $y = w$. Since $uxy^0zv = uxzv \in L$ for all letters $y = w \in \Sigma$ and all (possibly empty) words u and v we obtain that each word $\tilde{w}$ of L can be pumped by each of its letters, i.e, by each letter of each prefix and each suffix of $\tilde{w}$. Hence, L is prefix- and suffix-closed. □

Next we want to compare mps with the other minimal pumping constants considered in [2]. We find the following situation—similarly as in Theorem 2 the nondeterministic state complexity is also an upper bound:

Theorem 5. *Let L be a regular language L over Σ. Then $\mathtt{mpc}(L) \leq \mathtt{mpl}(L) \leq \mathtt{mps}(L) \leq \mathtt{sc}(L)$.*

Proof. It suffices to show $\mathtt{mpl}(L) \leq \mathtt{mps}(L) \leq \mathtt{sc}(L)$. For the first inequality observe that if we set $u = v = \lambda$ in Lemma 3 we obtain statement of Lemma 1 with the additional length condition $|xy| \leq p$, which implies that $\mathtt{mpl}(L) \leq \mathtt{mps}(L)$. Finally, the $\mathtt{sc}(L)$ upper bound is immediate by the proof of the lemma given in [11, page 49, Theorem 3.10]. □

Now the question arises whether we can come up with a similar result as stated in Theorem 2, but now also taking the minimal pumping constant w.r.t. Lemma 3 into account. The following Theorem will be very useful for this endeavor; a similar statement was shown in [5] for the minimal pumping constant w.r.t. Jaffe's pumping lemma [7], a pumping lemma that is necessary and sufficient for regular languages.

Theorem 6. *Let $A = (Q, \Sigma, \cdot_A, q_0, F)$ be a minimal DFA, state $q \in Q$, and letter $a \in \Sigma$. Define the finite automaton $B = (Q, \Sigma, \cdot_B, q_0, F)$ with the transition function $\cdot_B$ that is equal to the transition function of $\cdot_A$, except for the state q and the letter a, where $q \cdot_B a = q$. Then, $K(L(B)) \leq K(L(A))$ for $K \in \{\mathtt{mpc}, \mathtt{mpl}, \mathtt{mps}\}$.*

Proof. Obviously we have that in each word of the form $w = xaz$ with $q_0 \cdot_B x = q$ the $(|x|+1)$st letter can be pumped, because by construction

$$w = xazv \in L(B) \quad \text{if and only if} \quad xa^t zv \in L(B),$$

for all $t \geq 0$ and each $v \in \Sigma^*$. On the other hand the change of the a-transition of q does not affect all other words not satisfying the above property. On these words the pumping is that of the pumping induced by the device A. Thus, we conclude that the three mentioned minimal pumping constants for the language $L(B)$ are bounded by the according ones of A. □

Observe, that the statement of Lemma 3 for the constant n can also be understood as follows: for each word $\tilde{w}$ in L and each sub-word w of $\tilde{w}$ with length at least n there is a sub-word y of w such that y can be pumped in $\tilde{w}$. We will use this alternative version of Lemma 3 in the lemmata to come without further notice.

Theorem 7. *Let p_1, p_2, p_3, and p_4 be four natural numbers with $1 \leq p_1 \leq p_2 \leq p_3 \leq p_4$. Then, there is a regular language L over a* quinary alphabet *such that $\mathtt{mpc}(L) = p_1$, $\mathtt{mpl}(L) = p_2$, $\mathtt{mps}(L) = p_3$, and $\mathtt{sc}(L) = p_4$ holds.*

Proof. By taking an intense look at the constructions shown in the proof of Theorem 2 we observe that $\mathtt{mpl}(L) = \mathtt{mps}(L)$ holds for all used languages L. Therefore we safely assume for the rest of the proof that $p_2 < p_3$. On the other hand we distinguish for the proof whether $p_3 \leq p_4 - 1$ or $p_3 = p_4$. In the former case we additionally differ between $p_1 = 1$ or $p_1 \geq 2$. Since the constructions in all cases are adaptions of the case $p_3 \leq p_4 - 1$ and $p_1 \geq 2$ we give all constructions next.

We define the automaton $A = (\{q_0, q_1, \ldots, q_{p_4-1}\}, \{a,b,c,d,e\}, \cdot, q_0, \{q_{p_1-1}\} \cup F)$ with the state set $F = \{\, q_i \mid p_3 \leq i \leq p_4 - 2 \,\}$, if $p_1 = 1$, and $F = \{\, q_i \mid p_3 - 1 \leq i \leq p_4 - 2 \,\}$, otherwise. The transition function of A depends on the relation of p_3 and p_4. For $p_3 = p_4$ we set

$$\begin{aligned}
q_{2i} \cdot a &= q_{2i+1}, && \text{for } 0 \leq i \leq (p_3 - 2) \div 2,\\
q_{2i+1} \cdot c &= q_{2i+2}, && \text{for } 0 \leq i \leq (p_3 - 3) \div 2,\\
q_i \cdot b &= q_{i-1}, && \text{for } 1 \leq i \leq p_3 - 1,\\
q_i \cdot d &= q_{i+1 \bmod p_2}, && \text{for } 0 \leq i \leq p_2 - 1.
\end{aligned}$$

On the other hand we set for $p_3 \leq p_4 - 1$ and $p_1 \geq 2$,

$$\begin{aligned}
q_{2i} \cdot a &= q_{2i+1}, && \text{for } 0 \leq i \leq (p_3 - 3) \div 2,\\
q_{2i+1} \cdot a &= q_{2i+1}, && \text{for } 0 \leq i \leq (p_3 - 3) \div 2,\\
q_i \cdot b &= q_{i-1}, && \text{for } 1 \leq i \leq p_3 - 2,\\
q_{2i-1} \cdot c &= q_{2i}, && \text{for } 1 \leq i \leq (p_3 - 2) \div 2,\\
q_{2i} \cdot c &= q_{2i}, && \text{for } 0 \leq i \leq (p_3 - 2) \div 2,\\
q_{p_3+2i-1} \cdot c &= q_{p_3+2i}, && \text{for } 0 \leq i \leq (p_4 - p_3 - 1)/2 - 1,\\
q_{p_3+2i} \cdot c &= q_{p_3+2i}, && \text{for } 0 \leq i \leq (p_4 - p_3 - 1)/2 - 1,\\
q_i \cdot d &= q_{i+1 \bmod p_2}, && \text{for } 0 \leq i \leq p_2 - 1,\\
q_0 \cdot e &= q_{p_3-1}, &&\\
q_{p_3+2i-1} \cdot e &= q_{p_3+2i-1}, && \text{for } 0 \leq i \leq (p_4 - p_3 - 1)/2 - 1,\\
q_{p_3+2i} \cdot e &= q_{p_3+2i+1}, && \text{for } 0 \leq i \leq (p_4 - p_3 - 1)/2 - 1.
\end{aligned}$$

For $p_3 \leq p_4 - 1$ and $p_1 = 1$ we elongate the chain of states which are reachable by applying words from $\{a,c\}^*$ to q_0 by setting

$$\begin{aligned}
q_{2i} \cdot a &= q_{2i+1}, && \text{for } 0 \leq i \leq (p_3 - 2) \div 2\\
q_{2i+1} \cdot a &= q_{2i+1}, && \text{for } 0 \leq i \leq (p_3 - 2) \div 2,\\
q_i \cdot b &= q_{i-1}, && \text{for } 1 \leq i \leq p_3 - 1\\
q_{2i-1} \cdot c &= q_{2i}, && \text{for } 1 \leq i \leq (p_3 - 1) \div 2,\\
q_{2i} \cdot c &= q_{2i}, && \text{for } 0 \leq 0 \leq (p_3 - 1) \div 2,\\
q_{p_3+2i} \cdot c &= q_{p_3+2i+1}, && \text{for } 0 \leq i \leq (p_4 - p_3 - 1)/2 - 1,\\
q_{p_3+2i-1} \cdot c &= q_{p_3+2i-1}, && \text{for } 1 \leq i \leq (p_4 - p_3 - 1)/2 - 1,\\
q_i \cdot d &= q_{i+1 \bmod p_2}, && \text{for } 0 \leq i \leq p_2 - 1,\\
q_0 \cdot e &= q_{p_3}, &&\\
q_{p_3+2i} \cdot e &= q_{p_3+2i}, && \text{for } 0 \leq i \leq (p_4 - p_3 - 1)/2 - 1,\\
q_{p_3+2i-1} \cdot e &= q_{p_3+2i}, && \text{for } 1 \leq i \leq (p_4 - p_3 - 1)/2 - 1.
\end{aligned}$$

Additionally to the previously explicitly given transitions we set all other transitions to be transitions to the non-accepting sink state q_{p_4-1} for $p_3 \leq p_4 - 1$ and for $p_3 = p_4$ we set them to be self-loops. The automaton A is depicted in Figure 4 for the case $p_3 \leq p_4 - 1$, $p_1 \geq 2$ (on top), if $p_3 \leq p_4 - 1$, $p_1 = 1$ (in the middle) and for the case $p_3 = p_4$ (on the bottom). We will use small claims for making it easier to

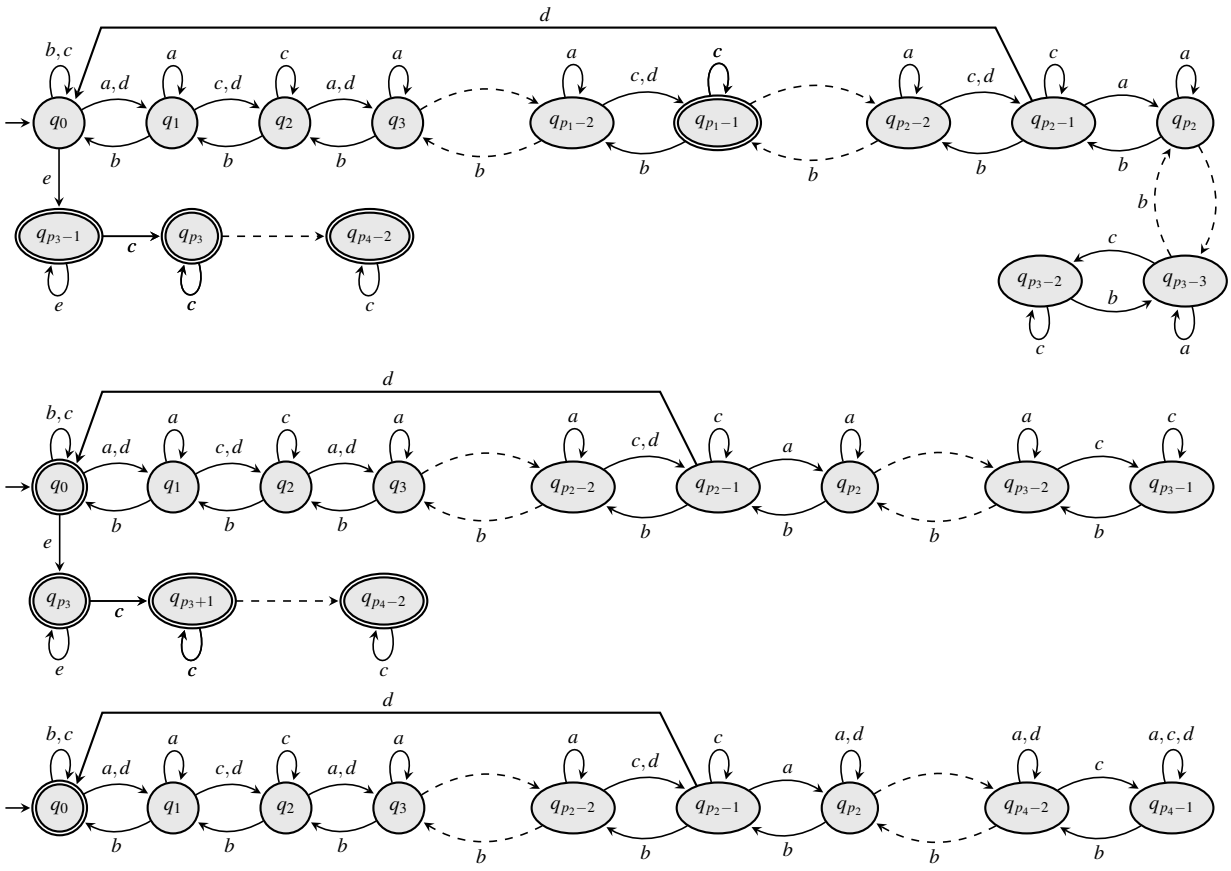


Figure 4: The automaton A for the case $p_3 \leq p_4 - 1$, $p_1 \geq 2$ (on top), if $p_3 \leq p_4 - 1$, $p_1 = 1$ (in the middle) and for the case $p_3 = p_4$ (on the bottom). For the first two cases the state q_{p_4-1} is a non-accepting sink state and all not shown transitions are mappings onto q_{p_4-1}. In the case $p_3 = p_4$ the letter e is not needed. Recall, that the a-, c-, and e-transitions in all cases depend on the parity of $p_3 - 2$ and $p_4 - 2$, respectively.

prove that the language $L := L(A)$ fulfills the requested properties.

Claim 1. *The automaton A is minimal.*

Proof. We observe that for all states in $S_1 := \{q_0, q_1, \ldots, q_{p_1-2}\}$ there is a unique shortest word in $\{a,c\}^*$ mapping the state onto q_{p_1-1}. The analogue is true for the states in $S_2 := \{q_{p_1-1}, q_{p_1}, \ldots, q_{p_3-2}\}$ and the set $\{b\}^*$. Therefore the above mentioned states cannot contain a pair of equivalent states. Additionally for all states $S_3 := \{q_{p_3}, q_{p_3+1}, \ldots, q_{p_4-2}, q_0\}$ there is a unique shortest word in $\{c,e\}^*$ mapping the state onto the state q_{p_4-2} which implies S_3 cannot contain equivalent states. Since $S_1 \cdot b^{p_1} = \{q_0\}$ and $S_3 \cdot b^{p_1} = \{q_{p_4-1}\}$ we obtain that there are no states in $S_1 \cup S_2 \cup S_3 \cup \{q_{p_4-1}\}$ which are equivalent. Indeed this directly implies that A is minimal. □

Claim 2. *We have* $\mathtt{mpl}(L) = p_2$.

Proof. Due to the fact that $L \cap \{d\}^* = (\{d\}^{p_2})^*\{d\}^{p_1-1}$ we have that the word $d^{p_2+p_1-1}$ is only pumpable by the sub-word d^{p_2} and no shorter sub-word. Indeed this implies that $\mathtt{mpl}(L) \geq p_2$. We will show that each word $\tilde{w} \in L$ of length at least p_1 is pumpable by a sub-word of its p_2-prefix. Therefore we distinguish between the several beginnings of $\tilde{w}$:

- The first letter of $\tilde{w}$ is an a or a d. Here we observe that either $\tilde{w}$ contains one of the words ab, cb, db, aa or cc in its p_2-prefix or its p_2-prefix w_1 is from $\{a,c,d\}^{p_2}$ such that $q_0 \cdot w_1 = q_0$.

  If $\tilde{w}$ contains one of the words ab, cb, db, aa or cc in its p_2-prefix then $\tilde{w}$ can be pumped by the sub-words ab, cb, db, a and c, respectively.

  If $\tilde{w}$ has a p_2-prefix w_1 which is from $\{a,b,c\}^{p_2}$ such that $q_0 \cdot w_1 = q_0$ then we can pump $\tilde{w}$ by w_1 since $q_0 \cdot w_1^i = q_0$ for all $i \geq 0$.
- The word $\tilde{w}$ starts with the letter b or c. It is obvious that $\tilde{w}$ is pumpable by its first letter.
- If the word $\tilde{w}$ has e as its first letter we observe that $\tilde{w} \in \{e,c\}^*$. For $p_2 = 1$ we can pump $\tilde{w}$ by its first letter since $q_0 \cdot c = q_0$ and $q_0 \cdot e^i = q_{p_3-1}$ for all $i \geq 1$, which are both accepting states. For $p_2 \geq 2$ we can pump $\tilde{w}$ by its second letter since $q_{p_3} \cdot c^i = q_{p_3}$ and $q_{p_3-1} \cdot e^i = q_{p_3-1}$ for all $i \geq 0$.

□

Claim 3. *We have* $\mathtt{mpc}(L) = p_1$.

Proof. Since we have shown that each word of length at least p_1 is pumpable by its p_2-prefix it remains to observe that the word d^{p_1-1} is not pumpable since $L \cap \{d\}^* = (\{d\}^{p_2})^*\{d\}^{p_1-1}$. □

Claim 4. *We have* $\mathtt{mps}(L) = p_3$.

Proof. Observe that for $p_1 = 1$ and $p_3 \leq p_4 - 1$ the chain of non-sink states which are reachable from the initial state in A by applying a word in $\{a,c\}^*$ is exactly one state longer as for $p_1 \geq 2$ and $p_3 \leq p_4 - 1$. Therefore we have that $\tilde{w} = (ac)^{(p_3-2 \div 2)}a^{p_3-2 \bmod 2}b^{p_3-2}e$ is not pumpable by any sub-word of $w = b^{p_3-2}e$ for $p_1 \geq 2$ and $\tilde{w} = (ac)^{(p_3-1 \div 2)}a^{p_3-1 \bmod 2}b^{p_3-1}$ is not pumpable by any sub-word of $w = b^{p_3-1}$ for $p_1 = 1$ which implies that $\mathtt{mps}(L) \geq p_3$. We now distinguish between all possible words $w \in \Sigma^*$ with $|w| = p_3$ and the words $\tilde{w}$ which can contain them to give a sub-word y of w such that $\tilde{w}$ is pumpable by y:

- If w contains aa or cc then $\tilde{w}$ can be pumped by $y = a$ and c, respectively.
- In the case w contains a sub-word in $\{\, xb \mid x \in \{a,c,d\}\,\}$ then we can pump $\tilde{w}$ by $y = x$ if x induces a self-loop for the according state or by $y = b$ if b from xb induces a self-loop on the according state or by xb otherwise. The last way of pumping is possible since xb induces a self-loop on the according state.
- The case that w contains a sub-word from $\{bx \mid x \in \{a,c,d\}\}$ can be treated similarly as above.
- If w contains a sub-word y from $\{a,c,d\}^*$ with length p_2 such that $\tilde{w} = uxyzv$ and $w = xyz$ for words $u,x,z,v \in \Sigma^*$, $q_0 \cdot ux \in \{q_0, q_1, \ldots, q_{p_2}\}$ and $q_0 \cdot uxy = q_0 \cdot ux$. Clearly $\tilde{w}$ can be pumped by y.
- The word w contains the letter $b = y$ such that $\tilde{w} = uxyzv$ and $w = xyz$ for words $u,x,z,v \in \Sigma^*$, and $q_0 \cdot ux = q_0$. Then $\tilde{w}$ can be pumped by $y = b$ because $q_0 \cdot uxy = q_0 \cdot y^i = q_0 \cdot b^i = q_0$ for all $i \geq 0$.
- If the word w contains the sub-word ec or ee then we can pump $\tilde{w}$ by $y = c$ or $y = e$, respectively.

It remains to observe that w has to contain one of the previously mentioned sub-words. Therefore we study how long the longest prefix w' of w in $\{a,b,c,d\}^*$ can be such that none of the above-mentioned sub-words are contained. Afterwards we elongate this prefix by a word in Σ^*.

First one may understand that for any given state q of A the longest word w' in $\{a,b,c,d\}^*$, that cannot be decomposed into $w' = xyz$ for words $x, y, z \in \Sigma^*$ such that $|y| \geq 1$ and $q \cdot x = q \cdot xy$, has length at most $p_3 - 1$ for $p_3 = p_4$ and length at most $p_3 - 2$ for $p_3 \leq p_4 - 1$. Roughly speaking this can be seen by observing that the longest such word has to map the state q onto each of the states $q_0, q_1, \ldots, q_{p_3-1}$ for $p_3 = p_4$ and onto each of the states $q_0, q_1, \ldots, q_{p_3-2}$ for $p_3 \leq p_4 - 1$. The only possibilities to elongate such a word w' are to either violate the previously described decomposing property or to elongate w' by the letter e. Due to the construction of the automaton the word $w'e$ can only be a sub-word of a word $\tilde{w} \in L$ iff $\tilde{w} = uw'ew''v$ for $q_0 \cdot uw'e = q_{p_3}$, $w'', v \in \Sigma^*$, and $w = w'ew''$. Again the transition mapping of A implies that w'' is empty or starts with one of the letters c and e. Indeed this implies that $w = w'ew''$ either has length $|w| = |w'e| \leq p_3 - 1$ for $w'' = \lambda$ or contains one of the sub-words ee or ec and is therefore pumpable by its p_3-th letter.

One observes that if we choose w' to be not maximal it similarly that w either has length less than p_3 or it contains one of the sub-words y mentioned above such that that $\tilde{w}$ is pumpable by y. □

In conclusion we have that $\texttt{mpc}(L) = p_1$, $\texttt{mpl}(L) = p_2$, $\texttt{mps}(L) = p_3$, and $\texttt{sc}(L) = p_4$ for $p_3 \leq p_4 - 1$. Due to Theorem 6 we directly obtain for $p_3 = p_4$ that the according pumping constants have to be at most equal to the pumping constants in the case $p_3 \leq p_4 - 1$. In turn we observe that the witnesses for $\texttt{mpc}(L) \geq p_1$ and $\texttt{mpl}(L) \geq p_2$ can also applied for $p_3 = p_4$. Additionally the word $\tilde{w} = (ac)^{(p_3-1 \div 2)} a^{p_3-1 \bmod 2} b^{p_3-1}$ with $w = b^{p_3-1}$ witnesses $\texttt{mps}(L) \geq p_3$ for $p_3 = p_4$. The minimality of A can be shown similarly as for $p_3 \leq p_4 - 1$. Therefore we conclude that $\texttt{mpc}(L) = p_1$, $\texttt{mpl}(L) = p_2$, $\texttt{mps}(L) = p_3$, and $\texttt{sc}(L) = p_4$. □

3.2 Operational Complexity of Sub-Word Pumping

We study the effect of regularity preserving standard formal language operations on the minimal pumping constant w.r.t. Lemma 3 and compare them to previously obtained results [2] for the other minimal pumping constants. To this end we need some notation: let $\circ$ be a regularity preserving n-ary function on languages and $K \in \{\texttt{mpc}, \texttt{mpl}, \texttt{mps}\}$. Then, we define $g_\circ^K(k_1, k_2, \ldots, k_n)$ as the set of all numbers k such that there are regular languages $L_1, L_2, \ldots, L_n$ with $K(L_i) = k_i$, for $1 \leq i \leq n$ and $K(\circ(L_1, L_2, \ldots, L_n)) = k$. Results for some regularity preserving operations on $\texttt{mpc}$ and $\texttt{mpl}$ can be found in the comprehensive Table 1. The set of all natural numbers not including zero is denoted by $\mathbb{N}$; if zero is included, then we write $\mathbb{N}_0$ instead. The gray shaded entries in Table 1 are new results, left open results, or corrected results from [2]. We only give the proofs for two of these new results, namely Kleene star and intersection.

Let us start with the Kleene star operation. In [2] it was shown that for the Kleene star operation the following results hold:

$$g_*^{\texttt{mpc}}(n) = \{1\} \quad \text{and} \quad g_*^{\texttt{mpl}}(n) = \begin{cases} \{1\}, & \text{if } n = 0, \\ \{1, 2, \ldots, n\}, & \text{otherwise,} \end{cases}$$

for every $n \geq 0$. For the minimal pumping constant $\texttt{mps}$ a larger set of numbers is attainable as we show next.

Theorem 8. *It holds*

$$g_*^{\texttt{mps}}(n) = \begin{cases} \{1\}, & \text{if } n = 0, \\ \{1, 2, \ldots, 2n-1\}, & \text{otherwise.} \end{cases}$$

Operation	Minimal pumping constant: mpc	Minimal pumping constant: mpl	Minimal pumping constant: mps
Kleene star	$\{1\}$	$\{1\}$, if $n=0$, $\{1,2,\dots,n\}$, otherwise.	$\{1\}$, if $n=0$, $\{1,2,\dots,2n-1\}$, otherwise.
Reversal	$\{n\}$	$\{0\}$, if $n=0$, $\mathbb{N}$, otherwise.	$\{n\}$
Complement	$\{1\}$, if $n=0$, $\mathbb{N}_0\setminus\{1\}$, if $n=1$, $\mathbb{N}$, otherwise.	$\{1\}$, if $n=0$ $\mathbb{N}_0\setminus\{1\}$, if $n=1$, $\mathbb{N}$, otherwise.	$\{1\}$, if $n=0$, $\mathbb{N}_0\setminus\{1\}$, if $n=1$, $\mathbb{N}$, otherwise.
Prefix-Closure	$\{0\}$, if $n=0$, $\mathbb{N}$, otherwise.	$\{0\}$, if $n=0$, $\{1,2,\dots,n\}$, otherwise.	$\{0\}$, if $n=0$, $\{1,2,\dots,n\}$, otherwise.
Suffix-Closure	$\{0\}$, if $n=0$, $\mathbb{N}$, otherwise.	$\{0\}$, if $n=0$, $\{1\}$, if $n=1$, $\mathbb{N}$, otherwise.	$\{0\}$, if $n=0$, $\{1,2,\dots,n\}$, otherwise.
Union	$\max\{m,n\}$, if $m=0$ or $n=0$, $\{1,2,\dots,\max\{m,n\}\}$, otherwise.	$\max\{m,n\}$, if $m=0$ or $n=0$, $\{1,2,\dots,\max\{m,n\}\}$, otherwise.	$\max\{m,n\}$, if $m=0$ or $n=0$, $\{1,2,\dots,\max\{m,n\}\}$, otherwise.
Set-Subtraction	$\{0\}$, if $m=0, n\geq 0$, $\{m\}$, if $m\geq 0, n=0$, $\mathbb{N}_0\setminus\{1\}$, if $m\geq 1, n=1$, $\mathbb{N}_0$, otherwise.	$\{0\}$, if $m=0, n\geq 0$, $\{m\}$, if $m\geq 0, n=0$, $\mathbb{N}_0\setminus\{1\}$, if $m\geq 1, n=1$, $\mathbb{N}_0$, otherwise.	$\{0\}$, if $m=0, n\geq 0$, $\{m\}$, if $m\geq 0, n=0$, $\mathbb{N}_0\setminus\{1\}$, if $m\geq 1, n=1$, $\mathbb{N}_0$, otherwise.
Concatenation	$\{0\}$, if $m=0$ or $n=0$, $\{1,2,\dots,m+n-1\}$, otherwise.	$\{0\}$, if $m=0$ or $n=0$, $\{1,2,\dots,m+n-1\}$, otherwise.	$\{0\}$, if $m=0$ or $n=0$, $\{1,2,\dots,m+n-1\}$, otherwise.
Intersection	$\{0\}$, if $m=0$ or $n=0$, $\mathbb{N}_0\setminus\{2\}$, if $m=n=1$, $\mathbb{N}_0$, otherwise.	$\{0\}$, if $m=0$ or $n=0$, $\{1\}$, if $m=n=1$, $\mathbb{N}_0$, otherwise.	$\{0\}$, if $m=0$ or $n=0$, $\{1\}$, if $m=n=1$, $\mathbb{N}_0$, otherwise.
Symmetric Difference	$\max\{m,n\}$, if $m=0$ or $n=0$, $\mathbb{N}_0\setminus\{1\}$, if $m=n=1$, $\mathbb{N}_0$, if $m=n>1$, $\mathbb{N}$, otherwise.	$\max\{m,n\}$, if $m=0$ or $n=0$, $\mathbb{N}_0\setminus\{1\}$, if $m=n=1$, $\mathbb{N}_0$, if $m=n>1$, $\mathbb{N}$, otherwise.	$\max\{m,n\}$, if $m=0$ or $n=0$, $\mathbb{N}_0\setminus\{1\}$, if $m=n=1$, $\mathbb{N}_0$, if $m=n>1$, $\mathbb{N}$, otherwise.

Table 1: Results on the operational complexity of the minimal pumping constants mpc, mpl, and mps. The results for the former two minimal pumping constants are from [2]. Gray shaded entries indicate new results, previous left open results, or corrected ones. Here $\mathbb{N}$ refers to the set of all natural number *not* including zero; if zero is included we refer to this set as $\mathbb{N}_0$.

Proof. First we look at the case where $n = 0$. Afterwards we argue why, for $n \geq 1$, no value in $\mathbb{N}_0 \setminus \{1, 2, \ldots, 2n-1\}$ can be reached, and at last we define languages $L_{n,k}$ with the property that $\mathtt{mps}(L_{n,k}) = n$ and for the Kleene star of $L_{n,k}$ we have $\mathtt{mps}(L_{n,k}^*) = k$.

For $\mathtt{mps}(L_{0,k}) = n = 0$ we observe that $L_{0,k} = \emptyset$. So we have that $\mathtt{mps}(L_{0,k}) = n = 0$ implies that $k = \mathtt{mps}(L_{0,k}^*) = \mathtt{mps}(\emptyset^*) = \mathtt{mps}(\{\lambda\}) = 1$. Next we show that for any language L with $\mathtt{mps}(L) = n$ we have that $\mathtt{mps}(L^*) \leq 2n-1$. We observe that each non-empty word $\tilde{w} \in L^*$ is equal to $\tilde{w}_1\tilde{w}_2\ldots\tilde{w}_t$, for $\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_t \in L$. We know for each of those words that each of their sub-words of length n can be pumped by a sub-word of length at most n. Assume that $\mathtt{mps}(L^*) \geq 2n$ and the sub-word w of $\tilde{w}$ is a witness for that, which means there are words u and v in Σ^* such that $\tilde{w} = uwv \in L^*$ cannot be pumped by a sub-word of the $2n-1$-prefix of w. W.l.o.g. we assume that the words $\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_t \in L$ are not empty. Obviously, we have that

$$\tilde{w} = \tilde{w}_1\tilde{w}_2\ldots\tilde{w}_t = \tilde{w}_1\tilde{w}_2\ldots\tilde{w}_{i-1}w_i'ww_j'\tilde{w}_{j+1}\ldots\tilde{w}_t$$

for $w_i'ww_j' = \tilde{w}_i\tilde{w}_{i+1}\ldots\tilde{w}_{j-1}\tilde{w}_j$. We know that each sub-word of length n of $\tilde{w}_i$ and $\tilde{w}_{i+1}$ can be pumped by one of its sub-words. Especially this holds for the n-suffix of $\tilde{w}_i$. If this suffix is contained in w than $uwv = \tilde{w}$ can be pumped by that sub-word of w which contradicts the assumption that w is a witness for $\mathtt{mps}(L^*) \geq 2n$. The analogue holds true if the n-prefix of $\tilde{w}_{i+1}$ is contained in w. Additionally, if $\tilde{w}_i$ (or $\tilde{w}_{i+1}$, respectively) is completely contained in w and has length less than n, then word $uwv = \tilde{w}$ can be pumped by $\tilde{w}_i$ (or $\tilde{w}_{i+1}$, respectively). Again this contradicts the assumption that w is a witness for $\mathtt{mps}(L^*) \geq 2n$. Due to the fact that $|w| \geq 2n-1$ one of the previously described cases must occur. In conclusion we have that w cannot be a witness for $\mathtt{mps}(L^*) \geq 2n$. Therefore, $\mathtt{mps}(L^*) \leq 2n-1$.

Now we prove the reachability of the above-mentioned values for k. We distinguish the cases whether $n > k$, $n = k$, or $n < k$:

1. Case $n > k$: let $L_{n,k} = \{a^i \mid 0 \leq i \leq n-1\} \cup \{b^k\}$ which is a finite language and thus $\mathtt{mps}(L_{n,k}) = n$. Observe, that $L_{n,k}^*$ is the language of all words that contain only b-blocks with lengths that are divisible by k. Therefore the word $w = b^k$ cannot be pumped by a sub-word of length at most $k-1$ which implies that $\mathtt{mps}(L_{n,k}^*) \geq k$. Assume there is a word $w \in \{a,b\}^*$ witnessing $\mathtt{mps}(L_{n,k}^*) > k$ then there are words $u, v \in \{a,b\}^*$ such that uwv cannot be pumped by a sub-word of the k-prefix y of w. Due to the structure of $L_{n,k}^*$ we know that uwv can be pumped by a sub-word of y, if y contains an a or it contains the sub-word b^k. Since $|y| = k$ one of the conditions must be fulfilled which implies that $\mathtt{mps}(L_{n,k}) = k$.

2. Case $k = n$: Let $L = (a^n)^* = L^*$. Then $\mathtt{mps}(L) = \mathtt{mps}(L^*) = n = k$.

3. Case $n < k$: Let $L_{n,k} = (a^n)^* \cup (b^{k-n+1})^*$. We have $\mathtt{mps}(L_{n,k}) = n$, since $k-n+1 \leq (2n-1)-n+1 = n$ due to the fact that $k \in \{1, 2, \ldots, 2n-1\}$. On the other hand we have that $L_{n,k}^*$ contains all words which only contain a- and b-blocks whose length are divisible by n and $k-n+1$, respectively. Therefore the word $w = a^{n-1}b^{k-n}$ cannot be pumped by a sub-word of length $n-1+k-n = k-1$, which implies that $\mathtt{mps}(L_{n,k}^*) \geq k$. So we assume that there is a word $w \in \{a,b\}^*$ witnessing $\mathtt{mps}(L_{n,k}^*) > k$, which implies that there are words $u, v \in \{a,b\}^*$ such that uwv cannot be pumped by a sub-word of the k-prefix y of w. Since $|y| = k \geq n \geq k-n+1$ the word y must contain the sub-word a^n or b^{k-n+1}, which implies that uwv can be pumped by that sub-word of y. Since this contradicts the assumption that w is a witness for $\mathtt{mps}(L_{n,k}^*) > k$ we have that $\mathtt{mps}(L_{n,k}^*) \leq k$. In summary $\mathtt{mps}(L_{n,k}^*) = k$ as desired.

This proves the stated claim. □

For the intersection operation it was left open in [2], which numbers are reachable for the pumping constant mpl. We close this gap and show that for mpc, mpl, and mps the same set of numbers is reachable.

Theorem 9. *For $K \in \{\texttt{mpl}, \texttt{mps}\}$ we have*

$$g_{\cap}^{K}(m,n) = \begin{cases} \{0\}, & \text{if } m = 0 \text{ or } n = 0, \\ \{1\}, & \text{if } m = n = 1, \\ \mathbb{N}_0, & \text{otherwise,} \end{cases}$$

Proof. Obviously we have that $L \cap \emptyset = \emptyset \cap L = \emptyset$ for each regular language L. Assume that L, L' are regular languages with $\texttt{mpl}(L) = \texttt{mpl}(L') = 1$. Given a word $\tilde{w} \in L \cap L'$ such that $\tilde{w}$ is a witness for $\texttt{mpl}(L \cap L') \geq 2$ then $\tilde{w}$ cannot be pumped by its first letter. On the other side we know that $\tilde{w}$ can be pumped in L and in L' by its first letter since $\texttt{mpl}(L) = \texttt{mpl}(L') = 1$. This implies that each word we obtain from $\tilde{w}$ by pumping its first letter is in L and L'; therefore in $L \cap L'$. Hence, we can pump $\tilde{w}$ in $L \cap L'$ by its first letter which is a contradiction to the assumption on $\tilde{w}$. The previously shown reasoning also applies similarly for $\texttt{mps}(L) = \texttt{mps}(L') = 1$ because with this property each word in L and L' can be pumped by any of its letters. The value $k = 0$ is unreachable for $n = m = 1$ because each language L with $\texttt{mpl}(L) = 1$ or $\texttt{mps}(L) = 1$ contains the letter λ due to Lemma 4 and the remark after Lemma 1. Next we construct languages such that all values $k \geq 0$ can be achieved in the general case for m and n. Here we distinguish whether k is equal to zero, one or an odd or an even value which is at least two—notice that the construction for $k = 1$ also applies for $m = n = 1$:

- For $k = 0$ we define $L_{m,k} = \{a^{m-1}\}$ and $L_{n,k} = \{b^{n-1}\}$ which are finite languages and therefore fulfill $\texttt{mpl}(L_{m,k}) = \texttt{mps}(L_{m,k}) = m$ and $\texttt{mpl}(L_{n,k}) = \texttt{mps}(L_{n,k}) = n$. Clearly $L_{m,k} \cap L_{n,k} = \emptyset$ which provides $\texttt{mpl}(\emptyset) = \texttt{mps}(\emptyset) = 0 = k$.
- In the case $k = 1$ we define $L_{m,k} = \{a^{m-1}\} \cup \{b\}^*$ and $L_{n,k} = \{c^{n-1}\} \cup \{b\}^*$ which fulfill $\texttt{mpl}(L_{m,k}) = \texttt{mps}(L_{m,k}) = m$ and $\texttt{mpl}(L_{n,k}) = \texttt{mps}(L_{n,k}) = n$ because $a^{m-1} \in L_{m,k}$ and $c^{n-1} \in L_{n,k}$ are not pumpable by any of their sub-words. Obviously we have $L_{m,k} \cap L_{n,k} = \{b\}^*$ which suffices $\texttt{mpl}(\{b\}^*) = \texttt{mps}(\{b\}^*) = 1 = k$.
- Now we study the case where $k \geq 2$ is an even integer. If $k \geq 2$ one of the values m and n must be at least equal to two. Since the intersection of regular languages is symmetric in its arguments we assume without loss of generality that $m \geq 2$ and $n \geq 1$.

  We set $L_{m,k} = \{c^{m-1}\} \cup \{ba\}^*\{b\}\{ad\}^* \cup \{da\}^*\{d\}$ and $L_{n,k} = \{e^{n-1}\} \cup B_{k-2}^{(*)}\{d\}^*$. We observe that $c^{m-1} \in L_{m,k}$ and $e^{n-1} \in L_{n,k}$ are not pumpable which implies that $m \leq \texttt{mpl}(L_{m,k}) \leq \texttt{mps}(L_{m,k})$ and $n \leq \texttt{mpl}(L_{n,k}) \leq \texttt{mps}(L_{n,k})$. Since each word in $L_{n,k}$ is pumpable by each of its letters except e^{n-1} we obtain $n = \texttt{mpl}(L_{n,k}) = \texttt{mps}(L_{n,k})$. Further each word in $\tilde{w} \in \{ba\}^*\{b\}\{ad\}^* \cup \{da\}^*\{d\}$ is pumpable by a sub-word y of each sub-word w of $\tilde{w}$ with $|w| \geq 2$. This can be seen by looking at the different cases for the prefixes of length two of w which is done next. For the sake of simplicity we assume $|w| = 2$. We will give for each case the word y and then verify that $\tilde{w}$ can be pumped by y by distinguishing between the words $\tilde{w}$ which can contain w:
  - For $w = ab$ we can choose $y = ab$ which is observed by understanding that $\{ba\}^*\{b\}\{ad\}^* = \{b\}\{ab\}^+\{ad\}^* \cup \{b\}\{ad\}^*$.
  - For $w = ba$ we can choose $y = ba$. First let $\tilde{w} = (ba)^i b(ad)^j \in \{ba\}^*\{b\}\{ad\}^*$. If $\tilde{w} = (ba)^{i'} w (ba)^{i''} b(ad)^j$ for $i = i' + i'' + 1$ then we obtain by pumping $\tilde{w}$ *via* y a word $(ba)^{i+\ell} b(ad)^*$ for $-1 \leq \ell$. For $\tilde{w} = (ba)^i wd(ad)^{j-1}$ we obtain by pumping $\tilde{w}$ *via* y a word $(ba)^{i+\ell} b(ad)^*$ for $0 \leq \ell$ and the word $d(ad)^{j-1} = (da)^{j-1}d \in \{da\}^*\{d\}$ for $\ell = -1$.

- For $w = ad$ we can choose $y = ad$ which is easy to confirm for each word in $\{ba\}^*\{b\}\{ad\}^*$ and each word in $\{da\}^*\{d\} = \{d\}\{ad\}^*$.
- For $w = da$ we can choose $y = da$ because $\{ba\}^*\{b\}\{ad\}^* = \{ba\}^*\{b\}\{a\}\{da\}^*\{d\} \cup \{ba\}^*\{b\}$ and for the words in $\{ba\}^*\{b\}\{a\}\{da\}^*\{d\} \cup \{da\}^*\{d\}$ it is obvious that they can be pumped by $y = da$.

In conclusion each word $\tilde{w}$ in $L_{m,k}$ can be pumped by a sub-word y of every sub-word w of $\tilde{w}$ if $|w| \geq 2$. Therefore we obtain that $\texttt{mpl}(L_{m,k}) = \texttt{mps}(L_{m,k}) = m$.

We observe that $L_{m,k} \cap L_{n,k} = (\{c^{m-1}\} \cup \{ba\}^*\{b\}\{ad\}^* \cup \{da\}^*\{d\}) \cap (\{e^{n-1}\} \cup B_{k-2}^*\{d\}^*) = \{\, (ba)^i d \mid 0 \leq i \leq (k-2)/2 \,\}$ which is a finite language and therefore suffices $\texttt{mpl}(L_{m,k} \cap L_{n,k}) = \texttt{mps}(L_{m,k} \cap L_{n,k}) = k$. This is due to the fact that the longest word in this language is $\tilde{w} = (ba)^{(k-2)/2}d$ which fulfills $|\tilde{w}| = 2 \cdot (k-2)/2 + 1 = k-1$.

- In the case that $k \geq 2$ is an odd integer we adapt the language $L_{m,k}$ shown in the previous case to be equal to $\{c^{m-1}\} \cup \{ba\}^*\{bd\}^* \cup \{da\}^*\{d\}$. Indeed the property $\texttt{mpl}(L_{m,k}) = \texttt{mps}(L_{m,k}) = m$ can be proven in the same style as in the previous case. Additionally we have $L_{m,k} \cap L_{n,k} = (\{c^{m-1}\} \cup \{ba\}^*\{bd\}^* \cup \{da\}^*\{d\}) \cap (\{e^{n-1}\} \cup B_{k-2}^*\{d\}^*) = \{\, (ba)^i bd \mid 0 \leq i \leq (k-3)/2 \,\}$, which is a finite language and therefore suffices $\texttt{mpl}(L_{m,k} \cap L_{n,k}) = \texttt{mps}(L_{m,k} \cap L_{n,k}) = k$. Here the longest word in this language is $\tilde{w} = (ba)^{(k-3)/2}bd$ which fulfills $|\tilde{w}| = 2 \cdot (k-3)/2 + 2 = k-1$.

□

References

[1] W. Brauer (1984): *Automatentheorie: Eine Einführung in die Theorie endlicher Automaten*. Leitfäden und Monographien der Informatik, Teubner Stuttgart, doi:10.1007/978-3-322-92151-2. (in German).

[2] J. Dassow & I. Jecker (2022): *Operational complexity and pumping lemmas*. Acta Inform. 59, pp. 337—355, doi:10.1007/s00236-022-00431-3.

[3] H. Gruber, M. Holzer & C. Rauch (2023): *The Pumping Lemma for Regular Languages is Hard*. In B. Nagy, editor: Proceedings of the 27th International Conference on Implementation and Application of Automata, LNCS 14151, Springer, Famagusta, Cyprus. Accepted for publication.

[4] M. A. Harrison (1978): *Introduction to Formal Language Theory*. Addison-Wesley.

[5] M. Holzer & C. Rauch (2023): *On Jaffe's Pumping Lemma, Revisited*. In H. Bordihn, N. Tran & G. Vaszil, editors: Proceedings of the 25th International Conference on Descriptional Complexity of Formal Systems, LNCS 13918, Springer, Potsdam, Germany, pp. 65–78, doi:10.1007/978-3-031-34326-1_5.

[6] J. E. Hopcroft & J. D. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

[7] J. Jaffe (1978): *A necessary and sufficient pumping lemma for regular languages*. SIGACT News 10(2), pp. 48–49, doi:10.1145/990524.990528.

[8] D. C. Kozen (1997): *Automata and Computability*. Undergraduate Texts in Computer Science, Springer, doi:10.1007/978-1-4612-1844-9.

[9] A. Nijholt (1982): *YABBER—Yet Another Bibliography: Pumping Lemma's. An Annotated Bibliography of Pumping*. Bull. EATCS 17, pp. 34–53.

[10] M. O. Rabin & D. Scott (1959): *Finite Automata and Their Decision Problems*. IBM J. Res. Dev. 3, pp. 114–125, doi:10.1147/rd.32.0114.

[11] W. J. Savitch (1982): *Abstract Machines and Grammars*. Little, Brown and Company.

Reversible Two-Party Computations

Martin Kutrib and Andreas Malcher
Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,andreas.malcher}@informatik.uni-giessen.de

Deterministic synchronous systems consisting of two finite automata running in opposite directions on a shared read-only input are studied with respect to their ability to perform reversible computations, which means that the automata are also backward deterministic and, thus, are able to uniquely step the computation back and forth. We study the computational capacity of such devices and obtain on the one hand that there are regular languages that cannot be accepted by such systems. On the other hand, such systems can accept even non-semilinear languages. Since the systems communicate by sending messages, we consider also systems where the number of messages sent during a computation is restricted. We obtain a finite hierarchy with respect to the allowed amount of communication inside the reversible classes and separations to general, not necessarily reversible, classes. Finally, we study closure properties and decidability questions and obtain that the questions of emptiness, finiteness, inclusion, and equivalence are not semidecidable if a superlogarithmic amount of communication is allowed.

1 Introduction

Watson-Crick automata have been introduced in [7] as a formal model for DNA computing. The motivation for such automata comes from processes observed in nature and laboratories. Basically, the idea is to consider finite automata with two reading heads that run on either strand of a double stranded DNA-molecule. It is noted in [20] that in nature enzymes moving along DNA strands may obey the biochemical direction of the single strands of the DNA sequence. Hence, so-called $5' \to 3'$ Watson-Crick automata have been introduced in [20], which are two-head finite automata where the heads start at opposite ends of a strand and move in opposite physical directions. It is known that no additional information is encoded in the second strand provided that the complementarity relation of the double stranded sequence is one-to-one. In this case, $5' \to 3'$ Watson-Crick automata share a common input sequence.

Watson-Crick automata and $5' \to 3'$ Watson-Crick automata have intensively been investigated in the last years from different points of view. Descriptional complexity aspects of Watson-Crick automata are studied in [6]. $5' \to 3'$ Watson-Crick automata with several runs, which means that both heads are sweeping between both ends of the input, are investigated in [18] and a hierarchy with respect to the number of runs has been obtained. The aspect of the amount of communication between the two heads that is necessary in accepting computations is highlighted in [12] where $5' \to 3'$ Watson-Crick automata with restricted communication are introduced and a finite hierarchy concerning the amount of communication could be obtained. The concept of sensing heads, where one head can sense the presence of the other head, has been applied to $5' \to 3'$ Watson-Crick automata in [21, 24]. The concept of jumping automata, where the input is processed in a discontinuous way, has been introduced and investigated for $5' \to 3'$ Watson-Crick automata in [9]. Finally, the impact of replacing the underlying devices of finite automata by finite transducers or pushdown automata is studied in [23] and in [5, 22], respectively.

Another line of research in recent years is the study of reversible devices. Here, a computation is considered *reversible* if every configuration has at most one unique successor configuration and at most one

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 142–154, doi:10.4204/EPTCS.386.12

unique predecessor configuration. The study of such devices that perform logically reversible computations is motivated by Landauer's question of whether logical irreversibility is an unavoidable feature of useful computers. This question is of particular interest, since Landauer has demonstrated that whenever a physical computer throws away information about its previous state it must generate a corresponding amount of entropy that results in heat dissipation. A detailed discussion and suitable references can be found in [2]. Reversible variants of many computational models have been studied in the literature. For Turing machines the first investigations on reversible computations date back to the sixties of the last century. It is shown in the work of Lecerf [17] and Bennett [2] that it is possible for every Turing machine to construct an equivalent reversible Turing machine. Hence, every irreversible computation can be made reversible. This is no longer true if finite automata are considered. On the one hand, it is known that reversible one-way deterministic finite automata are computationally weaker than one-way deterministic finite automata in general [1] (cf. also [8]). On the other hand, two-way deterministic finite automata and reversible two-way deterministic finite automata are equally powerful [10]. Similar results are known for multihead finite automata. In case of one-way motion, the reversible variant is computationally weaker than the general model ([14]), whereas in case of two-way motion the computational power of the reversible variant and the general model coincides [19]. Several more types of devices as, for example, queue automata [16], one-way counter machines with multiple counters [15], and parallel communicating finite automata [3] have been investigated with respect to reversibility. An overview of the topic is given in [11].

The aspect of reversibility has been studied for Watson-Crick automata in [4]. One result is that every regular language can be accepted by a reversible Watson-Crick automaton. Here, it is essential that the complementarity relation of the double stranded sequence is *not* one-to-one. If the complementarity relation is one-to-one, another result of [4] gives that the computational power of reversible Watson-Crick automata and reversible two-head finite automata ([14]) coincides. In this paper, we study $5' \to 3'$ Watson-Crick automata having a one-to-one complementarity relation and to differentiate the notation from other variants we will call the devices in question *two-party Watson-Crick systems*. This paper can be seen as a continuation of [12] where communication restricted two-party Watson-Crick systems are introduced and a strict four-level hierarchy depending on the number of messages sent was established, where the levels are given by $O(1)$, $O(\log(n))$, $O(\sqrt{n})$, and $O(n)$ messages allowed. Moreover, it could be shown that the questions of emptiness, finiteness, inclusion, and equivalence are not semidecidable, that is, not recursively enumerable, even if the communication is reduced to a limit $O(\log(n) \cdot \log(\log(n)))$. Here, we complement these results. After defining the model and giving two illustrating examples in Section 2 we show in Section 3 that there are regular languages which can not be accepted by any reversible two-party Watson-Crick systems with any amount of communication. This is in strong contrast to general two-party Watson-Crick systems where no communication is necessary to accept regular languages. This result can be used in Section 4 in which closure properties are investigated. It turns out that reversible two-party Watson-Crick systems are closed under complementation and reversal, whereas they are not closed under union, intersection, intersection with regular languages, concatenation, iteration, length-preserving homomorphism, and inverse homomorphism. In Section 5, we can extend the strict four-level hierarchy depending on the number of messages sent from [12] to reversible two-party Watson-Crick systems. Moreover, we obtain that for every level the reversible systems are computationally weaker than the general systems. Finally, we discuss in Section 6 decidability questions. In a first step, we show that the questions of emptiness, finiteness, inclusion, and equivalence are not semidecidable for reversible two-party Watson-Crick systems essentially disregarding the number of messages communicated. In a second step, we refine the argumentation and apply and adapt a result from [14] which enables us to show that the questions of emptiness, finiteness, inclusion, and

equivalence are not semidecidable for reversible two-party Watson-Crick systems even if the number of messages allowed is bounded by $O(\log(n)\cdot\log(\log(n)))$.

2 Definitions and Preliminaries

We denote the set of nonnegative integers by $\mathbb{N}$. We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of a word w is denoted by w^R and for the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions.

A two-party Watson-Crick system is a device of two finite automata working independently and in opposite directions on a common read-only input data. The automata communicate by broadcasting messages. The transition function of a single automaton depends on its current state, the currently scanned input symbol, and the message currently received from the other automaton. Both automata work synchronously and the messages are delivered instantly. Whenever the transition function of (at least) one of the single automata is undefined the whole systems halts. The input is accepted if at least one of the automata is in an accepting state. A formal definition is as follows.

A *deterministic two-party Watson-Crick system* (DPWK) is a construct $\mathscr{A} = \langle \Sigma, M, \rhd, \lhd, A_1, A_2\rangle$, where Σ is the finite set of *input symbols*, M is the set of possible *messages*, $\rhd \notin \Sigma$ and $\lhd \notin \Sigma$ are the *left and right endmarkers*, and each $A_i = \langle Q_i, \Sigma, \delta_i, \mu_i, q_{0,i}, F_i\rangle$, $i \in \{1,2\}$, is basically a *deterministic finite automaton* with *state set* Q_i, *initial state* $q_{0,i} \in Q_i$, and set of *accepting states* $F_i \subseteq Q_i$. Additionally, each A_i has a *broadcast function* $\mu_i : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \to M \cup \{\bot\}$ which determines the message *to be sent*, where $\bot \notin M$ means *nothing to send*, and a *(partial) transition function* $\delta_i : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \times (M \cup \{\bot\}) \to Q_i \times \{0,+\}$, where $+$ means to move the head one square and 0 means to keep the head on the current square.

The automata A_1 and A_2 are called *components* of the system $\mathscr{A}$, where the so-called *upper* component A_1 starts at the left end of the input and moves from left to right, and the *lower* component A_2 starts at the right end of the input and moves from right to left. A *configuration* of $\mathscr{A}$ is represented by a string $\rhd v_1 \overrightarrow{p} x v_2 y \underleftarrow{q} v_3 \lhd$, where $v_1 x v_2 y v_3$ is the input and it is understood that component A_1 is in state p with its head scanning symbol x, and component A_2 is in state q with its head scanning symbol y. System $\mathscr{A}$ starts with component A_1 in its initial state scanning the left endmarker and component A_2 in its initial state scanning the right endmarker. So, for input $w \in \Sigma^*$, the initial configuration is $\overrightarrow{q_{0,1}} \rhd w \lhd \underleftarrow{q_{0,2}}$. A computation of $\mathscr{A}$ is a sequence of configurations beginning with an initial configuration. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $w = a_1 a_2 \cdots a_n$ be the input, $a_0 = \rhd$, and $a_{n+1} = \lhd$, then we set

$$a_0 \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{n+1} \vdash a_0 \cdots a_{i'-1} \overrightarrow{p_1} a_{i'} \cdots a_{j'} \underleftarrow{q_1} a_{j'+1} \cdots a_{n+1},$$

for $0 \le i, j \le n+1$, iff $\delta_1(p, a_i, \mu_2(q, a_j)) = (p_1, d_1)$ and $\delta_2(q, a_j, \mu_1(p, a_i)) = (q_1, d_2)$, $i' = i + d_1$ and $j' = j - d_2$. As usual we define the reflexive, transitive closure of $\vdash$ by $\vdash^*$.

A computation *halts* when the successor configuration is not defined for the current configuration. This may happen when the transition function of one component is not defined. The language $L(\mathscr{A})$ accepted by a DPWK $\mathscr{A}$ is the set of inputs $w \in \Sigma^*$ such that there is some computation beginning with the initial configuration for w and halting with at least one component being in an accepting state.

Now we turn to *reversible* two-party Watson-Crick systems. Basically, reversibility is meant with respect to the possibility of stepping the computation back and forth. So, the system has also to be backward deterministic. That is, any configuration must have at most one predecessor which, in addition, is

computable by a two-party Watson-Crick system. In particular for the read-only input tape, the machines reread the input symbol which they have been read in a preceding forward computation step. Therefore, for reverse computation steps the head of the upper component is either moved to the *left* or stays stationary, whereas the head of the lower component is either moved to the *right* or stays stationary. One can imagine that in a forward step, first the input symbol is read and then the input head is moved to its new position, whereas in a backward step, first the input head is moved to its new position and then the input symbol is read.

So, a deterministic two-party Watson-Crick system $\mathscr{A}$ is said to be *reversible* (REV-PWK) if and only if there exist *reverse transition functions* $\delta_i^{\leftarrow} : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \times (M \cup \{\bot\}) \to Q_i \times \{0, -\}$ and *reverse broadcast functions* $\mu_i^{\leftarrow} : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \to M \cup \{\bot\}$ inducing a relation $\vdash^{\leftarrow}$ from a configuration to its *predecessor configuration*, such that

$$a_0 \cdots a_{i'-1} \overrightarrow{p_1} a_{i'} \cdots a_{j'} \underleftarrow{q_1} a_{j'+1} \cdots a_{n+1} \vdash^{\leftarrow} a_0 \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{n+1}$$

if and only if

$$a_0 \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{n+1} \vdash a_0 \cdots a_{i'-1} \overrightarrow{p_1} a_{i'} \cdots a_{j'} \underleftarrow{q_1} a_{j'+1} \cdots a_{n+1}.$$

In the following, we study the impact of communication in deterministic two-party Watson-Crick systems. The communication is measured by the total number of messages sent during a computation, where it is understood that $\bot$ means no message and, thus, is not counted.

Let $f : \mathbb{N} \to \mathbb{N}$ be a mapping. If all $w \in L(\mathscr{A})$ are accepted with computations where the total number of messages sent is bounded by $f(|w|)$, then $\mathscr{A}$ is said to be *communication bounded by* f. We denote the class of DPWKs that are communication bounded by f by DPWK(f). In case of reversible DPWKs we have to consider the number of messages sent in reverse computations as well. If all $w \in L(\mathscr{A})$ are accepted with computations where the total number of messages sent in forward computations and in reverse computations is each bounded by $f(|w|)$, then $\mathscr{A}$ is said to be *communication bounded by* f and the corresponding class of REV-PWKs is denoted by REV-PWK(f).

In general, the *family of languages accepted* by devices of type X is denoted by $\mathscr{L}(X)$. To illustrate the definitions we start with two examples.

Example 1. The non-regular language $L = \{ a^n b^n \mid n \geq 1 \}$ is accepted by a REV-PWK. The principal idea of the construction is that the upper component starts with one time step delay and then moves its head with maximum speed to the right, whereas the lower component immediately starts to move its head with maximum speed to the left. Both components communicate in every time step the symbol they read. When the lower component has read the rightmost a of the a-block after having passed the b-block, the transition functions ensure that the upper component has to read the leftmost b of the b-block after having passed the a-block. When the lower component has reached the left endmarker, it waits for one time step. To accept the input, the upper component has to read the right endmarker in the final step. In the backward computation the upper component immediately starts, whereas the lower component starts with with one time step delay. When the upper component has read the rightmost a of the a-block after having passed the b-block, the transition functions ensure that the lower component has to read the leftmost b of the b-block after having passed the a-block. Finally, when the upper component has reached the right endmarker, it waits for one time step. To reach the initial configuration the lower component has to read the left endmarker in the next time step.

For the precise construction of a REV-PWK accepting the language $L = \{ a^n b^n \mid n \geq 1 \}$ we define $\mathscr{A} = \langle \{a,b\}, \{a,b,\rhd,\lhd\}, \rhd, \lhd, A_1, A_2 \rangle$ where

$$A_1 = \langle \{p_0, p_1, \ldots, p_5\}, \{a,b\}, \delta_1, \mu_1, p_0, \{p_5\} \rangle \text{ and } A_2 = \langle \{q_0, q_1, \ldots, q_5\}, \{a,b\}, \delta_2, \mu_2, q_0, \{\} \rangle.$$

The broadcast functions μ_1, μ_2 and the reverse broadcast functions $\mu_1^{\leftarrow}, \mu_2^{\leftarrow}$ are defined as $\mu_1(p,x) = \mu_1^{\leftarrow}(p,x) = x$ and $\mu_2(q,x) = \mu_2^{\leftarrow}(q,x) = x$ for all $p \in \{p_0, p_1, \ldots, p_5\}$, $q \in \{q_0, q_1, \ldots, q_5\}$, and $x \in \{a, b, \rhd, \lhd\}$. The transition functions δ_1, δ_2 and $\delta_1^{\leftarrow}, \delta_2^{\leftarrow}$ are as follows.

A_1 forward			
(1)	$\delta_1(p_0, \rhd, \lhd)$	=	$(p_1, 0)$
(2)	$\delta_1(p_1, \rhd, b)$	=	$(p_2, +)$
(3)	$\delta_1(p_2, a, b)$	=	$(p_2, +)$
(4)	$\delta_1(p_2, a, a)$	=	$(p_3, +)$
(5)	$\delta_1(p_3, b, a)$	=	$(p_3, +)$
(6)	$\delta_1(p_3, b, \rhd)$	=	$(p_4, +)$
(7)	$\delta_1(p_4, \lhd, \rhd)$	=	$(p_5, 0)$

A_1 backward			
(1)	$\delta_1^{\leftarrow}(p_1, \rhd, \lhd)$	=	$(p_0, 0)$
(2)	$\delta_1^{\leftarrow}(p_2, \rhd, b)$	=	$(p_1, -)$
(3)	$\delta_1^{\leftarrow}(p_2, a, b)$	=	$(p_2, -)$
(4)	$\delta_1^{\leftarrow}(p_3, a, a)$	=	$(p_2, -)$
(5)	$\delta_1^{\leftarrow}(p_3, b, a)$	=	$(p_3, -)$
(6)	$\delta_1^{\leftarrow}(p_4, b, \lhd)$	=	$(p_3, -)$
(7)	$\delta_1^{\leftarrow}(p_5, \lhd, \rhd)$	=	$(p_4, 0)$

A_2 forward			
(1)	$\delta_2(q_0, \lhd, \rhd)$	=	$(q_1, +)$
(2)	$\delta_2(q_1, b, \rhd)$	=	$(q_2, +)$
(3)	$\delta_2(q_2, b, a)$	=	$(q_2, +)$
(4)	$\delta_2(q_2, a, a)$	=	$(q_3, +)$
(5)	$\delta_2(q_3, a, b)$	=	$(q_3, +)$
(6)	$\delta_2(q_3, \rhd, b)$	=	$(q_4, 0)$
(7)	$\delta_2(q_4, \rhd, \lhd)$	=	$(q_5, 0)$

A_2 backward			
(1)	$\delta_2^{\leftarrow}(q_1, \lhd, \rhd)$	=	$(q_0, -)$
(2)	$\delta_2^{\leftarrow}(q_2, b, \rhd)$	=	$(q_1, -)$
(3)	$\delta_2^{\leftarrow}(q_2, b, a)$	=	$(q_2, -)$
(4)	$\delta_2^{\leftarrow}(q_3, a, a)$	=	$(q_2, -)$
(5)	$\delta_2^{\leftarrow}(q_3, a, b)$	=	$(q_3, -)$
(6)	$\delta_2^{\leftarrow}(q_4, \rhd, b)$	=	$(q_3, 0)$
(7)	$\delta_2^{\leftarrow}(q_5, \rhd, \lhd)$	=	$(q_4, 0)$

We note that it is shown in [13] that $L = \{ a^n b^n \mid n \geq 1 \}$ is not accepted by any reversible pushdown automaton. ■

Example 2. The non-context-free language $L' = \{ w\$w^R\$a^{|w|} \mid w \in \{a,b\}^* \}$ is accepted by a REV-PWK. Here, the principal idea is that the upper component waits at the left endmarker, while the lower component moves across the a-block. Having reached the second \$, both components move with maximum speed and test the structure $w\$w^R$ by communicating in every time step they read. If no error occurred, the upper component moves to the second \$, while the lower component waits at the first \$. Finally, both components move with maximum speed and test the length of w equals the length of the a-block. The moving of the components in the backward computation is straightforward. ■

3 Reversibility versus Irreversibility

We now turn to the question of whether reversible two-party Watson-Crick systems are weaker than irreversible ones or not; it turns out that they are. In fact, there are languages accepted by irreversible two-party Watson-Crick systems that do not need any communication which cannot be accepted by any reversible two-party Watson-Crick system regardless of the number of communications. To show this claim, we will use regular witness languages. Let $\Sigma \supseteq \{a,b\}$ be an alphabet and $I \subseteq \Sigma^*$ be regular such that $I = I^R$. Then we define $L_I = \{ a^{m_1} bvba^{m_2} \mid m_1, m_2 \geq 0, v \in b^* \text{ or } v \in I \}$. So, the words in L_I have a nonempty prefix of a's, followed by a b, followed by a factor of b's or a factor from I, followed by a b, followed by a nonempty suffix of a's.

Theorem 3. *Let $\Sigma \supseteq \{a,b\}$ and $I \subseteq \Sigma^*$. Then language L_I is not accepted by any REV-PWK.*

Proof. Assume for the purpose of contradiction that L_I is accepted by some REV-PWK $\mathscr{A}$. Since we do not limit the number of possible communications, we simply assume that both components send a message at every time step. In this case, for the sake of easier writing, we can assume that there is *one common* finite-state control for both components. This control receives a pair of input symbols in every step, changes the state, and moves the components if required. Now we can argue that the system is irreversible if there are two reachable states that have a common successor state for the same pair of input symbols.

We denote this system M, its set of states Q, and its transition function δ. We now consider accepting computations on words $w = a^x b^y a^z \in L_I$, where x, y, z are long enough. In a first phase of such a computation, eventually at least one component has to start to move across the a-prefix or a-suffix. Otherwise the overall computation would loop forever. Since $L_I = L_I^R$, we can safely assume that the upper component moves. The lower component may move across the a-suffix or stay stationary on the endmarker or some a. We choose x and z large enough such that M runs into a state cycle in this phase. Moreover, we choose z that large that the upper component arrives at the first b after the a-prefix before the lower component has passed the a-suffix. Let $p_1, p_2, \ldots, p_k$ be the state cycle. We can adjust the length of the prefix such that M moves the upper component on the first b while entering state p_k. So, we have a configuration of the form $p_k\colon \rhd aa\cdots a\,\overrightarrow{b}\,b\cdots baa\cdots \underset{\leftarrow}{\sigma}\cdots$, where the state of M is written in front of $\rhd$, and $\sigma = a$ or $\sigma = \lhd$, and the components are scanning the symbols indicated by the arrows. Next, we can enlarge z such that M runs again in a state loop while the upper component is reading b's and the lower component is reading $\lhd$ or a's. Assume that the sequence of states passed through is extended from p_k by $p'_1, p'_2, \ldots, p'_i, p''_1, \ldots p''_j, p''_1$. Then we know $\delta(p'_i, (b, \sigma_1)) = (p''_1, d_1, d_2)$ and $\delta(p''_j, (b, \sigma_2)) = (p''_1, d_1, d_2)$, where d_1, d_2 indicate whether the components are moved or not. Since M is reversible, we derive $p'_1, p'_2, \ldots, p'_i, p''_1, \ldots p''_j, p''_1 = p_1, p_2, \ldots, p_k, p_1$ or $(b, \sigma_1) \neq (b, \sigma_2)$ and, thus, $\sigma_1 \neq \sigma_2$ and, hence, $\sigma_1 = \lhd$ and $\sigma_2 = a$. Dependent on whether the loop on the (a, σ)'s is continued on the (b, σ)'s, or the second possibility, we distinguish two cases. A similar distinction will be made in several sub-cases.

Case A The system M continues to loop through the states $p_1, p_2, \ldots, p_k$ while reading (b, σ)'s. Recall that the current state determines the last movements of the components. Therefore, the upper component moves across the b's. Moreover, we can choose y and z again large enough such that the upper component runs through several loops and M moves the upper component on the first a of the suffix while entering state p_k. So, we have a configuration of the form $p_k\colon \rhd aa\cdots abb\cdots b\,\overrightarrow{a}\,a\cdots a\,\underset{\leftarrow}{\sigma}\cdots$. Now, we can repeat the argument from above and distinguish the two sub-cases that M continues to loop through the states $p_1, p_2, \ldots, p_k, p_1$, or $(a, \sigma_1) \neq (a, \sigma_2)$ and, thus, $\sigma_1 \neq \sigma_2$ and, hence, $\sigma_1 = \lhd$ and $\sigma_2 = a$.

Case A.1 The system M continues to loop through the states $p_1, p_2, \ldots, p_k$ while reading (a, σ)'s. In this sub-case the upper component may reach the right endmarker before the lower component reaches the b before the a-suffix. Then the remaining computation of M is that of a finite automaton, that is, of the lower component. Since the language $a^*b^*a^*$ is not accepted by any reversible DFA, we obtain a contradiction.

Therefore, the upper component may reach the right endmarker not before the lower component reaches the b before the a-suffix. Now, again we can repeat the argument from above and distinguish the two sub-cases that M continues to loop through the states $p_1, p_2, \ldots, p_k$ while moving the lower component or (a, σ_1) must not be equal to (a, σ_2) which can be violated by adjusting the value of z. In this way $\sigma_1 = \sigma_2 = a$, a contradiction. If, however, M continues to loop through the states $p_1, p_2, \ldots, p_k$, by almost the same arguments as before we can obtain a contradiction unless M continues to loop through the states $p_1, p_2, \ldots, p_k$ until the lower component has reached the left endmarker. In this case, the language $\{a, b\}^+$ is accepted.

Case A.2 The sequence of states passed through to reach the configuration $p_k\colon \rhd aa\cdots abb\cdots b\overrightarrow{a}\,a\cdots a\underleftarrow{\sigma}\cdots$ is extended from state p_k by the states $q'_1,q'_2,\ldots,q'_{i'},q''_1,\ldots q''_{j'},q''_1$, and we have $\delta(q'_{i'},(a,\sigma_1))=(q''_1,d_1,d_2)$ and $\delta(q''_{j'},(a,\sigma_2))=(q''_1,d_1,d_2)$, and therefore $(a,\sigma_1)\neq(a,\sigma_2)$ which implies $\sigma_1=\lhd$ and $\sigma_2=a$.

Now, the upper component may or may not reach the right endmarker before the lower component reaches the b before the a-suffix. We obtain a contradiction almost literally as in Case A.1.

Case B The sequence of states passed through to reach the configuration $\rhd aa\cdots a\overrightarrow{b}\,b\cdots baa\cdots\underleftarrow{\sigma}\cdots$ in state p_k is extended from p_k by $p'_1,\ldots,p'_i,p''_1,\ldots p''_j,p''_1$. Then we have $\delta(p'_i,(b,\sigma_1))=(p''_1,d_1,d_2)$ and $\delta(p''_j,(b,\sigma_2))=(p''_1,d_1,d_2)$, and therefore, $(b,\sigma_1)\neq(b,\sigma_2)$ which implies $\sigma_1=\lhd$ and $\sigma_2=a$.

Case B.1 If the upper component moves in the state cycle $p''_1,\ldots p''_j$, then we can choose z again large enough such that the upper component reaches the first a after the b-factor before the lower component reaches the b before the a-suffix. So, a configuration $\rhd aa\cdots abb\cdots b\overrightarrow{a}\,a\cdots\underleftarrow{a}\cdots$ is reached in some state from the cycle. We obtain a contradiction along the argumentation as in Case A.1.

Case B.2 If the upper component does not move in the state cycle $p''_1,\ldots p''_j$, then a configuration $\cdots\overrightarrow{b}\,b\cdots\underleftarrow{b}\,aa\cdots$ is reached in some state from the cycle.

Assume that from here the computation continues in the same state cycle until the lower component has reached the left endmarker. Then the upper component stays on the current input in this phase, and the remaining computation of M is that of a finite automaton, that is, of the upper component on its remaining input of the form b^*a^*, which is not accepted by any reversible DFA. So, we obtain a contradiction.

We conclude that the computation cannot continue in the same state cycle. If it continues in some state cycle $q'_1,q'_2,\ldots,q'_{i'},q''_1,\ldots q''_{j'},q''_1$ while both components read b's, then we have $\delta(q'_{i'},(b,b))=(q''_1,d_1,d_2)$ and $\delta(p''_{j'},(b,b))=(p''_1,d_1,d_2)$ which violates the reversibility.

If the computation continues in some state cycle $q'_1,q'_2,\ldots,q'_{i'},q''_1,\ldots q''_{j'},q''_1$ after at least one component has passed across the b-factor, we obtain a similar contradiction with input pairs (a,b), (b,a), or (a,a).

This concludes the case analysis. Since in any possible case a contradiction is derived, the initial assumption that L_I is accepted by some REV-PWK is wrong and the assertion follows. □

The result of Theorem 3 that there is a regular language that is not accepted by any REV-PWK together with Example 1 showing that the non-regular language $\{\,a^nb^n\mid n\geq 1\,\}$ is accepted by a REV-PWK proves that the class of languages accepted by REV-PWK and the regular languages are incomparable. Since $\{\,a^nb^n\mid n\geq 1\,\}$ is a linear and real-time deterministic context-free language, we immediately obtain the incomparability to the linear context-free languages as well as to the real-time deterministic context-free languages. It is shown in [13] that every regular language can be accepted by a reversible pushdown automaton. Moreover, it is shown that the language $\{\,a^nb^n\mid n\geq 1\,\}$ cannot be accepted by any reversible pushdown automaton. Hence, the classes of languages accepted by REV-PWK and reversible pushdown automata are incomparable as well.

4 Closure Properties

The goal of this section is to collect some closure properties of the families $\mathscr{L}$(REV-PWK). For this purpose, the regular languages L_I can be used very well in several cases. In particular, we consider Boolean operations (complementation, union, intersection) and AFL operations (union, intersection with regu-

lar languages, homomorphism, inverse homomorphism, concatenation, iteration). The positive closure under reversal is trivial. The results are summarized in Table 1 at the end of the section.

Proposition 4. *The family $\mathscr{L}$(REV-PWK) is closed under complementation.*

Proposition 5. *The family $\mathscr{L}$(REV-PWK) is not closed under union, intersection, and intersection with regular languages.*

Proof. Let $\Sigma = \{a,b\}$. For $I = \emptyset$, we consider the regular language $L_\emptyset = \{ a^{m_1}bb^{m_3}ba^{m_2} \mid m_1,m_2,m_3 \geq 0 \}$. By Theorem 3, the regular language $L_\emptyset$ does not belong to the family $\mathscr{L}$(REV-PWK). On the other hand, the language Σ^* does belong to the family. Since $\Sigma^* \cap L_\emptyset = L_\emptyset$, we obtain the non-closure under intersection with regular languages.

The non-closure under intersection is witnessed by the languages, $L_1 = \{ a^m bbv \mid m \geq 0, v \in \{a,b\}^* \}$ and $L_2 = \{ vbba^m \mid m \geq 0, v \in \{a,b\}^* \}$.

We show that L_1 is accepted by some more or less trivial REV-PWK without any communication as follows.

The lower component does nothing, that is, it loops in its non-accepting initial state on the right endmarker. The behavior of the upper component is depicted as a state graph in Figure 1. If and only if the component has seen a correct prefix of the form a^*bb it halts in an accepting state (the rest of the input cannot affect the computation result any more and, by definition, there is no need to read it).

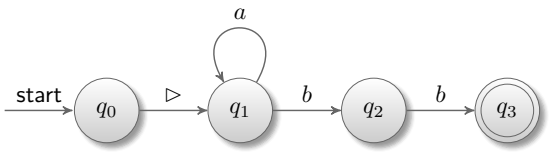


Figure 1: State graph of the upper component of a REV-PWK accepting L_1.

Since $L_2 = L_1^R$ and the closure of $\mathscr{L}$(REV-PWK) under reversal, we conclude that L_2 belongs to $\mathscr{L}$(REV-PWK) as well. However, $L_1 \cap L_2 = L_I$ for $I = b\{a,b\}^*b$ and, thus, the non-closure under intersection follows.

The non-closure under union follows from the closure under complementation and the non-closure under intersection by De Morgan's law. □

Proposition 6. *The family $\mathscr{L}$(REV-PWK) is not closed under concatenation and iteration.*

Proof. The witness language for both operations is $L = \{ a^n b^n \mid n \geq 1 \}$ which belongs to $\mathscr{L}$(REV-PWK) by Example 1.

For the concatenation we consider $L \cdot L$ and for the iteration we consider L^*.

Essentially, using a different but similar language, in [18] it is shown that for n long enough both components have to scan some symbol from each two factors whose lengths have to be compared simultaneously. This argument applies also here. However, the two components can simultaneously stay in two corresponding factors at most for one such pair. This implies that neither the language $L \cdot L$ nor the language L^* is accepted even by any not necessarily reversible DPWK. □

Proposition 7. *The family $\mathscr{L}$(REV-PWK) is not closed under length-preserving homomorphisms.*

Proposition 8. *The family $\mathscr{L}$(REV-PWK) is not closed under inverse homomorphisms.*

Family	$\overline{}$	$\cup$	$\cap$	$\cap_{\text{reg}}$	$\cdot$	$*$	$h_{\text{len.pres.}}$	h^{-1}	R
REV-PWK	✓	✗	✗	✗	✗	✗	✗	✗	✓

Table 1: Closure properties of the language families discussed.

5 Restricted Communication

The REV-PWK considered in the previous sections may communicate arbitrarily often. In this section, we want to consider DPWK and REV-PWK with a restricted amount of communications. According to the definition in Section 2 we have a function $f : \mathbb{N} \to \mathbb{N}$ and define that a DPWK is communication bounded by f if all words w in the language are accepted with computations where the total number of messages sent is bounded by $f(|w|)$. A REV-PWK is communication bounded by f if, in addition, the total number of messages sent in reverse computations is bounded by $f(|w|)$ as well. Here, we will study the language class with constant communication, where $f \in O(1)$, the class with logarithmic communication, where $f \in O(\log(n))$, the class with square root communication, where $f \in O(\sqrt{n})$, and the class with arbitrary, i.e., linear communication, where $f \in O(n)$. The relations of these classes have been investigated for DPWK in [12]. Here, we will complement the results for REV-PWK and clarify the relations between reversible and general, possibly irreversible, devices. We start with an example presenting a non-semilinear language that is accepted by a REV-PWK$(O(\log(n)))$.

Example 9. The language $L_{expo} = \{ a^{2^0} b a^{2^2} b \cdots b a^{2^{2m}} c a^{2^{2m+1}} b \cdots b a^{2^3} b a^{2^1} \mid m \geq 1 \}$ is accepted by a REV-PWK. The rough idea of the construction is that in a first phase the components compare the lengths 2^0 with 2^1, 2^2 with 2^3, ..., and 2^{2m} with 2^{2m+1}. The first phase ends when both components reach the center symbol c. In a second phase, the components compare the length 2^{2m} with 2^{2m-1}, 2^{2m-2} with 2^{2m-3}, ..., and 2^2 with 2^1. To achieve this the lower component has to wait on the c until the upper component has moved across the block $a^{2^{2m+1}}$. To realize the comparisons, the upper component moves across its a-blocks with half speed, whereas the lower component moves across its a-blocks with full speed, that is, one square per step. The length comparisons in the first and second phase are checked by communicating when a b, c, or the right endmarker is reached which must happen synchronously.

The length of an accepted input is $n = 2^{2m+2} + 2m$. There are communications only on symbols b, c, and $\lhd$ both in forward computations and reverse computations. Hence, there are at most $2m+3$ communications in forward computations as well as in reverse computations. Thus, the REV-PWK constructed is a REV-PWK$(O(\log(n)))$ and L_{expo} belongs to $\mathscr{L}$(REV-PWK$(O(\log(n))))$. ■

Lemma 10. *The language* $L_{lin} = \{ wcw^R \mid w \in \{0,1\}^* \}$ *belongs to* $\mathscr{L}$*(REV-PWK*$(O(n))$*).*

Proof. A REV-PWK accepting L_{lin} will move its both components synchronously towards the middle marker c as long as the input symbol read and communicated in every step is equal. In case of inequivalence the computation halts non-accepting. If both components reach the middle marker c at the same time, the first task is nearly accomplished. It remains for the lower component to read the input completely and to halt non-accepting in case of another symbol c occurring. Since both components move synchronously and communicate in every step, it is clear that L_{lin} can be accepted by a REV-PWK$(O(n))$. □

As a combination of Example 9 and Lemma 10 we obtain the following lemma.

Lemma 11. $\hat{L}_{expo} = \{ a^{2^0} x_1 a^{2^2} x_2 \cdots x_m a^{2^{2m}} c a^{2^{2m+1}} x_m \cdots x_2 a^{2^3} x_1 a^{2^1} \mid m \geq 1 \text{ and } x_i \in \{0,1\}, 1 \leq i \leq m \}$ *belongs to* $\mathscr{L}$*(REV-PWK*$(O(\log(n))))$.

Proof. It can be observed from the construction in Example 9 that in the first phase both components communicate on every symbol b and c. So, on the corresponding input from $\hat{L}_{expo}$ both components can communicate on every symbol 0, 1, and c in order to simulate the REV-PWK accepting L_{lin} as a subtask. □

With similar ideas it is possible to show the following lemma.

Lemma 12. *$\hat{L}_{poly} = \{ ax_1a^5x_2 \cdots x_m a^{4m+1} c a^{4m+3} x_m \cdots x_2 a^7 x_1 a^3 \mid m \geq 0$ and $x_i \in \{0,1\}, 1 \leq i \leq m\}$ belongs to $\mathscr{L}$(REV-PWK$(O(\sqrt{n})))$.*

It is shown in [12] that L_{lin} does not belong to $\mathscr{L}(\text{DPWK}(O(f)))$ if $f \in \frac{n}{\omega(\log(n))}$. Hence, L_{lin} does not belong to $\mathscr{L}(\text{REV-PWK}(O(\sqrt{n})))$. It is also shown in [12] that $\hat{L}_{poly}$ does not belong to $\mathscr{L}(\text{DPWK}(O(f)))$ if $f \in O(\log(n))$. Thus, $\hat{L}_{poly}$ does not belong to $\mathscr{L}(\text{REV-PWK}(O(\log(n))))$. Finally, it is known due to [12] that every language in $\mathscr{L}(\text{DPWK}(O(1)))$ is semilinear. Since $\hat{L}_{expo}$ is not semilinear, it does not belong to $\mathscr{L}(\text{REV-PWK}(O(1)))$. Together with Lemma 10, Lemma 11, and Lemma 12 we obtain the following proper hierarchy:

$$\mathscr{L}(\text{REV-PWK}(O(1))) \subset \mathscr{L}(\text{REV-PWK}(O(\log(n)))) \subset \mathscr{L}(\text{REV-PWK}(O(\sqrt{n}))) \subset \mathscr{L}(\text{REV-PWK}(O(n)))$$

Theorem 3 presents a regular language that is not accepted by any REV-PWK$(O(n))$. Since the regular languages belong to $\mathscr{L}(\text{DPWK}(O(1)))$ we immediately obtain proper inclusions between reversible and general language classes with the same amount of communication. These results and the other results of this section are summarized in Figure 2.

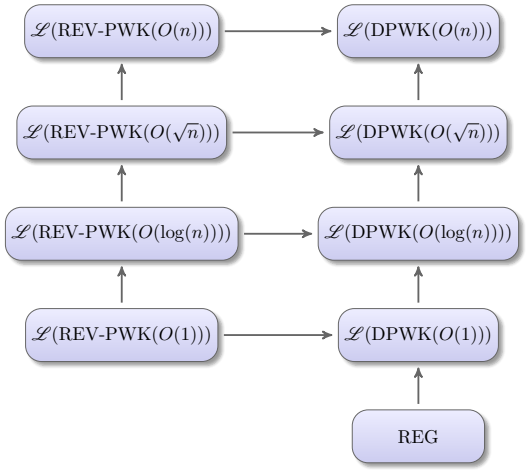


Figure 2: Relationships between language families induced by two-party Watson-Crick systems. An arrow between families indicates a strict inclusion.

6 Decidability Questions

In this section, we will discuss several decidability questions for REV-PWK. It has been shown in [12] that the questions of emptiness, finiteness, inclusion, and equivalence are decidable for general, possibly irreversible, DPWK in case of a finite number of communications. This result leads immediately to the following decidability results for REV-PWK in case of a finite number of communications.

Theorem 13. *Let $k \geq 0$ be a constant. Then emptiness, finiteness, inclusion, and equivalence are decidable for REV-PWK(k).*

Next, we want to obtain that the decidability questions become undecidable if a non-constant number of communications is used. In a first step, we show that the questions of emptiness, finiteness, inclusion, and equivalence are undecidable and, moreover, not even semidecidable for REV-PWK in case of a linear number of communications used. In a second step, we will obtain the same non-semidecidability results with a superlogarithmic number of communications used.

It has been shown in [14] that the questions of testing emptiness, finiteness, inclusion, and equivalence are not semidecidable for *reversible two-head finite automata*. The difference between such automata and DPWK is that the former move their two heads in the same direction from left to right, whereas the latter move both heads in opposite directions. Now, the idea is to simulate a reversible two-head finite automaton by a REV-PWK.

The non-semidecidability results for reversible two-head finite automata are obtained by showing that the set VALC_M of suitably encoded valid computations of a deterministic linearly space bounded one-tape, one-head Turing machine M, so-called linear bounded automaton (LBA) can be accepted by a reversible two-head finite automaton. It should be noted that the due to technical reasons the definition of the set VALC_M in [14] considers valid computations on inputs of length at least 2.

Now, we will construct a REV-PWK$(O(n))$ that accepts the set $\mathrm{VALC}'_M = \{ w^R c w \mid w \in \mathrm{VALC}_M \}$, where the set VALC_M is defined over some alphabet A and $c \notin A$ is a new symbol.

Lemma 14. *Let M be an LBA. Then, a REV-PWK$(O(n))$ accepting VALC'_M can effectively be constructed.*

Proof. Let M be an LBA. A REV-PWK M' accepting VALC'_M has to accomplish two tasks. First, M' will test the structure $w^R c w$ disregarding whether w belongs to VALC_M or not. To achieve this task we use a similar approach as described in the proof of Lemma 10. Both components will move synchronously towards the middle marker c as long as the input symbol read and communicated in every step is equal. The structure $w^R c w$ is correctly tested, if both components reach the middle marker c at the same time. Then, the first task is nearly accomplished, but it remains for the lower component, while accomplishing the second task, to read the input completely and to halt non-accepting in case of another symbol c occurring. Since both components move synchronously and communicate in every step, it is clear that the first task can be realized by a REV-PWK$(O(n))$.

For the second task, we first observe that the remaining input for both components is the same word w and it remains to be checked whether or not w belongs to VALC_M. This can now be realized by implementing the construction given in [14] for two-head finite automata. The head 1 is simulated by the upper component and head 2 is simulated by the lower component, whereby the middle marker c is interpreted as the left endmarker for the two-head finite automaton. In this construction the lower component reads the input completely and can halt non-accepting if another symbol c is read. Since the two-head finite automaton is reversible, the second task and, therefore, the complete construction can be realized by a REV-PWK$(O(n))$. □

This leads immediately to the following non-semidecidability results.

Theorem 15. *The problems of testing emptiness, finiteness, inclusion, and equivalence are not semidecidable for a given REV-PWK$(O(n))$.*

Proof. Let M be an LBA accepting inputs over the alphabet Σ. According to Lemma 14 we can effectively construct a REV-PWK$(O(n))$ M' accepting VALC'_M. Clearly, $L(M') = \mathrm{VALC}'_M$ is empty if and

only if VALC_M is empty if and only if $L(M)$ is either empty or contains some words from the finite set $\{\lambda\} \cup \Sigma$. The latter words have to be considered, since M may accept words of length less than two. Since the word problem is decidable for LBAs and emptiness is not semidecidable for LBAs, the non-semidecidability of emptiness follows.

We also obtain that $L(M') = \mathrm{VALC}'_M$ is finite if and only if VALC_M is finite if and only if $L(M)$ is finite. Since finiteness is not semidecidable for LBAs, the non-semidecidability of finiteness follows.

Finally, it is easy to effectively construct a REV-PWK(1) that accepts nothing. Hence, the non-semidecidability of equivalence and inclusion follows immediately. □

Our next step is to obtain these non-semidecidability results also for REV-PWK with less communication. Our approach is to define another variant of VALC'_M in which the length of each configuration is enlarged while the same amount of communication is being kept. A similar approach has been used in [12] for general, possibly irreversible, DPWK. However, here the details are quite different and more complicated since the construction has to be reversible. The detailed and lengthy construction is omitted here. With all these prerequisites it is possible to show the following theorem.

Theorem 16. *The problems of testing emptiness, finiteness, inclusion, and equivalence are not semidecidable for a given REV-PWK*$(O(\log(n) \cdot \log(\log(n))))$.

References

[1] Dana Angluin (1982): *Inference of reversible languages*. *J. ACM* 29(3), pp. 741–765, doi:10.1145/322326.322334.

[2] Charles H. Bennett (1973): *Logical Reversibility of Computation*. *IBM J. Res. Dev.* 17, pp. 525–532, doi:10.1147/rd.176.0525.

[3] Henning Bordihn & György Vaszil (2021): *Reversible parallel communicating finite automata systems*. *Acta Inf.* 58(4), pp. 263–279, doi:10.1007/s00236-021-00396-9.

[4] Kingshuk Chatterjee & Kumar Sankar Ray (2017): *Reversible Watson-Crick automata*. *Acta Inf.* 54(5), pp. 487–499, doi:10.1007/s00236-016-0267-0.

[5] Kingshuk Chatterjee & Kumar Sankar Ray (2017): *Watson-Crick pushdown automata*. *Kybernetika* 53(5), pp. 868–876, doi:10.14736/kyb-2017-5-0868.

[6] Elena Czeizler, Eugen Czeizler, Lila Kari & Kai Salomaa (2009): *On the descriptional complexity of Watson-Crick automata*. *Theor. Comput. Sci.* 410, pp. 3250–3260, doi:10.1016/j.tcs.2009.05.001.

[7] Rudolf Freund, Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa (1997): *Watson-Crick Finite Automata*. In: *DIMACS Workshop on DNA Based Computers*, University of Pennsylvania, Philadelphia, pp. 305–317, doi:10.1090/dimacs/048/22.

[8] Markus Holzer, Sebastian Jakobi & Martin Kutrib (2018): *Minimal Reversible Deterministic Finite Automata*. *Int. J. Found. Comput. Sci.* 29, pp. 251–270, doi:10.1142/S0129054118400063.

[9] Radim Kocman, Zbynek Krivka, Alexander Meduna & Benedek Nagy (2022): *A jumping $5' \to 3'$ Watson-Crick finite automata model*. *Acta Inf.* 59(5), pp. 557–584, doi:10.1007/s00236-021-00413-x.

[10] Attila Kondacs & John Watrous (1997): *On the Power of Quantum Finite State Automata*. In: *Foundations of Computer Science (FOCS 1997)*, IEEE Computer Society, pp. 66–75, doi:10.1109/SFCS.1997.646094.

[11] Martin Kutrib (2014): *Aspects of Reversibility for Classical Automata*. In C. S. Calude, G. R. Freivalds & K. Iwama, editors: *Computing with New Resources*, *LNCS* 8808, Springer, pp. 83–98, doi:10.1007/978-3-319-13350-8_7.

[12] Martin Kutrib & Andreas Malcher (2011): *Two-Party Watson-Crick Computations*. In: *Implementation and Application of Automata (CIAA 2010)*, *LNCS* 6482, Springer, pp. 191–200, doi:10.1007/978-3-642-18098-9_21.

[13] Martin Kutrib & Andreas Malcher (2012): *Reversible Pushdown Automata*. *J. Comput. Syst. Sci.* 78, pp. 1814–1827, doi:10.1016/j.jcss.2011.12.004.

[14] Martin Kutrib & Andreas Malcher (2017): *One-way reversible multi-head finite automata*. *Theor. Comput. Sci.* 682, pp. 149–164, doi:10.1016/j.tcs.2016.11.006.

[15] Martin Kutrib & Andreas Malcher (2022): *Reversible Computations of One-Way Counter Automata*. In Henning Bordihn, Géza Horváth & György Vaszil, editors: *NCMA 2022*, *EPTCS* 367, pp. 126–142, doi:10.4204/EPTCS.367.9.

[16] Martin Kutrib, Andreas Malcher & Matthias Wendlandt (2016): *Reversible Queue Automata*. *Fund. Inform.* 148, pp. 341–368, doi:10.3233/FI-2016-1438.

[17] Yves Lecerf (1963): *Logique Mathématique: Machines de Turing réversible*. *C. R. Séances Acad. Sci.* 257, pp. 2597–2600.

[18] Peter Leupold & Benedek Nagy (2010): $5' \to 3'$ *Watson-Crick Automata with Several Runs*. *Fund. Inform.* 104, pp. 71–91, doi:10.3233/FI-2010-336.

[19] Kenichi Morita (2011): *Two-Way Reversible Multi-Head Finite Automata*. *Fund. Inform.* 110, pp. 241–254, doi:10.3233/FI-2011-541.

[20] Benedek Nagy (2007): *On* $5' \to 3'$ *Sensing Watson-Crick Finite Automata*. In: *DNA Computing*, *LNCS* 4848, Springer, pp. 256–262, doi:10.1007/978-3-540-77962-9_27.

[21] Benedek Nagy (2013): *On a hierarchy of* $5' \to 3'$ *sensing Watson-Crick finite automata languages*. *J. Log. Comput.* 23, pp. 855–872, doi:10.1093/logcom/exr049.

[22] Benedek Nagy (2020): $5' \to 3'$ *Watson-Crick pushdown automata*. *Inf. Sci.* 537, pp. 452–466, doi:10.1016/j.ins.2020.06.031.

[23] Benedek Nagy & Zita Kovács (2021): *On deterministic 1-limited sensing* $5' \to 3'$ *Watson-Crick finite-state transducers*. *RAIRO Theor. Informatics Appl.* 55, p. 5, doi:10.1051/ita/2021007.

[24] Benedek Nagy, Shaghayegh Parchami & Hamid Mir Mohammad Sadeghi (2017): *A New Sensing* $5' \to 3'$ *Watson-Crick Automata Concept*. In Erzsébet Csuhaj-Varjú, Pál Dömösi & György Vaszil, editors: *AFL 2017*, *EPTCS* 252, pp. 195–204, doi:10.4204/EPTCS.252.19.

Pumping Lemmata for Recognizable Weighted Languages over ARTINIAN Semirings

Andreas Maletti
Universität Leipzig
Faculty of Mathematics and Computer Science
PO Box 100 920, 04009 Leipzig, Germany
maletti@informatik.uni-leipzig.de

Nils Oskar Nuernbergk
nils.nuernbergk@gmail.com

Pumping lemmata are the main tool to prove that a certain language does not belong to a class of languages like the recognizable languages or the context-free languages. Essentially two pumping lemmata exist for the recognizable weighted languages: the classical one for the BOOLEAN semiring (i.e., the unweighted case), which can be generalized to zero-sum free semirings, and the one for fields. A joint generalization of these two pumping lemmata is provided that applies to all ARTINIAN semirings, over which all finitely generated semimodules have a finite bound on the length of chains of strictly increasing subsemimodules. Since ARTINIAN rings are exactly those that satisfy the Descending Chain Condition, the ARTINIAN semirings include all fields and naturally also all finite semirings (like the BOOLEAN semiring). The new pumping lemma thus covers most previously known pumping lemmata for recognizable weighted languages.

1 Introduction

The class of recognizable languages [28] is certainly the best-studied and one of the most useful classes of languages. It has excellent closure properties, and all standard decision problems for it are decidable. Applications of the recognizable languages are too numerous to list, but include pattern matching [2, Chapter 10], lexical analysis [1], input validation [25], network protocols [18], and DNA sequence analysis [26]. Pumping lemmata are statements of the form that given a suitably long word in the language, we can always identify a subword that can be iterated (or pumped) at will without leaving the language. Such statements exist for many language classes including the recognizable [28] and context-free languages [6], and they allow a relatively straightforward proof that a given language does not belong to the class (e.g., is not recognizable).

In several applications [3, 7, 15], the purely qualitative yes/no-decision of languages is completely insufficient. This led to the introduction of weighted languages [24] (see [21] for an excellent survey), in which each word is assigned a weight from a semiring [12, 11]. The classical recognizable languages are reobtained by considering the support of the recognizable weighted languages over the BOOLEAN semiring $(\{0,1\}, \max, \min, 0, 1)$. The theory of recognizable weighted languages is also very well developed and several textbooks [22, 16, 9] provide excellent introductions.

Determining whether a given weighted language is recognizable is often even more difficult than in the unweighted case, and we again mostly rely on pumping lemmata [13, 20] to prove that a given weighted language is not recognizable. However, the coverage situation is very unsatisfactory. The classical pumping lemma for unweighted languages can be lifted to all zero-sum free semirings [12, 11] (i.e., semirings in which $a + b = 0$ implies $a = 0 = b$) by means of a semiring homomorphism from such a semiring into the BOOLEAN semiring [27] and a construction that avoids zero-divisors [14]. On the other hand, the pumping lemmata of [13, 20] require the semiring to be a field, which necessarily is not

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 155–169, doi:10.4204/EPTCS.386.13

zero-sum free. Despite their similarities, the two recalled pumping lemmata thus apply to completely disjoint sets of semirings, which do not even cover all semirings (e.g., the finite ring $\mathbb{Z}_4$ is not zero-sum free and not a field). Indeed it is well-known [10] how to handle finite semirings like $\mathbb{Z}_4$ (by encoding the weights into the states), so that the classical unweighted pumping lemma becomes applicable. Similarly, it is known how to handle semirings like $\mathbb{Z}$ that embed into a field, but there are also infinite semirings that are not zero-sum free and not (embeddable into) a field like the ring $\mathbb{Q}[x]/(x^2)$ of rational linear polynomials. The ring $\mathbb{Q}[x]/(x^2)$ cannot embed into a field since it has zero-divisors (e.g., $x \cdot x = 0$), but it fulfills the requirements for our pumping lemma. Hence there are semirings for which we currently have no available pumping lemma, as well as different semirings that permit essentially the same pumping lemma for their recognizable weighted languages but with totally different justifications.

Let us recall the statement of these pumping lemmata. Let $L\colon \Sigma^* \to S$ be a recognizable weighted language, which assigns to each word $w \in \Sigma^*$ a weight $L(w) \in S$ in the semiring S. The support of L is the set $\operatorname{supp} L = \{w \in \Sigma^* \mid L(w) \neq 0\}$ of nonzero-weighted words in L. The pumping lemma states that given a sufficiently long word $w \in \operatorname{supp} L$, there exists a decomposition $w = uxv$ such that $ux^kv \in \operatorname{supp} L$ for infinitely many $k \in \mathbb{N}$. In other words, ux^kv is also nonzero-weighted in L for infinitely many $k \in \mathbb{N}$, where $ux^kv = ux\cdots xv$ with k repetitions of x.

In this contribution we will establish such a pumping lemma for a class of semirings that includes all fields and all finite semirings. Thus, we directly cover both the pumping lemmata of [13, 20] as well as the classical pumping lemma [19, Lemma 2]. We achieve this by following the general approach of [20] while trying to avoid the vector space structure utilized there. This requires some minor adjustments and, in particular, a replacement for the dimension, for which we use the length of a semimodule. A semimodule has finite length if there is a finite bound on the length of strictly increasing chains of subsemimodules. This notion also allows us to define the ARTINIAN semirings that we consider. A semiring is ARTINIAN if each finitely generated semimodule has finite length. The ARTINIAN semirings include all fields and all finite semirings, but not all zero-sum free semirings. However, the mentioned approach for zero-sum free semirings (applying the homomorphism into the BOOLEAN semiring and avoiding zero-divisors) naturally also works with our pumping lemma.

We first show that any endomorphism of a semimodule over an ARTINIAN semiring is surjective if and only if it is injective, which is a generalization of a well-known statement for vector spaces. Following the approach of [20], we introduce pseudoregular endomorphisms using 2 of the 5 characterizing properties utilized in [20, Proposition 1]. Fortunately, these are the two main properties needed for the proof of our pumping lemma, and the remaining 3 properties rely on infrastructure that is not generally available in our semimodules (instead of the vector spaces used in [20]). The argument that a sufficiently long composition of endomorphisms needs to contain a pseudoregular endomorphism can be taken over mostly unchanged from [13], which then almost directly yields our main pumping lemma. Finally, we also briefly consider pumping lemmata for infinite alphabets.

2 Preliminaries

We denote the non-negative integers by $\mathbb{N}$ and the positive integers by $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. Moreover, we let $\mathbb{Q}^{\geqslant 0} = \{q \in \mathbb{Q} \mid q \geqslant 0\}$ be the set of non-negative rational numbers. For every alphabet Σ we denote the free monoid over Σ by Σ^*, i.e., Σ^* is the set of all finite words with letters in Σ. We write ε for the empty word (the neutral element of the free monoid). Additionally, we let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For all sets A, B, and C and all maps $f\colon A \to B$ and $g\colon B \to C$, we let $\mathrm{id}_A = \{(a,a) \mid a \in A\}$ and $(gf)\colon A \to C$ be the map such that $(gf)(a) = g(f(a))$ for every $a \in A$. Finally, if $A = B$, then we let $f^0 = \mathrm{id}_A$ and $f^{k+1} = ff^k$

for every $k \in \mathbb{N}$.

A *(commutative) semiring* [12, 11] is an algebraic structure $(S,+,\cdot,0,1)$, in which S is a set, called *carrier*, $(S,+,0)$ and $(S,\cdot,1)$ are commutative monoids, called *additive* and *multiplicative monoid* respectively, and

$$r\cdot(s+t) = (r\cdot s)+(r\cdot t), \qquad \text{(distributivity)}$$
$$0\cdot r = 0 \qquad \text{(absorption of 0)}$$

for all $r,s,t \in S$. We will refer to the semiring $(S,+,\cdot,0,1)$ simply by its carrier set S and denote multiplication by juxtaposition as usual. For the rest of the contribution, let S be a commutative semiring.

A (commutative) *ring* is simply a semiring in which every element has an additive inverse, and a (commutative) *semifield* is similarly a semiring in which every element $s \in S \setminus \{0\}$ has a multiplicative inverse. As usual, a (commutative) *field* is a ring that is also a semifield. The BOOLEAN semifield is $\mathbb{B} = (\{0,1\},\max,\min,0,1)$.

An S-semimodule [12, 11] is a tuple $(M,\oplus,0_M,\odot)$ consisting of a commutative monoid $(M,\oplus,0_M)$ and a mapping $\odot\colon S\times M\to M$ such that

$$(r\cdot s)\odot u = r\odot(s\odot u), \qquad \text{(associativity)}$$
$$r\odot(u\oplus v) = (r\odot u)\oplus(r\odot v), \qquad \text{(left distributivity)}$$
$$(r+s)\odot u = (r\odot u)\oplus(s\odot u), \qquad \text{(right distributivity)}$$
$$0\odot u = 0_M \qquad \text{(absorption of 0)}$$

for all semiring elements $r,s \in S$, also called *scalars*, and semimodule elements $u,v \in M$. As before, we write just M for the semimodule $(M,\oplus,0_M,\odot)$, and due to the compatibility axioms presented above, we can safely stop distinguishing the semimodule addition $\oplus$ and semiring addition $+$, writing just $+$ for both, as well as mixed multiplication $\odot$ and semiring multiplication $\cdot$, writing $\cdot$ for both, and the additive neutral element 0_M of the semimodule and its corresponding element 0 of the semiring, writing 0 for both. Finally, we let $su = s\cdot u$ for all $s \in S$ and $u \in M$. It is clear that the semiring S itself forms a semimodule, semimodules over rings are simply modules, and semimodules over fields are vector spaces. A *subsemimodule* of M is a subset $N \subseteq M$ such that $0 \in N$, $m+n \in N$ for all $m,n \in N$, and $r\cdot n \in N$ for all $r \in S$ and $n \in N$. In other words, a subsemimodule is a subset that forms a semimodule itself with respect to the operations of M suitably restricted to N. We write $N \preceq M$ if N is a subsemimodule of M. For every subset $V \subseteq M$ we write $\langle V\rangle$ for the *span* of V (i.e., the smallest subsemimodule of M that contains V) and say that $\langle V\rangle$ is *generated* by V.

Let M and N be two semimodules and $\varphi\colon M\to N$ a mapping. Then φ is *linear* (or a *semimodule homomorphism*) if

$$s\cdot\varphi(u) = \varphi(s\cdot u) \qquad \text{and} \qquad \varphi(u+v) = \varphi(u)+\varphi(v)$$

for all $s \in S$ and $u,v \in M$. Note that $\varphi(0) = 0$ if φ is linear by the former condition. If φ is bijective and linear, then we call φ an *isomorphism* and say that M and N are *isomorphic*, which we write as $M \cong N$. We let $\ker\varphi = \{m \in M \mid \varphi(m) = 0\}$ be the *kernel* of φ and $\operatorname{im}\varphi = \{\varphi(m) \mid m \in M\}$ be the *image* of φ in N, which is always a subsemimodule of N provided that φ is linear. The first isomorphism theorem [4, p. 162, Corollary 5.16] states that $M/\ker\varphi \cong \operatorname{im}\varphi$ for every ring S and linear map φ. Here, $M/\ker\varphi$ is the set of equivalence classes $M/\sim$ with the equivalence relation $\sim$ given by $m \sim n$ if $m-n \in \ker\varphi$ and addition and scalar multiplication defined by $[m]+[n] = [m+n]$ and $s[m] = [sm]$ (where $[m]$ denotes the equivalence class of m). Thus, over a ring S, the linear map φ is injective if and only if $\ker\varphi = \{0\}$.

Moreover, we let

$$\mathrm{Hom}(M,N) = \{\varphi \colon M \to N \mid \varphi \text{ is linear}\}, \quad \mathrm{End}(M) = \mathrm{Hom}(M,M), \quad \text{and} \quad M^\vee = \mathrm{Hom}(M,S),$$

which form semimodules with pointwise addition and scalar multiplication. The semimodule $\mathrm{End}(M)$ contains the *endomorphisms* of M, and $M^\vee$ is called the *dual semimodule* of M.

Let Q be an arbitrary set. Then

$$S^Q = \{f \colon Q \to S \mid \ker f \text{ is co-finite}\}$$

forms a semimodule with pointwise addition and scalar multiplication that we call the *free semimodule over* Q (unique up to isomorphism as usual). This is justified by the fact [11, p. 194] that for any semimodule M every mapping $\varphi \colon Q \to M$ uniquely extends to a linear map $\widetilde{\varphi} \colon S^Q \to M$ such that $\widetilde{\varphi}(\iota_q) = \varphi(q)$, where $\iota_q \in S^Q$ is the mapping given for every $p \in Q$ by

$$\iota_q(p) = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{otherwise.} \end{cases}$$

In particular, if $\operatorname{im}\varphi$ generates M, then $\widetilde{\varphi}$ is surjective. If S is a field, then every semimodule (i.e., vector space) is free, but the same is not true for arbitrary semirings S. If Q is finite, then we say that S^Q is of *rank* $n = |Q|$ and will often identify S^Q with the semimodule S^n.

The spaces $\mathrm{Hom}(M,N)$, $\mathrm{End}(M)$, and $M^\vee$ are particularly easy to describe when M and N are free of finite rank [11, p. 195]. These are exactly the matrix spaces

$$\mathrm{Hom}(S^Q,S^P) \cong S^{P\times Q}, \qquad \mathrm{End}(S^Q) \cong S^{Q\times Q}, \qquad \text{and} \qquad (S^Q)^\vee \cong S^{\{1\}\times Q} \cong S^Q.$$

Note also that S^Q itself can be identified with the matrix space $S^{Q\times\{1\}}$. Matrix multiplication (i.e., composition of linear maps) is then defined as follows: for every $M \in S^{P\times Q}$ and $N^{Q\times R}$, the matrix $M \cdot N \in S^{P\times R}$ is given for all $p \in P$ and $r \in R$ by

$$(M \cdot N)_{pr} = \sum_{q\in Q} M_{pq} \cdot N_{qr}.$$

We will usually state theorems in terms of linear maps instead of matrices due to their greater generality (non-free semimodules do not generally permit descriptions by matrices) and clarity of presentation.

Let Σ be an alphabet. A *weighted language* over Σ is a function $L \colon \Sigma^* \to S$. Given $w \in \Sigma^*$ and a weighted language $L \colon \Sigma^* \to S$, we occasionally write L_w instead of $L(w)$. The *support* of L is the set $\operatorname{supp} L = \{w \in \Sigma^* \mid L_w \neq 0\}$.

A *linear representation* [10] of a weighted language $L \colon \Sigma^* \to S$ is a tuple $(Q, \mathrm{in}, \mathrm{out}, \mu)$, where Q is a finite set of *states*, $\mathrm{in} \in (S^Q)^\vee$ is an *input vector*, $\mathrm{out} \in S^Q$ is an *output vector*, and $\mu \colon \Sigma^* \to \mathrm{End}(S^Q)$ is a monoid homomorphism (where the monoid structure on $\mathrm{End}(S^Q)$ is given by composition of maps), such that for every $w \in \Sigma^*$

$$L_w = \mathrm{in} \cdot \mu(w) \cdot \mathrm{out}.$$

If a weighted language L admits a linear representation, then we call L *recognizable*. This definition of recognizability is equivalent to other common definitions given in terms of weighted automata [21].

3 Semimodules of Finite Length

We recall that the dimension of a finite dimensional vector space V provides an upper bound on the number of proper inclusions in any chain of subspaces of V; i.e., if $V_0 \preceq \cdots \preceq V_r$ is a chain of subspaces of V and $r > \dim V$, then there is at least one $0 \leqslant i < r$ such that $V_i = V_{i+1}$.

In this spirit, we define the *length* $\ell(M) \in \mathbb{N} \cup \{\infty\}$ of a semimodule M to be the (possibly infinite) least upper bound on the number of proper inclusions in any chain of subsemimodules of M; i.e.,

$$\ell(M) = \sup\{r \mid M_0 \prec \cdots \prec M_r \text{ is a chain of strictly increasing subsemimodules of } M\}.$$

Clearly, $\dim V = \ell(V)$ for every finite dimensional vector space V. However, the length is distinct from the rank of a free module even if S is a ring. For example, $\mathbb{Z}$ has rank 1 as a $\mathbb{Z}$-module, but $\ell(\mathbb{Z}) = \infty$ since

$$\langle k^m \rangle \prec \langle k^{m-1} \rangle \prec \cdots \prec \langle k \rangle$$

is a chain of strictly increasing submodules of $\mathbb{Z}$ for every $k \in \mathbb{Z} \setminus \{0, 1, -1\}$ and $m \geqslant 2$.

Definition 3.1. We say that a semimodule M has *finite length* if $\ell(M) \in \mathbb{N}$; i.e., $\ell(M)$ is finite. ■

Let us provide some examples of semimodules that have finite length.

Example 3.2.

(i) Finite dimensional vector spaces over fields have finite length.

(ii) Finite semimodules have finite length.

(iii) We consider the commutative monoid $M = \mathbb{Q}^{\geqslant 0} \cup \{\infty\}$ with $u + \infty = \infty$ for all $u \in M$ and addition defined as in $\mathbb{Q}$ otherwise. Then M is a semimodule over $\mathbb{Q}^{\geqslant 0}$ via

$$m \odot u = \begin{cases} 0 & \text{if } m = 0 \\ \infty & \text{if } m \neq 0 \text{ and } u = \infty \\ m \cdot u & \text{otherwise.} \end{cases}$$

We can easily see that the only subsemimodules of M are $\{0\}$, $\{0, \infty\}$, $\mathbb{Q}^{\geqslant 0}$ and M itself. By considering the inclusions among these subsemimodules, we obtain $\ell(M) = 2$. Notably, this is an example of an infinite semimodule that has finite length, but cannot be embedded into a module over a ring. The embedding fails since ∞ is additively absorptive (i.e., $u + \infty = \infty$ for all $u \in M$, which yields that ∞ cannot be inverted). ■

Let M be a semimodule that has finite length. Next we show that the image $\operatorname{im}\varphi$ of a linear map $\varphi \colon M \to N$ necessarily has finite length as well.

Lemma 3.3. *Let M and N be semimodules and $\varphi \colon M \to N$ be a linear map. Then $\ell(\operatorname{im}\varphi) \leqslant \ell(M)$.*

Proof. If $\ell(M) = \infty$, then the statement holds automatically. Therefore, suppose that $\ell(M) \in \mathbb{N}$ is finite. We recall that the preimage $\varphi^{-1}(L)$ of a subsemimodule $L \preceq N$ is a subsemimodule of M. To see this, let $u, v \in \varphi^{-1}(L)$. Then $\varphi(u+v) = \varphi(u) + \varphi(v) \in L$, and thus $u + v \in \varphi^{-1}(L)$. Similarly, for every $s \in S$ we have $\varphi(su) = s\varphi(u) \in L$, and thus $su \in \varphi^{-1}(L)$. Thus, any chain $N_0 \preceq \cdots \preceq N_r$ of subsemimodules of $\operatorname{im}\varphi$ induces a chain $\varphi^{-1}(N_0) \preceq \cdots \preceq \varphi^{-1}(N_r)$ of subsemimodules of M. Next, we establish that $\varphi^{-1}(N_i) \prec \varphi^{-1}(N_{i+1})$ for every $0 \leqslant i < r$ such that $N_i \prec N_{i+1}$. To this end, let $u \in N_{i+1} \setminus N_i$ and select $v \in \varphi^{-1}(\{u\})$, which is possible because $N_{i+1} \preceq \operatorname{im}\varphi$. Obviously, $v \notin \varphi^{-1}(N_i)$, which proves that $v \in \varphi^{-1}(N_{i+1}) \setminus \varphi^{-1}(N_i)$ and thus $\varphi^{-1}(N_i) \prec \varphi^{-1}(N_{i+1})$. Hence, $\ell(\operatorname{im}\varphi) \leqslant \ell(M)$ follows immediately from the definition. □

The preceding lemma already suggests that semimodules of finite length share nice properties with finite dimensional vector spaces. In order to harness these, it would be very desirable for the class of finite length semimodules to have good closure properties. However, it is not even closed under direct sums, as the following example demonstrates.

Example 3.4. Consider the semifield $S = \mathbb{Q}^{\max} = (\mathbb{Q}^{\geqslant 0}, \max, \cdot, 0, 1)$. As usual, $\mathbb{Q}^{\max}$ is a semimodule over itself, and the presence of multiplicative inverses immediately yields that $\ell(\mathbb{Q}^{\max}) = 1$ because its only subsemimodules are $\{0\}$ and $\mathbb{Q}^{\max}$: if $H \preceq \mathbb{Q}^{\max}$ and $H \neq \{0\}$, there is an $h \in H$ with $h \neq 0$, so $s = s \cdot h^{-1} \cdot h \in \mathbb{Q}^{\max}$ for all $s \in \mathbb{Q}^{\max}$; whereby $H = \mathbb{Q}^{\max}$ (indeed, this argument works for any semifield).

Now we consider the direct sum $M = \mathbb{Q}^{\max} \oplus \mathbb{Q}^{\max}$ of two copies of $\mathbb{Q}^{\max}$, which consists of pairs of rational numbers with the maximum applied coordinate-wise. Clearly, M is also a $\mathbb{Q}^{\max}$-semimodule via a coordinate-wise product. However, M does not have finite length over $\mathbb{Q}^{\max}$ by the following lemma. ■

Lemma 3.5. *The $\mathbb{Q}^{\max}$-semimodule $\mathbb{Q}^{\max} \oplus \mathbb{Q}^{\max}$ has length $\ell(\mathbb{Q}^{\max} \oplus \mathbb{Q}^{\max}) = \infty$.*

Proof. Let $M = \mathbb{Q}^{\max} \oplus \mathbb{Q}^{\max}$. First, we define the function $q\colon M \to \mathbb{Q}$ such that $q\big(\langle a,b\rangle\big) = \frac{a}{b}$ for every $a, b \in \mathbb{Q}^{\max}$. Obviously,

$$q\big(s\langle a,b\rangle\big) = q\big(\langle sa, sb\rangle\big) = \frac{sa}{sb} = \frac{a}{b} = q\big(\langle a,b\rangle\big) \tag{1}$$

for all $\langle a,b\rangle \in M$ and $s \in \mathbb{Q}^{\max}$. Additionally, for all $\langle a,b\rangle, \langle c,d\rangle \in M$ we have

$$q\Big(\max\big(\langle a,b\rangle, \langle c,d\rangle\big)\Big) \leqslant \max\Big(q\big(\langle a,b\rangle\big), q\big(\langle c,d\rangle\big)\Big) \tag{2}$$

because

$$\frac{a}{\max(b,d)} \leqslant \frac{a}{b} = q\big(\langle a,b\rangle\big) \qquad \text{and} \qquad \frac{c}{\max(b,d)} \leqslant \frac{c}{d} = q\big(\langle c,d\rangle\big),$$

which yield

$$q\Big(\max\big(\langle a,b\rangle, \langle c,d\rangle\big)\Big) = \frac{\max(a,c)}{\max(b,d)} = \max\Big(\frac{a}{\max(b,d)}, \frac{c}{\max(b,d)}\Big) \leqslant \max\Big(q\big(\langle a,b\rangle\big), q\big(\langle c,d\rangle\big)\Big).$$

For every $i \in \mathbb{N}$ let $u_i = \langle i, 1\rangle$ and $M_i = \langle\{u_0, \ldots, u_i\}\rangle$ be the subsemimodule generated by $\{u_0, \ldots, u_i\}$. Due to the properties (1) and (2) of q, we have $q(u) \leqslant q(u_i)$ for every $u \in M_i$. This immediately yields $M_i \prec M_{i+1}$ for every $i \in \mathbb{N}$ and thus $M_0 \prec \cdots \prec M_i \prec \cdots$ is an infinite chain of strictly increasing subsemimodules.[1] □

Fortunately, for rings S the situation does not look nearly as bleak and the expected equalities for length hold, as expressed in the next theorem.

Theorem 3.6. *Suppose that S is a ring.*

(i) Let M and N be modules such that $N \preceq M$. If N and M/N both have finite length, then M has finite length and $\ell(M) = \ell(N) + \ell(M/N)$.

[1] In fact, this is an example of a more general pathology of semimodules. Finite length semimodules are NOETHERIAN since they satisfy the Ascending Chain Condition (i.e., every ascending chain of subsemimodules terminates). This proof demonstrates that NOETHERIAN semimodules, unlike NOETHERIAN modules over rings, are not closed under direct sums. The same is true for the Descending Chain Condition, which can be seen by setting $v_i = (1, i)$ and $N_i = \langle\{v_0, \ldots, v_i\}\rangle$ for all $i \in \mathbb{N}$. Then the chain $N_0 \succ \cdots \succ N_i \succ \cdots$ does not terminate by the same argument as above.

(ii) If M and N are finite-length modules, then $\ell(M \oplus N) = \ell(M) + \ell(N)$.

(iii) If S has finite length, then every module generated by $n \in \mathbb{N}$ elements has length at most $n \cdot \ell(S)$.

Proof.

(i) The proof idea for the inequality $\ell(M) \leqslant \ell(N) + \ell(M/N)$ draws from a proof of the analogous fact for NOETHERIAN rings [17, 10f, Proposition 3.3]. For the sake of a contradiction, assume that $\ell(M) \geqslant r$, where $r = \ell(N) + \ell(M/N) + 1$. Then there exists a chain $L_0 \prec \cdots \prec L_r$ of strictly increasing submodules of M. In the corresponding chain

$$\frac{N + L_0}{N} \preceq \cdots \preceq \frac{N + L_r}{N}$$

of $r+1$ submodules of M/N, at most $\ell(M/N)$ inclusions are proper, so $\ell(N)+1$ inclusions are not. Similarly, in the chain $L_0 \cap N \preceq \cdots \preceq L_r \cap N$ of submodules of N, at most $\ell(N)$ inclusions are proper, so $\ell(M/N)+1$ inclusions are not. By the pigeonhole principle, there exists $0 \leqslant i < r$ such that

$$L_i \cap N = L_{i+1} \cap N \quad \text{and} \quad N + L_i = N + L_{i+1}.$$

We note that the latter result relies on the fact that $N \leqslant H \leqslant K$ and $H/N = K/N$ together imply $H = K$ (since if $k \in K$ and $[h] = [k]$ for some $h \in H$, then $k - h \in N \preceq H$, so $k = h + (k-h) \in H$).

Now, let $u \in L_{i+1}$ be arbitrary. By the second equation above, we have $u = n + v$ for some $n \in N$ and $v \in L_i$. This yields $n = u - v \in L_{i+1} \cap N = L_i \cap N$ and thus $u \in L_i$. Therefore, $L_{i+1} = L_i$, which is the desired contradiction.

Thus, we have shown that $\ell(M) \leqslant \ell(N) + \ell(M/N)$. It remains to show the converse inequality. Note that any submodule of M/N has the form L/N for some $L \preceq M$ such that $N \preceq L$ since N is the preimage of $[0] \in L/N$. This claim was already shown in a more general setting in the proof of Lemma 3.3. Therefore, let $N_0 \prec \cdots \prec N_n$ with $n = \ell(N)$ and $L_0/N \prec \cdots \prec L_m/N$ with $m = \ell(M/N)$ be chains of strictly increasing submodules of N and M/N, respectively, which exist by the definition of the respective length. These chains can be concatenated to obtain a chain

$$N_0 \prec \cdots \prec N_n \preceq L_0 \prec \cdots \prec L_m$$

of submodules of M. Any proper inclusion in the original chains must also be a proper inclusion in the concatenated chain. Thus, $\ell(M) \geqslant n + m = \ell(N) + \ell(M/N)$.

(ii) Let us consider $N_0 = \{(0,n) \mid n \in N\}$. Then $(M \oplus N)/N_0 \cong M$ and $N_0 \cong N$, which yields the claim by Statement (i).

(iii) Let M be a module generated by n elements. Hence M is a linear image of the free module S^n, which by iteration of Statement (ii) satisfies $\ell(S^n) = n \cdot \ell(S)$. Thus, the claim follows directly from Lemma 3.3. □

Hence every finite-length ring has the property that all its finitely generated modules also have finite length. Naturally, there are other semirings that enjoy this property. Trivially, every finitely generated semimodule over a finite semiring (such as the BOOLEAN semifield $\mathbb{B}$) is also finite and therefore of finite length. The following definition establishes the property just discussed, which is fulfilled in all rings and all finite semirings.

Definition 3.7. We say that S is ARTINIAN if every finitely generated semimodule has finite length. ■

As demonstrated in the proof of Theorem 3.6(iii), in order to establish that S is ARTINIAN it suffices to show that free semimodules of finite rank have finite length. Our naming ARTINIAN is a slight abuse of traditional notions since the term is usually used to characterize those modules that satisfy the Descending Chain Condition (i.e., every descending chain of submodules terminates). However, in rings these two notions coincide. Any ring that satisfies the Descending Chain Condition (DCC) also satisfies the Ascending Chain Condition (ACC) [5, p. 90, Theorem 8.5], and any module that satisfies both DCC and ACC has finite length [5, p. 77, Propositions 6.7 and 6.8]. By Theorem 3.6(iii), all finitely generated modules over a finite-length ring also have finite length. The converse implication is trivial. In general, for semirings this equivalence need not hold (see footnote to Lemma 3.5), but since the DCC is nowhere as important for semirings as it is for rings, the authors believe that our use of terminology is harmless.

ARTINIAN semirings retain a very convenient property of endomorphisms of vector spaces, which will be crucial for our approach.

Theorem 3.8. *Suppose that S is* ARTINIAN*, and let M be a finite-length semimodule and $\alpha \in \mathrm{End}(M)$. Then α is surjective if and only if α is injective.*

Proof. The proof simply combines the well-known facts that surjective endomorphisms of NOETHERIAN modules are injective, and injective endomorphisms of modules that satisfy the Descending Chain Condition (i.e., ARTINIAN in the traditional sense) are surjective. These two facts are established here for our semimodules.

We start with necessity. Suppose that α is surjective. For every endomorphism $\varphi \in \mathrm{End}(M)$, we let

$$\mathrm{Ker}\,\varphi = \{(u,v) \in M \oplus M \mid \varphi(u) = \varphi(v)\}.$$

Then $\mathrm{Ker}\,\varphi$ is a subsemimodule of $M \oplus M$ by the linearity of φ. Let $r = \ell(M \oplus M)$ and consider the chain

$$\{0\} \prec \mathrm{Ker}\,\alpha^0 \preceq \mathrm{Ker}\,\alpha^1 \preceq \cdots \preceq \mathrm{Ker}\,\alpha^r.$$

The first strictness is justified by $\mathrm{Ker}\,\alpha^0 = \mathrm{Ker}\,\mathrm{id}_M = \{(u,u) \mid u \in M\} \succ \{0\}$. Thus, by the finite length r, there exists some $0 \leqslant i < r$ such that $\mathrm{Ker}\,\alpha^i = \mathrm{Ker}\,\alpha^{i+1}$.

To prove injectivity, let $u, v \in M$ such that $\alpha(u) = \alpha(v)$. Recall that compositions of surjective functions are surjective. By the surjectivity of α and α^i, there exist $x, y \in M$ such that $\alpha^i(x) = u$ and $\alpha^i(y) = v$. Consequently, $\alpha^{i+1}(x) = \alpha^{i+1}(y)$ and thus $(x,y) \in \mathrm{Ker}\,\alpha^{i+1} = \mathrm{Ker}\,\alpha^i$ by our choice of i. However, $(x,y) \in \mathrm{Ker}\,\alpha^i$ directly yields $u = \alpha^i(x) = \alpha^i(y) = v$. Hence, α is injective.

We continue with sufficiency, so let α be injective. We show for all $j \in \mathbb{N}$ that the condition $u \notin \mathrm{im}\,\alpha^j$ implies $\alpha(u) \notin \mathrm{im}\,\alpha^{j+1}$. For the sake of a contradiction, suppose that $j \in \mathbb{N}$ and $u \in M \setminus \mathrm{im}\,\alpha^j$ are such that $\alpha(u) \in \mathrm{im}\,\alpha^{j+1}$. Clearly, there exists $v \in M$ such that $\alpha(u) = \alpha^{j+1}(v) = (\alpha\alpha^j)(v) = \alpha\big(\alpha^j(v)\big)$. Next we utilize the injectivity of α to conclude $u = \alpha^j(v)$, which yields $u \in \mathrm{im}\,\alpha^j$ and our desired contradiction. Thus, $\alpha(u) \notin \mathrm{im}\,\alpha^{j+1}$.

Suppose that α is not surjective. Then there exists $u \in M$ such that $u \notin \mathrm{im}\,\alpha$. A straightforward induction utilizing the statement proved in the previous paragraph can now be used to show that $\alpha^j(u) \notin \mathrm{im}\,\alpha^{j+1}$ for all $j \in \mathbb{N}$. However, this yields that the chain

$$M = \mathrm{im}\,\alpha^0 \succeq \mathrm{im}\,\alpha^1 \succeq \cdots \succeq \mathrm{im}\,\alpha^j \succeq \cdots$$

has infinitely many proper inclusions, which contradicts that M has finite length. Therefore, α must be surjective. We note that for sufficiency we only used that M has finite length (not that S is actually ARTINIAN). □

4 Pseudoregular Endomorphisms

At this point we have established sufficient background for our main notion, pseudoregular endomorphisms, that will be successfully utilized in our pumping lemmata. The special properties that define them are established in the next lemma.

Lemma 4.1 (see [20, Proposition 1])**.** *Let M be a semimodule and $\alpha \in \mathrm{End}(M)$. The following are equivalent:*

(i) $\mathrm{im}\,\alpha = \mathrm{im}\,\alpha^2$.

(ii) There exist $\gamma, \beta \in \mathrm{End}(M)$ such that $\alpha = \gamma\beta$ and $\mathrm{im}\,\beta = \mathrm{im}(\beta\gamma\beta)$.

Proof.

- We start with the implication (i) $\to$ (ii). To this end, we select $\gamma = \mathrm{id}_M$ and $\beta = \alpha$ and observe that

$$\alpha = \mathrm{id}_M\alpha = \gamma\beta \qquad \text{and} \qquad \mathrm{im}\,\beta = \mathrm{im}\,\alpha = \mathrm{im}\,\alpha^2 = \mathrm{im}(\alpha\mathrm{id}_M\alpha) = \mathrm{im}(\beta\gamma\beta).$$

- For the converse implication (ii) $\to$ (i), let $\gamma, \beta \in \mathrm{End}(M)$ such that $\alpha = \gamma\beta$ and $\mathrm{im}\,\beta = \mathrm{im}(\beta\gamma\beta)$. Then

$$\mathrm{im}\,\alpha^2 = \mathrm{im}(\gamma\beta\gamma\beta) = \gamma\big(\mathrm{im}(\beta\gamma\beta)\big) = \gamma(\mathrm{im}\,\beta) = \mathrm{im}(\gamma\beta) = \mathrm{im}\,\alpha. \qquad \square$$

Definition 4.2. Let M be a semimodule. An endomorphism $\alpha \in \mathrm{End}(M)$ satisfying the conditions of Lemma 4.1 is called *pseudoregular*. ■

REUTENAUER [20, Proposition 1] provides further characterizations of pseudoregular endomorphisms that hold for a field S. It is worthwhile to consider the following consequence. Let α be a nonzero pseudoregular endomorphism of a finite dimensional vector space V. Then there exists $k \leqslant \dim V$ and a basis $\mathscr{B}$ of V such that the matrix representation of α with respect to $\mathscr{B}$ is a block matrix

$$\begin{pmatrix} A & 0_{(n-k)\times k} \\ 0_{k\times(n-k)} & 0_{(n-k)\times(n-k)} \end{pmatrix},$$

where A is an invertible $k \times k$-matrix and $0_{m\times n}$ is the $m \times n$-zero matrix for every $m, n \in \mathbb{N}_+$.

Using Theorem 3.8 we can adapt another characterization mentioned in [20, Proposition 1] to ARTINIAN semirings.

Lemma 4.3. *Suppose that S is* ARTINIAN*, and let M be a semimodule that has finite length. Then $\alpha \in \mathrm{End}(M)$ is pseudoregular if and only if $\alpha_* \colon \mathrm{im}\,\alpha \to \mathrm{im}\,\alpha$, which is defined for every $u \in \mathrm{im}\,\alpha$ by $\alpha_*(u) = \alpha(u)$, is an isomorphism. If S is a ring, then this is equivalent to $\mathrm{im}\,\alpha \cap \ker\alpha = \{0\}$.*

Proof. Clearly, $\mathrm{im}\,\alpha = \mathrm{im}\,\alpha^2$ is equivalent to surjectivity of α_*, so the result follows from Theorem 3.8. If S is a ring, then $\mathrm{im}\,\alpha \cap \ker\alpha = \{0\}$ is equivalent to injectivity of α_*, and thereby surjectivity. $\square$

Next we show a generalization of [13, Theorem 2.2]. The general proof idea is largely unchanged, but the lack of vector space structure requires some adjustments in the details. The same theorem can be shown for vector spaces in a much more straightforward manner using linear recurrences (see [20, Lemma 1]), but as this proof relies on the existence of characteristic polynomials of endomorphisms, it cannot be directly adapted to more general semirings.

Theorem 4.4 (see [13, Theorem 2.2])**.** *Let M be a semimodule such that its dual $M^\vee$ has finite length. Moreover, let $\alpha \in \mathrm{End}(M)$ be pseudoregular, and let $f \in M^\vee = \mathrm{Hom}(M,S)$ and $v \in M$. We consider the sequence $(s_k)_{k\in\mathbb{N}}$ of elements of S given for every $k \in \mathbb{N}$ by*

$$s_k = f\big(\alpha^k(v)\big).$$

If $s_1 \neq 0$, then $s_k \neq 0$ for infinitely many $k \in \mathbb{N}$. More precisely, at most $\ell(M^\vee)$ values of s_k vanish in a row.

Proof. We prove this statement in three steps.

(i) As before, we define $\alpha_*\colon \mathrm{im}\,\alpha \to \mathrm{im}\,\alpha$ for every $u \in \mathrm{im}\,\alpha$ by $\alpha_*(u) = \alpha(u)$. Since α_* is surjective, we can find a right inverse $\alpha^*\colon \mathrm{im}\,\alpha \to \mathrm{im}\,\alpha$ such that $\alpha_*\alpha^* = \mathrm{id}_{(\mathrm{im}\,\alpha)}$.[2] Next, we define ρ to be the map that sends each element $g\colon M \to S$ of $M^\vee$ to its restriction $g|_{\mathrm{im}\,\alpha}$ to $\mathrm{im}\,\alpha$; i.e.,

$$\rho\colon M^\vee \to (\mathrm{im}\,\alpha)^\vee \qquad \text{with} \qquad \rho(g) = g|_{\mathrm{im}\,\alpha}$$

for all $g \in M^\vee = \mathrm{Hom}(M,S)$. Clearly, ρ is linear, so $\mathrm{im}\,\rho$ has finite length by Lemma 3.3. Fix some $n_0 \in \mathbb{N}_+$ and let $f_i = \rho(f\alpha^{n_0+i})$ for every $i \in \mathbb{N}$. Then

$$f_i = \rho(f\alpha^{n_0+i}) = \rho(f\alpha^{n_0+i})\mathrm{id}_M = \rho(f\alpha^{n_0+i})\alpha_*\alpha^* = \rho(f\alpha^{n_0+i+1})\alpha^* = f_{i+1}\alpha^*$$

for every $i \in \mathbb{N}$.

(ii) Let $r = \ell(\mathrm{im}\,\rho) + 1$ and $M_i = \langle\{f_r, \ldots, f_{r-i}\}\rangle$ be the subsemimodule of $\mathrm{im}\,\rho$ that is generated by $\{f_r, \ldots, f_{r-i}\}$ for every $0 \leqslant i \leqslant r$. We consider the chain $M_0 \preceq M_1 \preceq \ldots \preceq M_r$. Since $r > \ell(\mathrm{im}\,\rho)$, at least one of these inclusions is not proper. Let $0 < i \leqslant r$. If $M_{i-1} = M_i$, then $M_i = M_{i+1}$, which we prove as follows. Since $M_i = M_{i-1}$, there exist coefficients $\lambda_0, \ldots, \lambda_r \in S$ such that

$$f_{r-i} = \sum_{j=0}^{i-1} \lambda_j f_{r-j}$$

and thus

$$f_{r-(i+1)} = f_{r-i-1} = f_{r-i}\alpha^* = \Big(\sum_{j=0}^{i-1} \lambda_j f_{r-j}\Big)\alpha^* = \sum_{j=0}^{i-1} \lambda_j f_{r-j-1} = \sum_{j=1}^{i} \lambda_{j-1} f_{r-j}.$$

Therefore, $f_{r-(i+1)} \in \langle\{f_{r-1}, \ldots, f_{r-i}\}\rangle \preceq M_i$, so we have $M_{i+1} = M_i$ by the construction of M_i. A straightforward induction then proves that $M_r = M_{r-1}$. Hence, there are coefficients $\mu_1, \ldots, \mu_r \in S$ such that

$$f_0 = \sum_{j=1}^{r} \mu_j f_j. \tag{$\dagger$}$$

(iii) Finally, let $s_1 \neq 0$. Assume by way of contradiction that there are only finitely many $k \in \mathbb{N}$ such that $s_k \neq 0$. Then there is some $n \in \mathbb{N}$ such that $s_n \neq 0$ and $s_k = 0$ for all $k > n$. In particular, $s_{n+1} = \cdots = s_{n+r} = 0$. Set $n_0 = n-1$ and define f_i as above. Then

$$s_n = f_0\big(\alpha(v)\big) = \Big(\sum_{j=1}^{r} \mu_j f_j\Big)\big(\alpha(v)\big) = \sum_{j=1}^{r} \mu_j f_j\big(\alpha(v)\big) = \sum_{j=1}^{r} \mu_j s_{n+j} = 0$$

[2] In the most general setting, finding a right inverse of a surjective function requires the Axiom of Choice. In all cases of interest to us, this is not necessary. If S is ARTINIAN, then α_* is bijective, so there is a unique both-sided linear inverse. If $\mathrm{im}\,\alpha$ is free of finite rank, then it suffices to choose finitely many preimages for the free generators of $\mathrm{im}\,\alpha$.

by (†), which contradicts the choice of n. Therefore, there must be infinitely many $k \in \mathbb{N}$ such that $s_k \neq 0$. In particular, we have shown that at most $r-1 = \ell(\operatorname{im}\rho) \leqslant \ell(M^\vee)$ values of s_k can vanish in a row. □

We note that the previous proof relies crucially on the commutativity of S, since $M^\vee$ need not be a semimodule in the non-commutative case. Semimodules of finite length allow us to determine that an endomorphism is pseudoregular simply by looking at its factorizations. We will later use a statement of this kind for the proof of our pumping lemma. However, one similar proposition can already be adapted directly from the theory of vector spaces without any further work.

Lemma 4.5 (see [13, Proposition 2.1])**.** *Let M be a finite-length semimodule and $\alpha \in \operatorname{End}(M)$. Then $\alpha^{\ell(M)}$ is pseudoregular.*

Proof. Consider the chain

$$M = \operatorname{im}\alpha^0 \succeq \operatorname{im}\alpha^1 \succeq \operatorname{im}\alpha^2 \succeq \ldots \succeq \operatorname{im}\alpha^{\ell(M)} \succeq 0.$$

By definition at least one of these inclusions is not proper. Let $0 < i \leqslant \ell(M)$. If $\operatorname{im}\alpha^{i-1} = \operatorname{im}\alpha^i$, then indeed also $\operatorname{im}\alpha^i = \operatorname{im}\alpha^{i+1}$, so by another straightforward induction we also obtain $\operatorname{im}\alpha^{\ell(M)} = \operatorname{im}\alpha^{2\ell(M)}$, which yields that $\alpha^{\ell(M)}$ is pseudoregular. If only the last inclusion is improper (i.e., $\operatorname{im}\alpha^{\ell(M)} = \{0\}$), then $\alpha^{\ell(M)}$ is the zero morphism and thereby trivially pseudoregular as well. This concludes all cases and in each case $\alpha^{\ell(M)}$ is pseudoregular. □

5 Pumping Lemmata

In this final section, we combine our derived results to provide a pumping lemma for recognizable weighted languages. In general, pumping lemmata are used to prove that a (weighted) language is not recognizable. For illustration, we recall the classical pumping lemma for recognizable languages, which is the main tool to prove that a given language is not recognizable [28].

Theorem 5.1 (see [19, Lemma 2])**.** *Let L be a recognizable language. Then there exists $n \in \mathbb{N}$ such that for every $w \in L$ with $|w| \geqslant n$ there is a factorization $w = uxv$ with $x \neq \varepsilon$ such that $ux^kv \in L$ for all $k \in \mathbb{N}$.*

Next, we show a similar result for recognizable weighted languages, which was originally proven for fields in [13, Theorem 5], although we adapted our proof using the ideas of [20, Theorem 2]. These ideas directly yield the basic approach using our Theorem 4.4. Given a linear representation $(Q, \mathrm{in}, \mathrm{out}, \mu)$ of a weighted language $L\colon \Sigma^* \to S$ such that (i) S^Q has finite length, (ii) $\mu(x)$ is pseudoregular for some $x \in \Sigma^*$, and (iii) $uxv \in \operatorname{supp} L$ for some $u, v \in \Sigma^*$, then for infinitely many $k \in \mathbb{N}$,

$$L(ux^kv) = \mathrm{in} \cdot \mu(ux^kv) \cdot \mathrm{out} \neq 0.$$

We use that $S^Q \cong (S^Q)^\vee$; i.e., that S^Q and $(S^Q)^\vee$ are isomorphic, yielding finite length for $(S^Q)^\vee$. Additionally, we note that we do not conclude that $L(ux^kv) \neq 0$ for all $k \in \mathbb{N}$ (as in Theorem 5.1), but rather the inequality only holds for infinitely many $k \in \mathbb{N}$. However, to make this approach applicable to any recognizable weighted language, we still need to identify suitable conditions that enforce that a given word $w \in \Sigma^*$ contains a nontrivial subword $x \in \Sigma^*$ with pseudoregular image $\mu(x)$. A simple combinatorial argument following [20] shows that if w is long enough, then there always exists a factorization $w = uxy$ with $x \neq \varepsilon$ such that $\mu(x)$ is pseudoregular.

Definition 5.2 (see [20]). Let Σ be a finite alphabet, $w \in \Sigma^*$, and $n \in \mathbb{N}$. We recursively define when w is a *quasipower of order n*.

(i) If $n = 0$ and $w \neq \varepsilon$, then w is a quasipower of order 0.

(ii) If $n > 0$ and $w = uvu$ for some $u, v \in \Sigma^*$ such that u is a quasipower of order $n-1$, then w is a quasipower of order n. ■

Next, we recall that given any order $r \in \mathbb{N}$ we can identify a bound N_r such that words whose length is at least N_r necessarily contain a quasipower of order r. Indeed the constant N_r can be recursively defined for every $r \in \mathbb{N}$ by

$$N_0 = 1 \qquad \text{and} \qquad N_{r+1} = N_r \cdot (1 + |\Sigma|^{N_r}).$$

Lemma 5.3 (see [23, IV. 5] as cited in [20, Lemma 2]). *Let Σ be a finite alphabet and $r \in \mathbb{N}$. There exists an integer $N_r \in \mathbb{N}$ such that every word $w \in \Sigma^*$ with $|w| \geqslant N_r$ contains a subword that is a quasipower of order r.*

Next, still following [20], we show that quasipowers of suitably large order are sufficient to establish the existence of a subword x such that $\mu(x)$ is pseudoregular.

Lemma 5.4 (see [13] as cited in [20, Theorem 1]). *Let Σ be a finite alphabet, M a semimodule that has finite length, and $\mu : \Sigma^* \to \mathrm{End}(M)$ a monoid homomorphism. Every word $w \in \Sigma^*$ that is a quasipower of order $r = \ell(M) + 1$ contains a subword $x \neq \varepsilon$ such that $\mu(x)$ is pseudoregular.*

Proof. Let $w \in \Sigma^*$ be a quasipower of order r, and let $u_r = w$. There are words $u_0, \ldots, u_{r-1}, v_1, \ldots, v_n \in \Sigma^+$ such that $u_i = u_{i-1} v_i u_{i-1}$ for all $1 \leqslant i \leqslant r$. Thus,

$$\operatorname{im}\mu(u_i) = \operatorname{im}\mu(u_{i-1} v_i u_{i-1}) \preceq \operatorname{im}\mu(u_{i-1}),$$

so we obtain the chain

$$\operatorname{im}\mu(u_r) \preceq \operatorname{im}\mu(u_{r-1}) \preceq \ldots \preceq \operatorname{im}\mu(u_0)$$

of $r+1$ subsemimodules of M. Therefore, $\operatorname{im}\mu(u_i) = \operatorname{im}\mu(u_{i-1})$ for some $1 \leqslant i \leqslant r$, which yields

$$\operatorname{im}\mu(u_{i-1}) = \operatorname{im}\mu(u_i) = \operatorname{im}\mu(u_{i-1} v_i u_{i-1}) = \operatorname{im}\big(\mu(u_{i-1})\mu(v_i)\mu(u_{i-1})\big).$$

By Lemma 4.1(ii) we obtain that $\mu(v_i)\mu(u_{i-1}) = \mu(v_i u_{i-1})$ is pseudoregular. Hence, we set $x = v_i u_{i-1}$ to complete the proof. □

Our pumping lemma now follows directly. The next main theorem still contains the technical restriction that the semimodule S^Q has finite length, where Q is the set of states of a linear representation for a given recognizable weighted language. A slightly more direct statement is expressed in the corollary that follows the next theorem.

Theorem 5.5 (see [13, Theorem 5] as cited in [20, Theorem 2]). *Let Σ be a finite alphabet. Moreover, let $(Q, \mathrm{in}, \mathrm{out}, \mu)$ be a linear representation for the weighted language $L : \Sigma^* \to S$. If S^Q has finite length, then there exists an integer $N \in \mathbb{N}$ such that for every $w \in \operatorname{supp} L$ with $|w| \geqslant N$ there exists a factorization $w = uxv$ with $x \neq \varepsilon$ such that*

$$\{u x^k v \mid k \in \mathbb{N}\} \cap \operatorname{supp} L$$

is infinite.

Proof. Let $r = \ell(M) + 1$ and $N = N_r$ as in Lemma 5.3. Since $|w| \geqslant N$, the word w contains a quasipower of order r by Lemma 5.3, and by Lemma 5.4 there exists a factorization $w = uxv$ such that $x \neq \varepsilon$ and $\mu(x)$ is pseudoregular. Moreover, $\mathrm{in} \cdot \mu(u) \in (S^Q)^\vee$ and $\mu(v) \cdot \mathrm{out} \in S^Q$. By assumption we have

$$L_w = \mathrm{in} \cdot \mu(u)\mu(x)\mu(v) \cdot \mathrm{out} \neq 0.$$

Since $S^Q \cong (S^Q)^\vee$ and S^Q has finite length, we can apply Theorem 4.4 to obtain that for infinitely many $k \in \mathbb{N}$,

$$\big(\mathrm{in} \cdot \mu(u)\big) \cdot \mu(x)^k \cdot \big(\mu(v) \cdot \mathrm{out}\big) \neq 0.$$

Since $\mu(x)^k = \mu(x^k)$, this completes the proof. □

By extending this theorem from its original statement for fields to more general semirings, we have identified a unified framework for the classical pumping lemma by RABIN and SCOTT [19] (for the BOOLEAN semifield) and the pumping lemma for recognizable weighted languages over fields by JACOB [13]. In practice, it is useful to be able to reason about recognizability without knowing the number of states a potential linear representation might have, which makes the requirement that S^Q has finite length troublesome. This can be remedied by requiring our semiring S to be ARTINIAN, which of course still subsumes all the cases covered by the already mentioned pumping lemmata.

Corollary (of Theorem 5.5)**.** *Let Σ be a finite alphabet, S be an* ARTINIAN *semiring, and L be a recognizable weighted language $L\colon \Sigma^* \to S$. Then there exists an integer $N \in \mathbb{N}$ such that for every $w \in \operatorname{supp} L$ with $|w| \geqslant N$ there exists a factorization $w = uxv$ with $x \neq \varepsilon$ such that*

$$\{ux^k v \mid k \in \mathbb{N}\} \cap \operatorname{supp} L$$

is infinite.

Example 5.6. Directly generalizing a classical example of a non-regular language, there is no recognizable weighted language L over an ARTINIAN semiring such that $\operatorname{supp} L = \{a^n b^n \mid n \in \mathbb{N}\}$. Suppose that there is an ARTINIAN semiring S and a recognizable weighted language $L\colon \{a,b\}^* \to S$ such that $\operatorname{supp} L = \{a^n b^n \mid n \in \mathbb{N}\}$. By the Corollary of Theorem 5.5 there exists $N \in \mathbb{N}$ such that $w = a^N b^N$ admits a decomposition $w = uxv$ with $x \neq \varepsilon$ such that $\{ux^k v \mid k \in \mathbb{N}\} \cap \operatorname{supp} L$ is infinite. Obviously this is a contradiction since no suitable subword $x \neq \varepsilon$ (consider the cases $x = a^m$, $x = b^m$, and $x = a^m b^n$) exists. We note that such a recognizable weighted language L over a *non-commutative* semiring exists. ■

If we drop the assumption that the alphabet Σ is finite, then we obtain a notion of recognizable weighted languages that is useful when applying the same pumping techniques to weighted tree languages (see, for example, [8, Theorem 9.2]). One result that would be an ideal candidate for extension to semimodules is recalled next. Its extension would immediately yield pumping lemmata of various forms (e.g. [20, Theorem 4]).

Theorem 5.7 (see [20, Theorem 3])**.** *Let Σ be a (not necessarily finite) alphabet and V a vector space of finite nonzero dimension. There is an integer N such that for each homomorphism $\mu\colon \Sigma^* \to \mathrm{End}(V)$, every word $w \in \Sigma^*$ with $|w| \geqslant N$ contains a subword $x \neq \varepsilon$ such that $\mu(x)$ is pseudoregular.*

Unfortunately, the proof of this theorem uses the relationship of nonvanishing elements of exterior powers of V to their components' linear independence. This cannot be easily extended even to (non-integral) rings. We conclude this section by stating two weak pumping lemmata for recognizable weighted languages over infinite alphabets.

Theorem 5.8. *Let Σ be a (possibly infinite) alphabet and $L\colon \Sigma^* \to S$ be a recognizable weighted language with linear representation $(Q, \mathrm{in}, \mathrm{out}, \mu)$ such that S^Q has finite length $N = \ell(S^Q)$. If there exists $w \in \operatorname{supp} L$ with $w = ab^N c$, then the set $\{ab^k c \mid k \in \mathbb{N}\} \cap \operatorname{supp} L$ is infinite.*

Proof. By Lemma 4.5, $\mu(b^N) = \mu(b)^N$ is pseudoregular. Then the claim follows exactly as in Theorem 5.5. □

Theorem 5.9. *Let the semiring S be finite, Σ a (possibly infinite) alphabet, and $L\colon \Sigma^* \to S$ be a recognizable weighted language with linear representation $(Q, \mathrm{in}, \mathrm{out}, \mu)$. There is an integer N such that for every $w \in \operatorname{supp} L$ with $|w| \geqslant N$ there exists a factorization $w = uxv$ with $x \neq \varepsilon$ such that*

$$\{ux^k v \mid k \in \mathbb{N}\} \cap \operatorname{supp} L$$

is infinite.

Proof. Since $\mathrm{End}(S^Q)$ is finite, we can reduce to the case of finite alphabets. To this end, we define the relation $\sim = \operatorname{Ker} \mu$ on Σ (where $\operatorname{Ker} \mu$ is defined as in Theorem 3.8). Clearly, $\sim$ is an equivalence relation. From each of the finite number of equivalence classes $[m]$ we choose a representative r_m. Now, we let $\Gamma = \{r_m \mid m \in \Sigma\}$ and extend the mapping $m \mapsto r_m$ to the unique monoid homomorphism $\psi\colon \Sigma^* \to \Gamma^*$. By definition of $\sim$, it is obvious that $\mu(w) = \mu(\psi(w))$ for all $w \in \Sigma^*$.

Since $|\mathrm{End}(S^Q)| \leqslant |S|^{|Q|^2}$ (consider matrices), we have $|\Gamma| \leqslant |S|^{|Q|^2}$. Let N be as in Theorem 5.5. For every $w \in \Sigma^*$ with $|w| \geqslant N$. there exists a factorization $w = uxv$ with $x \neq \varepsilon$ and infinite

$$\{\psi(ux^k v) \mid k \in \mathbb{N}\} \cap \operatorname{supp} L.$$

By definition of ψ, it is clear that this implies the infiniteness of the set

$$\{ux^k v \mid k \in \mathbb{N}\} \cap \operatorname{supp} L.$$ □

References

[1] Alfred V. Aho, Ravi Sethi & Jeffrey D. Ullman (1985): *Compilers: Principles, Techniques, and Tools*. Addison-Wesley.

[2] Alfred V. Aho & Jeffrey D. Ullman (1992): *Foundations of Computer Science*. W.H. Freeman.

[3] Jürgen Albert & Jarkko Kari (2009): *Digital Image Compression*. In *Handbook of Weighted Automata* [9], chapter 11, pp. 453–479, doi:10.1007/978-3-642-01492-5_11.

[4] Paolo Aluffi (2009): *Algebra: Chapter 0*. *Graduate Studies in Mathematics* 104, American Mathematical Society, doi:10.1090/gsm/104/01.

[5] Michael F. Atiyah & Ian G. MacDonald (1994): *Introduction To Commutative Algebra*. Addison-Wesley Series in Mathematics, Avalon Publishing.

[6] Jean-Michel Autebert, Jean Berstel & Luc Boasson (1997): *Context-free Languages and Pushdown Automata*. In Grzegorz Rozenberg & Arto Salomaa, editors: *Handbook of Formal Languages*, chapter 3, 1, Springer, pp. 111–174, doi:10.1007/978-3-642-59136-5_3.

[7] Christel Baier, Marcus Größer & Frank Ciesinski (2009): *Model Checking Linear-Time Properties of Probabilistic Systems*. In *Handbook of Weighted Automata* [9], chapter 13, pp. 519–570, doi:10.1007/978-3-642-01492-5_13.

[8] Jean Berstel & Christophe Reutenauer (1982): *Recognizable Formal Power Series on Trees*. *Theoretical Computer Science* 18(2), pp. 115–148, doi:10.1016/0304-3975(82)90019-6.

[9] Manfred Droste, Werner Kuich & Heiko Vogler (2009): *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/978-3-642-01492-5.

[10] Manfred Droste & Dietrich Kuske (2021): *Weighted Automata*. In Jean-Eric Pin, editor: *Handbook of Automata Theory*, chapter 4, 1, EMS Press, pp. 113–150, doi:10.4171/automata-1/4.

[11] Jonathan S. Golan (1999): *Semirings and their Applications*. Kluwer Academic, Dordrecht, doi:10.1007/978-94-015-9333-5.

[12] Udo Hebisch & Hanns J. Weinert (1998): *Semirings—Algebraic Theory and Applications in Computer Science*. *Series in Algebra* 5, World Scientific, doi:10.1142/3903.

[13] Gerard Jacob (1980): *Un théorème de factorisation des produits d'endomorphismes de K^n*. *Journal of Algebra* 63(2), pp. 389–412, doi:10.1016/0021-8693(80)90080-0.

[14] Daniel Kirsten (2011): *The Support of a Recognizable Series over a Zero-sum Free, Commutative Semiring is Recognizable*. *Acta Cybernetica* 20(2), pp. 211–221, doi:10.14232/actacyb.20.2.2011.1.

[15] Kevin Knight & Jonathan May (2009): *Applications of Weighted Automata in Natural Language Processing*. In *Handbook of Weighted Automata* [9], chapter 14, pp. 571–596, doi:10.1007/978-3-642-01492-5_14.

[16] Werner Kuich & Arto Salomaa (1986): *Semirings, Automata, Languages*. Monographs in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/978-3-642-69959-7.

[17] James Milne (2020): *A Primer of Commutative Algebra*. https://www.jmilne.org/math/xnotes/CA.pdf.

[18] Antonio Munoz, Sakir Sezer, Dwayne Burns & Gareth Douglas (2011): *An Approach for Unifying Rule Based Deep Packet Inspection*. In: *Proc. IEEE Int. Conf. on Communications*, IEEE, pp. 1–5, doi:10.1109/icc.2011.5963095.

[19] Micheal O. Rabin & Dana Scott (1959): *Finite Automata and Their Decision Problems*. *IBM Journal of Research and Development* 3(2), pp. 114–125, doi:10.1147/rd.32.0114.

[20] Christophe Reutenauer (1980): *An Ogden-like Iteration Lemma for Rational Power Series*. *Acta Informatica* 13(2), pp. 189–197, doi:10.1007/bf00263993.

[21] Jacques Sakarovitch (2009): *Rational and Recognisable Power Series*. In *Handbook of Weighted Automata* [9], chapter 4, pp. 105–174, doi:10.1007/978-3-642-01492-5_4.

[22] Arto Salomaa & Matti Soittola (1978): *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science, Springer, doi:10.1007/978-1-4612-6264-0.

[23] Marcel-Paul Schützenberger (1961): *On a Special Class of Recurrent Events*. *The Annals of Mathematical Statistics* 32(4), pp. 1201–1213, doi:10.1214/aoms/1177704860.

[24] Marcel-Paul Schützenberger (1961): *On the Definition of a Family of Automata*. *Information and Control* 4(2–3), pp. 245–270, doi:10.1016/S0019-9958(61)80020-X.

[25] Mark P. J. van der Loo & Edwin de Jonge (2021): *Data Validation Infrastructure for R*. *Journal of Statistical Software* 97(10), pp. 1–31, doi:10.18637/jss.v097.i10.

[26] Gunnar von Heijne (1987): *Sequence Analysis in Molecular Biology*. Academic Press, doi:10.1016/B978-0-12-725130-1.X5001-2.

[27] Huaxiong Wang (1997): *On Characters of Semirings*. *Houston Journal of Mathematics* 23(3), pp. 391–405.

[28] Sheng Yu (1997): *Regular Languages*. In Grzegorz Rozenberg & Arto Salomaa, editors: *Handbook of Formal Languages*, chapter 2, 1, Springer, pp. 41–110, doi:10.1007/978-3-642-59136-5_2.

State-deterministic Finite Automata with Translucent Letters and Finite Automata with Nondeterministically Translucent Letters

Benedek Nagy
Department of Mathematics, Faculty of Arts and Sciences
Eastern Mediterranean University
99628 Famagusta, North Cyprus, Mersin-10, Turkey
and
Department of Computer Science, Institute of Mathematics and Informatics,
Eszterházy Károly Catholic University
Eger, Hungary
nbenedek.inf@gmail.com

Deterministic and nondeterministic finite automata with translucent letters were introduced by Nagy and Otto more than a decade ago as Cooperative Distributed systems of a kind of stateless restarting automata with window size one. These finite state machines have a surprisingly large expressive power: all commutative semi-linear languages and all rational trace languages can be accepted by them including various not context-free languages. While the nondeterministic variant defines a language class with nice closure properties, the deterministic variant is weaker, however it contains all regular languages, some non-regular context-free languages, as the Dyck language, and also some languages that are not even context-free. In all those models for each state, the letters of the alphabet could be in one of the following categories: the automaton cannot see the letter (it is translucent), there is a transition defined on the letter (maybe more than one transition in nondeterministic case) or none of the above categories (the automaton gets stuck by seeing this letter at the given state and this computation is not accepting).

State-deterministic automata are recent models, where the next state of the computation determined by the structure of the automata and it is independent of the processed letters. In this paper our aim is twofold, on the one hand, we investigate state-deterministic finite automata with translucent letters. These automata are specially restricted deterministic finite automata with translucent letters.

In the other novel model we present, it is allowed that for a state the set of translucent letters and the set of letters for which transition is defined are not disjoint. One can interpret this fact that the automaton has a nondeterministic choice for each occurrence of such letters to see them (and then erase and make the transition) or not to see that occurrence at that time. Based on these semi-translucent letters, the expressive power of the automata increases, i.e., in this way a proper generalization of the previous models is obtained.

Keywords: finite state machines, automata with translucent letters, determinism vs. nondeterminism, state-determinism

1 Introduction

The history of automata with translucent letters has begun using the technical name cooperative distributed systems of stateless restarting automata with window size one [27], while the term finite state acceptors with translucent letters appeared in [16] reinterpreting the aforementioned technical name. Basically (formal definitions will be recalled in Section 2), in a finite automaton with translucent letters,

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 170–184, doi:10.4204/EPTCS.386.14

in each state some of the letters of the alphabet are translucent, and the automaton sees the first occurrence of a non-translucent letter (after the occurrences of translucent letters in the prefix of the remaining input in the given configuration, if any) and if there is a transition defined on this letter (say, the letter is readable) in the actual state, after erasing this letter, the next state is chosen according to the transition function, and the computation continues. It may happen that there are only translucent letters (or no letters at all) in the remained input, then the computation is accepting if the actual state is a final state. Automata with translucent letters can be applied in linguistics [26], and also modelling various trace languages used to describe parallel processes [4, 10, 13, 25] based on commutations and partial commutations [2, 18].

In fact, there are various models in automata theory where the processing on the input may not go strictly left to right. One of these models is the restarting automata which is developed for linguistical motivation to do analysis by reduction: in a nutshell, these automata have a read-write window and they are searching for some specific pattern in the window to reduce, i.e., shorten its content, and then they are restarting the computation on the new content of the tape. It may also happen that the automaton accepts based on what is in its window. Interested readers may be referred to [9, 36] to see the various models, their computations, accepted languages and their properties. Restarting R automata with window size one can do only one type of reduction, to erase the letter in the window, hereby shortening the tape. Stateless deterministic variants of them are the simplest models of restarting automata. Instead of adding states to the system, their cooperative distributed systems (shortly CD systems) are developed [27, 30] and found to be very interesting with a surprisingly large expressive power as, e.g., they are able to accept all rational trace languages. The components of such systems play the role of the states in the reinterpreted model, in the nondeterministic finite automata with translucent letter. Two types of deterministic models of the CD systems of restarting R automata with window size one are also studied [29]: In strictly deterministic models, the next component is uniquely defined by the actual component and it does not depend on the letter being processed (i.e., erased) in the actual computation step. However, in globally deterministic CD systems of restarting R automata with window size one the next component is deterministically chosen based on the actual component and on the erased letter, somewhat similarly as it is in the usual deterministic finite automata. Consequently, this letter model is equivalent, by the reinterpretation, to the deterministic finite automata with translucent letters.

Other models not consuming the input from left to right are various 2-head models that process the input parallely from both extremes [6, 12, 17, 20, 21, 39]. Some of these finite state models are capable to accept exactly the linear context-free languages. Moreover, in the bio-inspired models named $5' \to 3'$ Watson-Crick finite automata, the automaton with its both heads may read strings in a computation step [23, 34, 35]. There are various interesting concepts related to determinism introduced and studied for these $5' \to 3'$ Watson-Crick finite automata models. The deterministic variant where the concept of determinism fits well to the usual concept of determinism is less powerful in the sense that only a proper subset of the class of the linear context-free languages can be accepted by them. This class is called 2detLIN, and it is incomparable with the class detLIN containing the languages that are accepted by deterministic one-turn pushdown automata [33, 37]. The model where the next state is uniquely defined by the actual state and does not depend on what is being processed from the input in this step of computation is called state-deterministic and studied in [22]. At $5' \to 3'$ Watson-Crick finite automata, the state-deterministic variants are very restricted, but they may do some nondeterministic computations, and thus, the language class accepted by them is incomparable with 2detLIN. Other type of determinism, the quasi-determinism is introduced and studied in [24]. In these automata, even if the state of the next configuration is uniquely determined by the actual configuration (actual state and remaining input), there could be more than one possible next configuration. The quasi-deterministic $5' \to 3'$ Watson-Crick finite

automata accept a superclass of languages of both the classes accepted by state-deterministic and by deterministic $5' \to 3'$ Watson-Crick finite automata.

There are other models of computations which process the input not strictly left to right, including various jumping automata [3, 14, 11] and input revolving automata [1], just to mention a few more models. These models became very popular in the last decades. The combination of the mentioned 2-head finite state model with translucent letters allow to accept all linear trace languages [32]. Pushdown automata with translucent letters can be used to characterise context-free trace languages [28].

In this paper, on the one hand, we investigate the state-deterministic finite automata with translucent letters and give some results on the class of languages accepted by them.

On the other hand, we investigate finite automata with translucent letters by relaxing the condition that for a state the set of translucent and readable letters is disjoint. In this way, the translucency becomes nondeterministic and thus, we may expand both the deterministic and nondeterministic finite automata with translucent letters to allow nondeterministic translucency. Our other main result is that we can show that this model is a real expansion of the basic models, the class of accepted languages is a superclass of the class languages of the original model.

Recently another extension of the finite automata with translucent letters was investigated in which in the computation the head is not restarting after erasing a symbol, but continues from the position where this letter has been erased [15]. We believe that our new type of restrictions and extensions are also giving some new interesting insights and results to this particular field of automata theory.

The structure of the paper is as follows. In the next section we recall some formal preliminaries and the basic definitions of finite automata with translucent letters. Section 3 is devoted to a restricted class of deterministic models, namely, to the state-deterministic finite automata with translucent letters; while in Section 4 we present our other new concept by allowing nondeterminism based on translucency. We show that this model is more powerful than the original model, however, still only semi-linear languages can be accepted. Finally, conclusions close the paper.

2 Preliminaries

We assume that the reader is familiar with the basic concepts of formal languages and automata [7, 8], however, to fix our notations, we formally recall some basic definitions. We denote the empty word by λ.

We say that the languages L_1 and L_2 are letter equivalent, if for any word $x \in L_1$ we may find a word $y \in L_2$ such that y is obtained from x by reordering (permuting) its letters and also for any word $x \in L_2$ we may find a word $y \in L_1$ with the same property. It is known that a language is *semi-linear* if there is a regular language that is letter equivalent with it. All context-free languages are semi-linear [38] and there are context-sensitive languages that are not semi-linear. We do not detail here partial commutations, commutations, traces and trace languages, interested readers may be referred to [2, 4] and for their relations to automata with translucent letters to [27, 28, 32].

A *nondeterministic finite automaton* (NFA) is a pentuple $A = (Q, \Sigma, I, F, \delta)$, where Q is the finite set of internal states, Σ is the finite alphabet containing the input letters, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final (or accepting) states, and $\delta : Q \times \Sigma \to 2^Q$ is the transition relation. If $|I| = 1$ and $|\delta(q,a)| \leq 1$ holds for all $q \in Q$ and all $a \in \Sigma$, then A is a *deterministic finite automaton* (DFA). Notice that, in general, in NFAs we allow multiple initial states, but we do not allow transitions by the empty word.

An NFA A works as follows. Let an input string $w \in \Sigma^*$ be given, then A starts its computation in

a state q_0 that is chosen nondeterministically from the set I of all initial states. This configuration is encoded as q_0w (for simplicity, we may assume that $Q \cap \Sigma = \emptyset$). Now A reads the first letter of w, say a (let $w = au$), thereby deleting (consuming) this occurrence of letter a, and it changes its internal state to a state q_1 that is chosen nondeterministically from the set $\delta(q_0,a)$, formally we may write that the new configuration q_1u is reached. However, it may happen that $\delta(q_0,a)$ is empty, then A gets stuck and this computation fails in this input. Otherwise, A continues the computation from the configuration q_1u by reading the input letter by letter until either w has been consumed completely or the computation fails (similarly as we have described). We say that A accepts w from the initial configuration q_0w if it reaches a configuration $q_f \cdot \lambda$ in a computation starting from q_0w, where $q_f \in F$ is a final state. By $L(A)$ we denote the set of all strings $w \in \Sigma^*$ for which A has an accepting computation in the sense described above.

It is well-known that the class of languages that are accepted by NFAs coincides with the class of regular languages, and that DFAs accept exactly the same languages.

Now we recall the nondeterministic finite automata with translucent letters from [16].

A *finite state automaton with translucent letters* (NFAwtl) is defined as $A = (Q,\Sigma,\$,\tau,I,F,\delta)$, where Q, Σ, I and F are the same as at an NFA; $\$ \notin \Sigma$ is a special symbol that is used technically as an *endmarker*, $\tau : Q \to 2^{\Sigma}$ is the *translucency mapping*, and $\delta : Q \times \Sigma \to 2^{Q}$ is the *transition relation* that satisfies the following condition:

$$\forall q \in Q \, \forall a \in \tau(q) : \delta(q,a) = \emptyset.$$

For each state $q \in Q$, the letters from the set $\tau(q)$ are translucent for q, that is, in state q the automaton A does not see these letters. A is called *deterministic finite state automaton with translucent letters*, abbreviated as DFAwtl, if $|I| = 1$ and if $|\delta(q,a)| \leq 1$ for all $q \in Q$ and all $a \in \Sigma$.

An NFAwtl $A = (Q,\Sigma,\$,\tau,I,F,\delta)$ works as follows. Let $w \in \Sigma^*$ be an input word. A starts in a nondeterministically chosen initial state $q \in I$ with the word $w \cdot \$$ on its input tape, that is $q_0w\$$ is an initial configuration. A computation step of A is defined as follows. Assume that $w = a_1a_2 \cdots a_n$ for some $n \geq 1$ and $a_1,\ldots,a_n \in \Sigma$. Then A looks for the first occurrence from the left of a letter that is not translucent (say visible) for the current state q, more precisely, if $w = uav$ such that $u \in (\tau(q))^*$ and $a \notin \tau(q)$, then A nondeterministically chooses a state $q' \in \delta(q,a)$, erases the letter a from the tape thus producing the tape contents $uv \cdot \$$, and its internal state is set to q'. Therefore after this computation step the configuration is $q'uv\$$ and the computation continues from this configuration by looking for the first visible letter of uv at state q'. However, it may happen that $\delta(q,a) = \emptyset$ for the first visible letter a, A halts without accepting, this computation fails. Finally, if $w \in (\tau(q))^*$ for a configuration $qw\$$ (including the possibility that the configuration is in fact $q \cdot \lambda \cdot \$$), then A reaches the \$-symbol and the computation halts. In this case A accepts if $q \in F$ is a final state; otherwise, it does not accept. A word $w \in \Sigma^*$ is *accepted by* A if there exists an initial state $q_0 \in I$ and an accepting computation from $q_0w \cdot \$$. Further, the empty word λ is accepted by A if there exists an initial state $q_0 \in Q$ such that q_0 is also a final state. Now $L(A) = \{ w \in \Sigma^* \mid w \text{ is accepted by } A \}$ is the *language accepted by* A. Notice that the endmarker is, in fact, needless; we kept it only for traditional reason.

The classical *nondeterministic finite automata* (NFA) is obtained from the NFAwtl by removing the endmarker \$ and by ignoring the translucency relation τ, and the *deterministic finite-state acceptor* (DFA) is obtained from the DFAwtl in the same way. Thus, the NFA (DFA) can be interpreted as a special type of NFAwtl (DFAwtl). Accordingly, all regular languages are accepted by DFAwtl. Moreover, DFAwtls are much more expressive than standard DFAs as shown by the following example.

Example 1 *Consider the DFAwtl* $A = (Q,\Sigma,\$,\tau,I,F,\delta)$, *where* $Q = \{q_0,q,q_a,q_b,q_c,q_d\}$, $I = \{q_0\}$, $F =$

$\{q\}$, $\Sigma = \{a,b,c,d\}$, and the functions τ and δ are defined as follows:

$$\begin{array}{rclrcl}
\tau(q_0) & = & \{b,c,d\}, & \delta(q_0,a) & = & \{q_a\}, \\
\tau(q) & = & \emptyset, & \delta(q,a) & = & \{q_a\},\ \delta(q,b) = \{q_b\},\ \delta(q,c) = \{q_c\},\ \delta(q,d) = \{q_d\}, \\
\tau(q_a) & = & \{a,c,d\}, & \delta(q_a,b) & = & \{q\}, \\
\tau(q_b) & = & \{b,c,d\}, & \delta(q_b,a) & = & \{q\}, \\
\tau(q_c) & = & \{a,b,c\}, & \delta(q_c,d) & = & \{q\}, \\
\tau(q_d) & = & \{a,b,d\}, & \delta(q_d,c) & = & \{q\}.
\end{array}$$

Further, $\delta(p,x) = \emptyset$ for all other pairs $(p,x) \in Q \times \Sigma$. Firstly, the input must have an a, which is consumed in the first step of the computation, then a b is consumed. One may see that after that the automaton reads the first letter of the remaining input and depending on what it was, in the next step consumes the first occurrence of a letter that is a pair of the previously erased one, where pairs are a-s with b-s and c-s with d-s. Consequently A accepts the language $L^{ab} = \{w \in \{a,b,c,d\}^* \mid |w|_a = |w|_b > 0$ and $|w|_c = |w|_d\}$. Similarly, by permuting the roles of the letters, e.g., the language $L^{ac} = \{w \in \{a,b,c,d\}^* \mid |w|_a = |w|_c > 0$ and $|w|_b = |w|_d\}$ is also accepted by a DFAwtl. However, the union of these two languages can be accepted by an NFAwtl, but cannot with any DFAwtl. This latter fact can be shown somewhat analogously to the fact that the context-free language $\{a^n b^n c^m d^m\} \cup \{a^n b^m c^m d^n\}$ is not deterministic context-free. We skip the formal proof because the lack of space.

As we have already described NFAwtl and DFAwtl are reformulations of cooperative distributed systems of stateless deterministic restarting R automata with window size one. The DFAwtl, in fact, are reinterpretations of stateless globally deterministic CD-R(1)-systems [29].

Recently various concepts about deterministic computations have been emerged, therefore, we recall the concept of state-determinism from [22].

An automaton is *state-deterministic* if for each of its state $q \in Q$, if there is a transition from q and it goes to state p (i.e., $p \in \delta(q,a)$), then every transition from q goes to p, that is, if an automaton has state q in its actual configuration, then, if the computation continues, the state of the next configuration is uniquely determined and it is p.

We are continuing the paper in this line.

3 On state-deterministic finite automata with translucent letters

As our first result, we investigate the state-deterministic FAwtl (SFAwtl for short) by applying this type of concept of determinism to NFAwtl.

First, we recall the concept of *stateless strictly deterministic CD-R(1)-systems* [29]. In these systems there is only one initial state, and there is exactly one successor component for each component. One may think, that in the terminology of finite automata with translucent letters we can interpret it with the conditions $|I| = 1$ and for each $q \in Q$, $|\bigcup_{a \in \Sigma} \delta(q,a)| = 1$ which may lead to a very similar concept as state-determinism. However, this is not exactly the case, stateless strictly deterministic CD-R(1)-systems and the state-deterministic FAwtl are in close relation, but in a CD-R(1)-system one may use the computation step "Accept" at any component on a given non translucent letter, while in NFAwtl the acceptance condition is defined in a different way. We are showing some explicit difference of these models in this section.

We present an example to show that these restricted automata are still able to accept non trivial languages.

Example 2 *Let $A = (Q, \Sigma, \$, \tau, I, F, \delta)$, where $Q = \{q_0, q_1\}$, $I = \{q_0\} = F$, $\Sigma = \{0,1\}$, and the functions τ and δ are defined as follows:*

$$\begin{array}{lcllcl} \tau(q_0) & = & \emptyset, & \delta(q_0,0) & = & \{q_1\}, \\ \tau(q_1) & = & \{0\}, & \delta(q_1,1) & = & \{q_0\}. \end{array}$$

By observing the structure of this automaton, it is clearly state-deterministic. Considering the accepted language, it is the Dyck language, where 0 refers to opening and 1 to closing brackets.

Example 3 *Let $A = (Q, \Sigma, \$, \tau, I, F, \delta)$, where $Q = \{q_0, q_1, q_2, q_3\}$, $I = \{q_0\} = F$, $\Sigma = \{a,b,c,d\}$, and the functions τ and δ are defined as follows:*

$$\begin{array}{lcllcl} \tau(q_0) & = & \emptyset, & \delta(q_0,a) & = & \{q_1\}, \\ \tau(q_1) & = & \{a,c,d\}, & \delta(q_1,b) & = & \{q_2\}, \\ \tau(q_2) & = & \{a,b,d\}, & \delta(q_2,c) & = & \{q_3\}, \\ \tau(q_3) & = & \{a,b,c\}, & \delta(q_3,d) & = & \{q_0\}. \end{array}$$

Further, $\delta(q,x) = \emptyset$ for all other pairs $(q,x) \in Q \times \Sigma$. On the one hand, it is easy to check that A is a DFAwtl which is, in fact, also state-deterministic. On the other hand, the language accepted by A intersected by the regular language $a^*d^*c^*b^*$ is the non context-free language $\{a^n d^n c^n b^n \mid n \geq 0\}$, and thus A accepts a language that is not context-free.

From this example, using the fact that any language accepted by NFAwtl has a letter-equivalent sublanguage that is regular [27] (but the language $\{a^n d^n c^n b^n \mid n \geq 0\}$ does not), we can deduce that:

Proposition 1 *The language class accepted by state-deterministic FAwtl is not closed under intersection with regular languages.*

Based on [22], we know that state-deterministic FAwtl have the graph structure with no branching, that is, either a line graph (starting from the sole initial state) or a line graph with an additional edge from the last state to a state.

Clearly languages like a^*, b^*, $a + aaa$, $ab + ba$ are accepted by state-deterministic FAwtl with 1, 1, 4 and 3 states, respectively. For $ab + ba$ translucency can be used in the initial state, e.g., a is translucent and transition with b leads to the next state, from which the computation may continue by reading an a to reach the final state.

Now, we present a relatively simple example language that is not accepted by any SFAwtl.

Example 4 *The regular language described by $a^* + b^*$ is not accepted by any state-deterministic FAwtl. It is easy to see that, by assuming that an SFAwtl A accepts the given language, after reading an a or a b, A must be in the same state, however, the possible computations after erasing an a or erasing a b must not be the same, since this would lead to accept words containing both a and b, contradicting to our assumption on the accepted language.*

Based on the above example, we may deduce the following closure property:

Proposition 2 *The language class accepted by SFAwtl is not closed under union.*

We may also summarize some hierarchy type results based on the previously shown examples.

Proposition 3 *State-deterministic FAwtl are deterministic, i.e., they are also DFAwtl.*
Further, the language class accepted by SFAwtl includes some non context-free languages, but on the other hand, does not include all regular languages.

We recall from [31] that the language class accepted by stateless strictly deterministic CD-R(1)-systems is closed under complement. We show that this is not the case with the state-deterministic FAwtl, thus, in this way, we also show that the new concept is not a reinterpretation of these CD systems.

Proposition 4 *The language class accepted by SFAwtl is not closed under complement.*

Proof On the one hand, as we have described, the Dyck language over $\{0,1\}$ is accepted by state-deterministic FAwtl. Now, on the other hand, we show that its complement L^c is not. Let as assume towards a contradiction that there is an SFAwtl A that accepts L^c. Since λ is not in L^c, the initial state q_0 is not a final state of A. Let us consider the cases by seeing which of the letters could be translucent at q_0.

- Clearly, it cannot happen that both 0 and 1 are translucent, since then, no words would be accepted.
- In case either 0 or 1 is translucent, there must be a transition with the other letter from q_0, to another state, say state q_1. Now, on the one hand, the input 01 should not be accepted, but the input 10 should be. However, in this case, both of these inputs lead to the same configuration after the first step of the computation. This leads to a contradiction, since from here either both of them are accepted by A, or none of them.
- The last possibility is when there are no translucent letters at q_0. Since A must accept words starting with 0, e.g., $0,00,000,001,010$ and also words starting with 1, e.g., $1,10,11,100$, the transition with both 0 and 1 must go an accepting state q_1, i.e., $\delta(q_0,a) = \delta(q_0,b) = q_1$. Considering the possible input words 01 and 11, thus we reach the same configuration $q_1$1\$, however, the former word should not be accepted, while the latter one is in L^c. By this contradiction, the proof has been finished.

•

Proposition 5 *The language class accepted by SFAwtl is not closed under concatenation.*

Proof Let us consider the languages $a+aaa$ and b which both are accepted by state-deterministic FAwtl. Let us consider now their concatenation $L_c = \{ab, aaab\}$. Let us assume that there is an SFAwtl A that accepts L_c. In its initial state q_0, there are the following options:

- There is no translucent letters for q_0, then there must be a transition with a to a state $q_1 (\neq q_0)$ and no other transition from q_0. Neither q_0, nor q_1 is a final state. Now, at q_1 A should be able to read a b and it must reach an accepting state q_2. Now there are two subcases:
  - If there is no translucency used at q_1, it must also have a transition with a (to q_2) allowing to process the word *aaab*, however, in this case there would be a false acceptance of *aa* by A arriving to a contradiction.
  - In the second subcase, a must be translucent in q_1, and thus from the original input *aaab*, the remaining was *aab* and in this way the last letter b could be read, and then the remaining *aa* should be accepted. However, in this case A would also accept the word *abaa* with a similar computation as *aaab* contradicting to the fact that it accepts L_c.
- If a is translucent at q_0, then we must have a transition with b, then from q_1 A should able to accept the remaining word a. However, in this case a similar computation would accept also *ba* as the computation for *ab*. Contradiction.
- If b is translucent at q_0, then we must have a transition with a, leading to the state q_1, similarly as in the first case. For q_1 the proof works in exactly in the same way as in that case.
- If both a and b are translucent in q_0, no transition can be defined, consequently, the nonempty language L_c cannot be accepted in this way.

•

4 The new models with nondeterministic translucency

In this section, first we provide the formal definition of the new automata models and their work.

A *(deterministic) finite state automaton with nondeterministically translucent letters*, abbreviated as (DFAwntl) NFAwntl, is defined as a septuple $A = (Q, \Sigma, \$, \tau, I, F, \delta)$, similarly to NFAwtl (DFAwtl), respectively, but without the condition that $\forall q \in Q\ \forall a \in \tau(q) : \delta(q,a) = \emptyset$. That is, for an NFAwntl (DFAwntl) it is allowed that for a state $q \in Q$ and for a letter $a \in \Sigma$ both $a \in \tau(q)$ and $\delta(q,a) \neq \emptyset$ hold. Notice that at a DFAwntl, there is only one initial state, and there is at most one transition defined for any input letter, as at DFAwtl.

A computation step of A is defined as follows. Assume that $w = a_1 a_2 \cdots a_n$ for some $n \geq 1$ and $a_1, \ldots, a_n \in \Sigma$ and A is in state q. Then A looks for an occurrence of a letter (say $a_i = b$) for which a transition is defined, i.e., $\delta(q,b) \neq \emptyset$ such that each letter $a_j \in \tau(q)$ with $j < i$. In this way, the actual configuration can be written as $q \cdot ubv\$$ with letter b in the position of a_i, $u \in \tau(q)^*$, $v \in \Sigma^*$, and the next configuration could be $p \cdot uv\$$ for a state $p \in \delta(q,b)$.
On the other hand, it may happen that such letter a_i does not exist, i.e., there is a letter $a_i = c \notin \tau(q)$ such that for each $j < i$ $a_j \in \tau(q)$ and $\delta(q,a_k) = \emptyset$ (for all $k \leq j$) including $\delta(q,c) = \emptyset$. In this case A halts without accepting; this computation fails.
Further, if $w \in (\tau(q))^*$ for a configuration $qw\$$ with $q \in F$, then A reaches the \$-symbol and the computation halts by accepting.
Finally, $w \in (\tau(q))^*$, $q \notin F$ and there is no letter in w for which a transition has been defined ($\delta(q,a_i) = \emptyset$ for each i, or $w = \lambda$), then the computation fails: A does not accept.

A word $w \in \Sigma^*$ is *accepted by A* if there exists an initial state $q_0 \in I$ and an accepting computation from $q_0 w \cdot \$$. Now $L(A) = \{ w \in \Sigma^* \mid w \text{ is accepted by } A \}$ is the *language accepted by A*.

Based on these definitions, we can define four categories of NFAwtl:

translucency \ transition mapping	deterministic	nondeterministic
disjoint	DFAwtl	NFAwtl
nondeterministic	DFAwntl	NFAwntl

As we will show although the model DFAwntl seems deterministic by its transition function, we may easily cheat by the nondeterminism allowed by translucency.

Example 5 *Let $A = (Q, \Sigma, \$, \tau, I, F, \delta)$, where $Q = \{q_0, q_a, q_b, q_c\}$, $I = \{q_0\}$, $F = \{q_c\}$, $\Sigma = \{a,b,c\}$, and the functions τ and δ are defined as follows:*

$$\begin{array}{lcllcl} \tau(q_0) & = & \emptyset, & \delta(q_0,a) & = & \{q_a\}, \delta(q_0,b) = \{q_b\}, \delta(q_0,c) = \{q_c\}, \\ \tau(q_a) & = & \{a,b,c\}, & \delta(q_a,b) & = & \{q_0\}, \\ \tau(q_b) & = & \{a,b,c\}, & \delta(q_b,a) & = & \{q_0\}, \\ \tau(q_c) & = & \emptyset. & & & \end{array}$$

Further, $\delta(q,x) = \emptyset$ for all other pairs $(q,x) \in Q \times \Sigma$. It is easy to check that A is a DFAwntl.
Let us see how A works. Since there are no translucent letters in q_0, A consumes the first letter of the remaining input always in this state. If it was an a, then it erases a b from anywhere in the tape; if A consumes a b at state q_0, then in the next computation step A erases an a from anywhere in the tape. Finally, the input is accepted if only a c remains on the tape and A is in state q_0, then it reaches the accepting state q_c.

Thus, for every accepted word the number of its a-s and b-s are the same and it contains a c. Let us write such a word in the form vcu with $v,u \in \{a,b\}^*$. It is also easy to see that A may accept various

words where $|v| \geq |u|$, but no words with $|v| < |u|$. On the other hand, let us see which words are accepted with the property $|v| = |u|$. By the work of A, the conditions $|v|_a = |u|_b$ and $|v|_b = |u|_a$ must hold, i.e., in v the number of a-s is the same as the number of b-s in u and vice versa.

Now we are arguing that the same language cannot be accepted by any NFAwtl without using nondeterministic translucency (due to lack of space we skip some parts of the formal proof). Let us assume that there is an NFAwtl B that accepts the same language as A. Let the number of states of B is n. Let us consider a word $w = a^k b^\ell c a^\ell b^k \in L(A)$ with $k, \ell > 2n$. By our assumption B accepts w, thus consider an accepting computation on w by B. Clearly, there are two cases based on the first $n+1$ steps of the computation.

- *If letter c is erased during these computation step, then it can be shown that some words vcu with $|v| < |u|$ would also be accepted by B having all the letters processed after the step in which c is read after the c in the original input word.*
- *If c is not read during the first $n+1$ steps, only a-s and b-s before the c (in part v) are accessed and processed in the first $n+1$ steps. However, by the pigeon-hole principle, a state is repeated during these steps, meaning that there are also values i and j ($0 \leq i, j \leq n+1$, $i+j > 0$) such that from input $w' = a^{k-i} b^{\ell-j} c a^\ell b^k$ in $n+1-i-j$ steps exactly the same configuration is reached as from w in $n+1$ steps. In this case, by continuing the computation on w' in the same way as the accepting computation on w, the word w' will also be accepted.*

Now, in both cases, we have reached contradiction by accepting words of the form vcu for which $|v| < |u|$.

4.1 Hierarchy of the accepted languages

In this subsection our aim is to give some hierarchy like results by establishing where the new families of languages are comparing them to various other classes.

First, we note that, in fact, the following inclusions hold by definition.

Proposition 6 *Every NFAwtl is an NFAwntl and every DFAwtl is a DFAwntl.*
Moreover, every DFAwntl is an NFAwntl.

Based on Example 5, we can also state some hierarchy results on the accepted language classes.

Proposition 7 *The language class accepted by NFAwtl is a proper subclass of the language class accepted by NFAwntl.*
The language class accepted by DFAwtl is a proper subclass of the language class accepted by DFAwntl.

Here, we leave open the question if NFAwntl is more efficient and expressive than DFAwntl.

On the one hand, we have seen that we can construct DFAwntl that accept some non context-free languages. Now, on the other hand, let us show some of their limitations.

To compare the new language classes with some classical classes of formal languages we establish the following result.

Lemma 1 *For every language accepted by an NFAwntl (DFAwntl), there is a letter equivalent sublanguage that is accepted by an NFAwtl (DFAwtl, resp.).*

Proof In case there is no such letter in any state which is both in the set of translucent letters and there is also a transition on it, the automaton is in fact, an NFAwtl (also a DFAwtl in deterministic case) and its language, as its own sublanguage, fulfils the statement of the lemma.

Now, let us assume that automaton A is an NFAwntl, but it is not an NFAwtl. On the one hand, since there is no "forced" way not to see a letter for which a transition is defined, A may always consumes the first occurrence of such a letter and in fact accepts words of a language that is also accepted by an NFAwtl. More precisely, by removing those letters from the translucency set of a state for which transitions are defined, an NFAwtl (DFAwtl) A' can be obtained. Clearly all words that A' may accept are also accepted by A with a similar computation.

To see that this language is letter equivalent to the originally accepted language, consider a computation on any of the accepted word by A. Since the automaton never knows if the consumed letter in a computation step is the first or only reached through some translucent letters, we may reorder the input according to an accepting computation, and in this way for each accepted word there will be a letter equivalent word that has also been accepted by A'. •

Moreover, if, by chance, the letters of the input are ordered in exactly the way as they are consumed during an accepting computation, then, in fact, an NFAwntl is working in the same way as an NFA, thus we may also establish the following fact:

Lemma 2 *For every language accepted by an NFAwntl, there is a letter equivalent regular sublanguage.*

From the previous lemma we can conclude:

Corollary 1 *All languages accepted by NFAwntl are semi-linear.*

This could be interesting in the mirror of the fact, that by changing the window size of an R automata from 1 to 2 (like allowing to have the translucency and transitions not letter by letter, but somehow by pairs of consecutive letters), the corresponding model, the CD system of stateless deterministic R(2) automata is able to accept some non semi-linear languages [19].

On the other hand, as the linear context-free language $\{a^n b^n\}$ does not have any letter equivalent regular sublanguage, there is no NFAwntl that could accept it. Since $\{a^n b^n\}$ is deterministic linear and also in 2detLIN (accepted by deterministic 2-head finite automata consuming the input letters from the two extremes until they are meeting [33]), we have the following incomparability results.

Theorem 1 *The language classes accepted by NFAwntl and DFAwntl properly include the class of regular languages. Further, the language classes accepted by NFAwntl and DFAwntl are incomparable to each of the following classes of languages: deterministic linear, 2detLIN, linear context-free, deterministic context-free, context-free.*

Finally, we analyse the computations of DFAwntl.

4.2 Simulating nondeterministic computations with DFAwntl

In this section our aim is to show that although seemingly by the transition function, DFAwntl seem to be deterministic automata, they have some real nondeterministic features.

Example 6 *As we mentioned in Example 1, there are DFAwtl that accept the languages*

$$L^{ab} = \{w \in \{a,b,c,d\}^* \mid |w|_a = |w|_b > 0 \text{ and } |w|_c = |w|_d\},$$

$$L^{ac} = \{w \in \{a,b,c,d\}^* \mid |w|_a = |w|_c > 0 \text{ and } |w|_b = |w|_d\}$$

and

$$L^{ad} = \{w \in \{a,b,c,d\}^* \mid |w|_a = |w|_d > 0 \text{ and } |w|_b = |w|_c\}.$$

Let these automata be having the set of states $Q^{ab} = \{q_0^{ab}, q_a^{ab}, q_b^{ab}, q_c^{ab}, q_d^{ab}, q^{ab}\}$; $Q^{ac} = \{q_0^{ac}, q_a^{ac}, q_b^{ac}, q_c^{ac}, q_d^{ac}, q^{ac}\}$ *and* $Q^{ad} = \{q_0^{ad}, q_a^{ad}, q_b^{ad}, q_c^{ad}, q_d^{ad}, q^{ad}\}$, *respectively.*

The union of these languages cannot be accepted by any DFAwtl. Let us define now a DFAwntl as follows.

$A = (Q, \Sigma, \$, \tau, I, F, \delta)$, *where* $Q = Q^{ab} \cup Q^{ac} \cup Q^{ad} \cup \{q_0\}$, $I = \{q_0\}$, $F = \{q^{ab}, q^{ac}, q^{ad}\}$, $\Sigma = \{a, b, c, d\}$, *and the functions* τ *and* δ *are defined as follows:*

$$\tau(q_0) = \{a,b,c,d\}, \qquad \delta(q_0,a) = \emptyset, \delta(q_0,b) = \{q_b^{ab}\}, \delta(q_0,c) = \{q_c^{ac}\}, \delta(q_0,d) = \{q_d^{ad}\}$$

for the newly added initial state and they are inherited from the respective automata for each other state.

Now, for any input, A nondeterministically guesses in which of the three languages the input is: as $\tau(q_0) = \Sigma$ *it has access to any occurrence of a b, a c or a d as transitions are defined for these letters in* q_0. *By guessing the input belonging to* L^{ab}, *it should have at least one b, thus by reading a b in the first step, the subautomaton accepting* L^{ab} *is chosen such that a b has already been processed. Now, it is easy to see that after this nondeterministic choice in the first step, the computation continues in a deterministic manner. The other nondeterministic choices in the first step of the computation are: by consuming a c anywhere from the input A chooses to check whether the input belongs to* L^{ac}, *and by consuming a d anywhere from the input in the first step of the computation, A chooses to check whether the input belongs to* L^{ad}. *If the guess was correct, the input will be accepted. Otherwise, A must use another computation to accept the given input, if any. Consequently the DFAwntl A accepts* $L^{ab} \cup L^{ac} \cup L^{ad}$.

We left open if NFAwntl can accept more languages than DFAwntl (the properness of the inclusion relation of these two language classes is open). It is known (see [31], for the proof) that DFAwtl cannot accept all rational trace languages, and this fact was used to prove the properness of the hierarchy between NFAwtl and DFAwtl. Actually, it was shown that the language $\{w \in \{a,b\}^* \mid |w|_a = |w|_b \text{ or } 2|w|_a = |w|_b\}$ cannot be accepted by any DFAwtl. Here we show that with a similar method as in Example 6, DFAwntl is able to accept this language. The automaton is shown in Figure 1. In each state the indicated set, if any, shows the translucent letters of the given state. The other notation is standard, e.g., double circles for final states, etc.

Last, but not least, we present some closure properties.

Proposition 8 *The class of languages accepted by NFAwntl is closed under union.*

Proof Wlog. we may assume that the two languages are over the same alphabet Σ. Now, having two NFAwntl, say $A_1 = (Q_1, \Sigma, \$, \tau_1, I_1, F_1, \delta_1)$ and $A_2 = (Q_2, \Sigma, \$, \tau_2, I_2, F_2, \delta_2)$ with $Q_1 \cap Q_2 = \emptyset$, we construct $A = (Q_1 \cup Q_2, \Sigma, \$, \tau, I_1 \cup I_2, F_1 \cup F_2, \delta)$, where $\tau(q) = \begin{cases} \tau_1(q), & \text{if } q \in Q_1; \\ \tau_2(q), & \text{if } q \in Q_2. \end{cases}$ and for each $a \in \Sigma$, $\delta(q,a) = \begin{cases} \delta_1(q,a), & \text{if } q \in Q_1; \\ \delta_2(q,a), & \text{if } q \in Q_2. \end{cases}$. Then A may choose nondeterministically among the possible initial states, depending on if the chosen state is in I_1 or I_2, A will do a computation that is similar to a computation of A_1 or A_2, respectively. •

It is known that the language class accepted by DFAwtl is not closed under union [30]. On the other hand, we have also seen, that DFAwntl may also be able to compute the union of some languages accepted by DFAwntl, however, in general we leave open the problem if this class is closed under union.

On the other hand, the language class of NFAwtl is closed under concatenation, and the proof was based on guessing when the last occurrence of the letters are consumed to give a construction when the

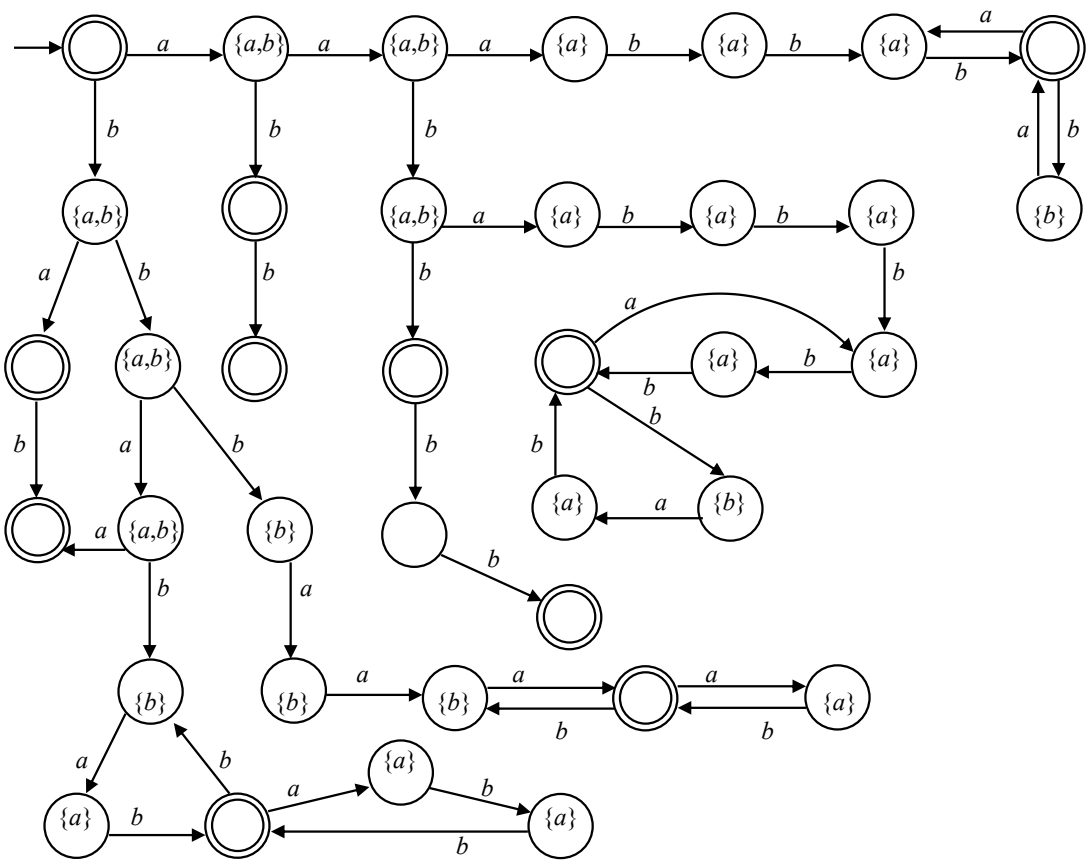


Figure 1: A DFAwntl accepting $\{w \in \{a,b\}^* \mid |w|_a = |w|_b \text{ or } 2|w|_a = |w|_b\}$.

last occurrence of any letter was consumed without using any translucency [30]. As in the new model, generally, we may not be sure when the last occurrence is consumed (maybe even in the first step), the original construction definitely does not work. Thus, the closure of the new classes under concatenation is also left as an open problem.

The fact that each accepted language must have a letter equivalent regular sublanguage and the examples shown lead also to the following non-closure property:

Proposition 9 *Language classes accepted by NFAwntl and DFAwtl are not closed under intersection with regular sets, and thus they are not closed under intersection.*

5 Conclusions

Recently, another extension of the finite automata with translucent letters was investigated in which in the computation the head is not restarting after erasing a symbol, but continues from the position where this letter has been erased (or by reaching the endmarker, it starts from the beginning again) [15]. This model is defining some new interesting classes of languages that are superclasses of the classes of languages of the original model, as the new model is able to simulate the original nondeterministic finite automata with translucent letters. Our extensions, the FAwntl, are such extensions that the original deterministic

and nondeterministic finite automata with translucent letters are special cases of our new models (we may not need to simulate them as they are included in our new classes of automata), thus the computational power of the original models has been increased by relaxing the condition of disjointness of the sets of letters for a state which is containing the translucent letters of the given state and which is containing the letters that can be read in the given state. However, we left open if nondeterministic transitions are more powerful in case we allow nondeterministic translucency (the author guesses/conjectures that the model DFAwntl is weaker than NFAwntl in terms of accepted languages). It is also left open if DFAwntl are able to accept all rational trace languages.

Although the expressive power has been increased, the new model still has various limitations, as the accepted languages must always have a letter equivalent regular sublanguage. As the class is not closed under intersection with regular languages, transduced-input variants can be investigated and studied in the future similarly to [5]. Various closure properties of the new classes of languages are left open, they are also subjects of future studies.

We believe that the combination of the new directions by continuing the computation from the position of the erased letter and by using nondeterministic translucency, can fruitfully be considered also in the future.

In the other newly investigated model we have applied the state-deterministic restriction for FAwtl showing that this model is accepting an interesting family of languages. Combining state-determinism and nondeterministic translucency and/or the non-returning feature could also be a nice topic for future research.

Acknowledgements

The comments of the anonymous reviewers are gratefully acknowledged.

References

[1] Henning Bordihn, Markus Holzer & Martin Kutrib (2005): *Revolving-Input Finite Automata*. In Clelia de Felice & Antonio Restivo, editors: *Developments in Language Theory, 9th International Conference, DLT 2005, Palermo, Italy, July 4-8, 2005, Proceedings*, *Lecture Notes in Computer Science* 3572, Springer, pp. 168–179, doi:10.1007/11505877_15.

[2] P. Cartier & D. Foata (1969): *Problèmes combinatoires de commutation et réarrangements*. Springer, doi:10.1007/BFb0079468.

[3] Hiroyuki Chigahara, Szilárd Zsolt Fazekas & Akihiro Yamamura (2016): *One-Way Jumping Finite Automata*. *Int. J. Found. Comput. Sci.* 27(3), p. 391, doi:10.1142/S0129054116400165.

[4] Volker Diekert & Gregorz (eds.) Rozenberg (1995): *The Book of Traces*. World Scientific, Singapore, doi:10.1142/2563.

[5] Madeeha Fatima & Benedek Nagy (2020): *Transduced-Input Automata with Translucent Letters*. *Comptes rendus de l'Académie bulgare des Sciences* 73(1), pp. 33–39, doi:10.7546/CRABS.2020.01.04.

[6] Rudolf Freund, Gheorghe Paun, Grzegorz Rozenberg & Arto Salomaa (1997): *Watson-Crick finite automata*. In Harvey Rubin & David Harlan Wood, editors: *DNA Based Computers, Proceedings of a DIMACS Workshop, Philadelphia, Pennsylvania, USA, June 23-25, 1997*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 48, DIMACS/AMS, pp. 297–327, doi:10.1090/dimacs/048/22.

[7] M. A. Harrison (1978): *Introduction to Formal Language Theory*. Addison-Wesley.

[8] J.E. Hopcroft & J.D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, M.A.

[9] Petr Jancar, Frantisek Mráz, Martin Plátek, Martin Procházka & Jörg Vogel (1995): *Restarting Automata, Marcus Grammars and Context-Free Languages*. In Jürgen Dassow, Grzegorz Rozenberg & Arto Salomaa, editors: *Developments in Language Theory II, At the Crossroads of Mathematics, Computer Science and Biology, Magdeburg, Germany, 17-21 July 1995*, World Scientific, Singapore, pp. 102–111.

[10] Ryszard Janicki, Jetty Kleijn, Maciej Koutny & Lukasz Mikulski (2019): *Classifying invariant structures of step traces*. *J. Comput. Syst. Sci.* 104, pp. 297–322, doi:10.1016/j.jcss.2017.05.002.

[11] Radim Kocman, Zbynek Krivka, Alexander Meduna & Benedek Nagy (2022): *A jumping $5'\rightarrow 3'$ Watson-Crick finite automata model*. *Acta Informatica* 59(5), pp. 557–584, doi:10.1007/s00236-021-00413-x.

[12] Roussanka Loukanova (2007): *Linear Context Free Languages*. In Cliff B. Jones, Zhiming Liu & Jim Woodcock, editors: *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, *Lecture Notes in Computer Science* 4711, Springer, pp. 351–365, doi:10.1007/978-3-540-75292-9_24.

[13] Alexandru Mateescu, Kai Salomaa & Sheng Yu (2000): *On Fairness of Many-Dimensional Trajectories*. *J. Autom. Lang. Comb.* 5(2), pp. 145–157, doi:10.25596/jalc-2000-145.

[14] Alexander Meduna & Petr Zemek (2012): *Jumping Finite Automata*. *Int. J. Found. Comput. Sci.* 23(7), pp. 1555–1578, doi:10.1142/S0129054112500244.

[15] Frantisek Mráz & Friedrich Otto (2022): *Non-Returning Finite Automata With Translucent Letters*. In Henning Bordihn, Géza Horváth & György Vaszil, editors: *Proceedings 12th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2022, Debrecen, Hungary, August 26-27, 2022*, *EPTCS* 367, pp. 143–159, doi:10.4204/EPTCS.367.10.

[16] B. Nagy & F. Otto (2011): *Finite-state acceptors with translucent letters*. In G. Bel-Enguix, V. Dahl & A.O. De La Puente, editors: *BILC 2011: AI Methods for Interdisciplinary Research in Language and Biology, Proc.; in ICAART 2011: 3rd International Conference on Agents and Artificial Intelligence*, SciTePress, Portugal, pp. 3–13.

[17] Benedek Nagy (2008): *On $5' \to 3'$ Sensing Watson-Crick Finite Automata*. In Max H. Garzon & Hao Yan, editors: *DNA Computing, 13th International Meeting on DNA Computing, DNA13, Memphis, TN, USA, June 4-8, 2007, Revised Selected Papers*, *Lecture Notes in Computer Science* 4848, Springer, pp. 256–262, doi:10.1007/978-3-540-77962-9_27.

[18] Benedek Nagy (2009): *Languages generated by context-free grammars extended by type $AB \to BA$ rules*. *Journal of Automata, Languages and Combinatorics* 14, pp. 175–186.

[19] Benedek Nagy (2011): *On CD-Systems of Stateless Deterministic R(2)-Automata*. *J. Autom. Lang. Comb.* 16(2-4), pp. 195–213, doi:10.25596/jalc-2011-195.

[20] Benedek Nagy (2012): *A class of 2-head finite automata for linear languages*. *Triangle* 8 (Languages. Mathematical Approaches), pp. 89–99.

[21] Benedek Nagy (2013): *On a hierarchy of $5' \to 3'$ sensing Watson-Crick finite automata languages*. *J. Log. Comput.* 23(4), pp. 855–872, doi:10.1093/logcom/exr049.

[22] Benedek Nagy (2021): *State-deterministic $5' \to 3'$ Watson-Crick automata*. *Nat. Comput.* 20(4), pp. 725–737, doi:10.1007/s11047-021-09865-z.

[23] Benedek Nagy (2009): *On a hierarchy of $5' \to 3'$ sensing WK finite automata languages*. In: *Mathematical Theory and Computational Practice, CiE, Abstract Booklet, Heidelberg, Germany*, pp. 266–275.

[24] Benedek Nagy (2022): *Quasi-deterministic $5' \to 3'$ Watson-Crick Automata*. In Henning Bordihn, Géza Horváth & György Vaszil, editors: *Proceedings 12th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2022, Debrecen, Hungary, August 26-27, 2022*, *EPTCS* 367, pp. 160–176, doi:10.4204/EPTCS.367.11.

[25] Benedek Nagy & Arif A. Akkeles (2017): *Trajectories and Traces on Non-traditional Regular Tessellations of the Plane*. In Valentin E. Brimkov & Reneta P. Barneva, editors: *Combinatorial Image Analysis - 18th International Workshop, IWCIA 2017, Plovdiv, Bulgaria, June 19-21, 2017, Proceedings*, *Lecture Notes in Computer Science* 10256, Springer, pp. 16–29, doi:10.1007/978-3-319-59108-7_2.

[26] Benedek Nagy & László Kovács (2014): *Finite Automata with Translucent Letters Applied in Natural and Formal Language Theory*. In: *Transactions on Computational Collective Intelligence XVII*, Lecture Notes in Computer Science 8790, Springer, pp. 107–127, doi:10.1007/978-3-662-44994-3_6.

[27] Benedek Nagy & Friedrich Otto (2010): *CD-Systems of Stateless Deterministic R(1)-Automata Accept All Rational Trace Languages*. In Adrian-Horia Dediu, Henning Fernau & Carlos Martín-Vide, editors: *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, *Lecture Notes in Computer Science* 6031, Springer, pp. 463–474, doi:10.1007/978-3-642-13089-2_39.

[28] Benedek Nagy & Friedrich Otto (2011): *An Automata-Theoretical Characterization of Context-Free Trace Languages*. In Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith G. Jeffery, Rastislav Královic, Marko Vukolic & Stefan Wolf, editors: *SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. Proceedings*, *Lecture Notes in Computer Science* 6543, Springer, pp. 406–417, doi:10.1007/978-3-642-18381-2_34.

[29] Benedek Nagy & Friedrich Otto (2011): *Globally Deterministic CD-Systems of Stateless R(1)-Automata*. In Adrian-Horia Dediu, Shunsuke Inenaga & Carlos Martín-Vide, editors: *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings*, *Lecture Notes in Computer Science* 6638, Springer, pp. 390–401, doi:10.1007/978-3-642-21254-3_31.

[30] Benedek Nagy & Friedrich Otto (2012): *On CD-systems of stateless deterministic R-automata with window size one*. *J. Comput. Syst. Sci.* 78(3), pp. 780–806, doi:10.1016/j.jcss.2011.12.009.

[31] Benedek Nagy & Friedrich Otto (2013): *Globally deterministic CD-systems of stateless R-automata with window size 1*. *Int. J. Comput. Math.* 90(6), pp. 1254–1277, doi:10.1080/00207160.2012.688820.

[32] Benedek Nagy & Friedrich Otto (2020): *Linear automata with translucent letters and linear context-free trace languages*. *RAIRO Theor. Informatics Appl.* 54, p. 3, doi:10.1051/ita/2020002.

[33] Benedek Nagy & Shaghayegh Parchami (2021): *On deterministic sensing $5' \to 3'$ Watson-Crick finite automata: a full hierarchy in 2detLIN*. *Acta Informatica* 58(3), pp. 153–175, doi:10.1007/s00236-019-00362-6.

[34] Benedek Nagy & Shaghayegh Parchami (2022): *$5' \to 3'$ Watson-Crick automata languages – without sensing parameter*. *Nat. Comput.* 21(4), pp. 679–691, doi:10.1007/s11047-021-09869-9.

[35] Benedek Nagy, Shaghayegh Parchami & Hamid Mir Mohammad Sadeghi (2017): *A New Sensing $5' \to 3'$ Watson-Crick Automata Concept*. In Erzsébet Csuhaj-Varjú, Pál Dömösi & György Vaszil, editors: *Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, September 4-6, 2017*, *EPTCS* 252, pp. 195–204, doi:10.4204/EPTCS.252.19.

[36] Friedrich Otto (2006): *Restarting Automata*. In Zoltán Ésik, Carlos Martín-Vide & Victor Mitrana, editors: *Recent Advances in Formal Languages and Applications*, *Studies in Computational Intelligence* 25, Springer, pp. 269–303, doi:10.1007/978-3-540-33461-3_11.

[37] Shaghayegh Parchami & Benedek Nagy (2018): *Deterministic Sensing $5' \to 3'$ Watson-Crick Automata Without Sensing Parameter*. In Susan Stepney & Sergey Verlan, editors: *Unconventional Computation and Natural Computation - 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018, Proceedings*, *Lecture Notes in Computer Science* 10867, Springer, pp. 173–187, doi:10.1007/978-3-319-92435-9_13.

[38] R. J. Parikh (1961): *Language generating devices*. *MIT Res. Lab., Quarterly Progress Report* 60, pp. 199–212.

[39] Gheorghe Paun, Grzegorz Rozenberg & Arto Salomaa (1998): *DNA Computing - New Computing Paradigms*. Texts in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/978-3-662-03563-4.

Words-to-Letters Valuations for Language Kleene Algebras with Variable Complements

Yoshiki Nakamura
Tokyo Institute of Technology, Japan
nakamura.yoshiki.ny@gmail.com

Ryoma Sin'ya
Akita University, Japan
ryoma@math.akita-u.ac.jp

We investigate the equational theory of Kleene algebra terms with *variable complements*—(language) complement where it applies only to variables—w.r.t. languages. While the equational theory w.r.t. languages coincides with the language equivalence (under the standard language valuation) for Kleene algebra terms, this coincidence is broken if we extend the terms with complements. In this paper, we prove the decidability of some fragments of the equational theory: the universality problem is coNP-complete, and the inequational theory $t \leq s$ is coNP-complete when t does not contain Kleene-star. To this end, we introduce *words-to-letters valuations*; they are sufficient valuations for the equational theory and ease us in investigating the equational theory w.r.t. languages. Additionally, we prove that for words with variable complements, the equational theory coincides with the word equivalence.

1 Introduction

Kleene algebra (KA) [3, 6] is an algebraic system for regular expressions consisting of union ($\cup$), composition ($\cdot$), Kleene-star ($\_^*$), emptiness ($\bot$), and identity (I). In this paper, we consider KAs *w.r.t. languages* (a.k.a., language models of KAs, language KAs). Interestingly, the equational theory of KAs w.r.t. languages coincides with the language equivalence under the standard language valuation (see also, e.g., [1, 11]): for all KA terms (i.e., regular expressions) t, s, we have

$$\mathsf{LANG} \models t = s \quad \Longleftrightarrow \quad [t] = [s]. \tag{$\dagger$}$$

Here, we write $\mathsf{LANG} \models t = s$ if the equation $t = s$ holds for all language models (i.e., each variable x maps to not only the singleton language $\{x\}$ but also any languages); we write $[u]$ for the language of a regular expression u (i.e., each variable x maps to the singleton language $\{x\}$). Since the valuation $[\_]$ is an instance of valuations in LANG, the direction $\Longrightarrow$ is trivial (this direction always holds even if we extend KA terms with some extra operators). The direction $\Longleftarrow$ is a consequence of the completeness of KAs (see Appendix A for an alternative proof not relying on the completeness of KAs). However, the direction $\Longleftarrow$ fails in general when we extend KA terms with extra operators. Namely, the equational theory w.r.t. languages does not coincide with the language equivalence in general (see below for complements). The equational theory of KAs with some operators w.r.t. languages was studied, e.g., with reverse [2], with tests [7] (where languages are of guarded strings, not words), with intersection [1], and with universality ($\top$) [11]. Nevertheless, to the best of authors' knowledge, *variable complements* (and even complements) w.r.t. languages has not yet been investigated, while those w.r.t. binary relations were studied, e.g., in [10] (for complements; cf. Tarski's calculus of relations [13]) and [9] (for variable complements).

In this paper, we investigate the equational theory of KA terms with *variable complements* (x^-)—(language) complement, where it applies only to variables (we use x to denote variables)—w.r.t. lan-

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 185–199, doi:10.4204/EPTCS.386.15

guages. For KA terms with variable complements, (†) fails. The following is an example:

$$\mathsf{LANG} \not\models x^- = x^- \cdot x^- \qquad\qquad [x^-] = [x^- \cdot x^-].$$

(For $\mathsf{LANG} \not\models$, consider a valuation such that x^- maps to the language $\{a\}$.) As the example above (see also Remark 1, for more examples) shows, the equational theory of KAs with variable complements w.r.t. languages significantly differs from the language equivalence under the standard language valuation. While the language equivalence problem of KA terms with variable complements is decidable in PSPACE by a standard automata construction [14] (and hence, PSPACE-complete [5, 8, 12]), it remains whether the equational theory w.r.t. languages is decidable.

We prove the decidability and complexity of some fragments of the equational theory of KA terms with variable complements w.r.t. languages: the universality problem is coNP-complete (Cor. 18), and the inequational theory $t \leq s$ is coNP-complete when t does not contain Kleene-star (Cor. 29). To this end, we introduce *words-to-letters valuations*. Words-to-letters valuations are sufficient for the equational theory of KA terms with variable complements w.r.t. languages (words-to-letters-valuation property; Cor. 30): for all terms t, s, if there is some language valuation such that it refutes $t = s$, there is some words-to-letters valuation such that it refutes $t = s$. This property eases us in investigating the equational theory w.r.t. languages.

Additionally, by using words-to-letters valuations, we prove a completeness theorem for words with variable complements: the equational theory coincides with the word equivalence (Thm. 32). A limitation of words-to-letters valuations is that the number of letters is not bounded; so, they cannot apply to the equational theory over LANG_n (language models over sets of cardinality at most a natural number n). Nevertheless, by giving another valuation, we can show the coincidence for one-variable words (Thm. 36). We leave open for the many-variable words.

Outline

In Sect. 2, we briefly give basic definitions, including the syntax and semantics of KA terms with variable complements. Additionally, we give languages for KA terms with variable complements (Sect. 2.3). In Sects. 3–5, we consider fragments of the equational theory of KA terms with variable complements w.r.t. languages, step-by-step. In Sect. 3, we consider the identity inclusion problem ($\mathsf{LANG} \models \mathsf{I} \leq t$?). This problem is relatively easy but contains the coNP-hardness result (Cor. 6). In Sect. 4, we consider the variable inclusion problem ($\mathsf{LANG} \models x \leq t$?) and the universality problem ($\mathsf{LANG} \models \top \leq t$?). In Sect. 5, we consider the word inclusion problem ($\mathsf{LANG} \models w \leq t$?). This section proceeds in the same way as Sect. 4, thanks to words-to-letters valuations (Def. 21). Consequently, the inequational theory $t \leq s$ is coNP-complete when t does not contain Kleene-star (Cor. 29). Additionally, we prove the words-to-letters valuation property (Cor. 30) for the equational theory of (full) KA terms with variable complements w.r.t. languages. In Sect. 6, we consider the equational theory of words with variable complements and show a completeness theorem (Thm. 32). Sect. 7 concludes this paper.

2 Preliminaries

We write $\mathbb{N}$ for the set of non-negative integers. For a set X, we write $\#(X)$ for the cardinality of X and $\wp(X)$ for the power set of X.

For a set X (of letters), we write X^* for the set of words over X: finite sequences of elements of X. We write I for the empty word. We write wv for the concatenation of words w and v. A language over

X is a subset of X^*. We use w, v to denote words and use L, K to denote languages, respectively. For languages $L, K \subseteq X^*$, the composition $L \cdot K$ and the Kleene star L^* is defined by:

$$\begin{array}{rcl} L \cdot K & \triangleq & \{wv \mid w \in L \wedge w \in K\}; \\ L^* & \triangleq & \{w_0 \dots w_{n-1} \mid \exists n \in \mathbb{N}, \forall i < n,\ w_i \in L\}. \end{array}$$

2.1 Syntax: KA terms with variable complements

Let $\mathbf{V}$ be a set of variables. The set of Kleene algebra (KA) terms with variable complements (x^-) is defined by the following grammar:

$$\mathbf{T} \ni t, s, u \quad ::= \quad x \mid \mathsf{I} \mid \bot \mid t \cdot s \mid t \cup s \mid t^* \mid x^- \qquad (x \in \mathbf{V})$$

We use parentheses in ambiguous situations. We often abbreviate $t \cdot s$ to ts. We write $\top$ for the term $x \cup x^-$, where x is any variable.

An *equation* $t = s$ is a pair of terms. An *inequation* $t \leq s$ is an abbreviation of the equation $t \cup s = s$.

2.2 Semantics: language models

Consider the signature $S \triangleq \{\mathsf{I}_{(0)}, \bot_{(0)}, \cdot_{(2)}, \cup_{(2)}, \_^*{}_{(1)}, \_^-{}_{(1)}\}$. An *$S$-algebra* A is a tuple $\langle |A|, \{f^A\}_{f_{(k)} \in S} \rangle$, where $|A|$ is a non-empty set and $f^A \colon |A|^k \to |A|$ is a k-ary map for each $f_{(k)} \in S$. A *valuation* $\mathfrak{v}$ on an S-algebra A is a map $\mathfrak{v} \colon \mathbf{V} \to |A|$. For a valuation $\mathfrak{v}$, we write $\hat{\mathfrak{v}} \colon \mathbf{T} \to |A|$ for the unique homomorphism extending $\mathfrak{v}$.

The *language model* A over a set X is an S-algebra such that $|A| = \wp(X^*)$ and for all $L, K \subseteq X^*$,

$$\mathsf{I}^A = \{\mathsf{I}\} \qquad L \cdot^A K = L \cdot K \qquad L^{*^A} = L^*$$
$$\bot^A = \emptyset \qquad L \cup^A K = L \cup K \qquad L^{-^A} = X^* \setminus L.$$

We write LANG for the class of all language models. A *language valuation* over a set X is a valuation on some language model over X. For an equation $t = s$, we let

$$\mathsf{LANG} \models t = s \quad \stackrel{\triangle}{\Longleftrightarrow} \quad \hat{\mathfrak{v}}(t) = \hat{\mathfrak{v}}(s) \text{ holds for all language valuations } \mathfrak{v}.$$

The *equational theory w.r.t. languages* is the set of all equations $t = s$ such that $\mathsf{LANG} \models t = s$.

Additionally, the *language* $[t] \subseteq \mathbf{V}^*$ of a term t is defined by:

$$[x] \triangleq \{x\} \qquad [\mathsf{I}] \triangleq \{\mathsf{I}\} \qquad [t \cdot s] \triangleq [t] \cdot [s] \qquad [t^*] \triangleq [t]^*$$
$$[\bot] \triangleq \emptyset \qquad [t \cup s] \triangleq [t] \cup [s] \qquad [t^-] \triangleq \mathbf{V}^* \setminus [t].$$

By definition, we have $[t] = \hat{\mathfrak{v}}(t)$ if $\mathfrak{v}$ is the valuation on the language model over the set $\mathbf{V}$ defined by $\mathfrak{v}(x) = \{x\}$ for all $x \in \mathbf{V}$. Hence, for all t, s, we have

$$\mathsf{LANG} \models t = s \quad \Longrightarrow \quad [t] = [s]. \qquad (\ddagger)$$

Remark 1. The converse direction fails[1]; for example, when $x \neq y$,

$$\mathsf{LANG} \not\models y \leq x^- \qquad [y] \subseteq [x^-].$$

[1]The failure can be also shown by that the universality $\top$ can be expressed by $x \cup x^-$; see also [11, Remark. 3.6].

Here $t \leq s$ denotes the equation $t \cup s = s$ (so, indeed, an equation). For $\mathsf{LANG} \not\models y \leq x^-$, consider a language valuation $\mathfrak{v}$ such that $a \in \mathfrak{v}(x)$ and $a \in \mathfrak{v}(y)$; then we have $a \in \hat{\mathfrak{v}}(y) \setminus \hat{\mathfrak{v}}(x^-)$. $[y] \subseteq [x^-]$ is shown by $[y] = \{y\} \subseteq \mathbf{V}^* \setminus \{x\} = [x^-]$. More generally, for any word w over $\mathbf{V}$ such that $w \neq x$,

$$\mathsf{LANG} \not\models w \leq x^- \qquad\qquad [w] \subseteq [x^-].$$

Moreover, for example, there are the following examples (for $\mathsf{LANG} \not\models$, consider a valuation such that both x and y map to the language $X^* \setminus \{a\}$, where X is a set and $a \in X$):

$$\begin{array}{ll} \mathsf{LANG} \not\models x^- = x^- \cdot x^- & [x^-] = [x^- \cdot x^-] \\ \mathsf{LANG} \not\models \top = x^- \cdot y^- & [\top] = [x^- \cdot y^-] \\ \mathsf{LANG} \not\models \top = x^- \cup y^- & [\top] = [x^- \cup y^-]. \end{array}$$

As the examples above show, for KA terms with variable complements, the equational theory w.r.t. languages ($\mathsf{LANG} \models t = s$?) significantly differs from the language equivalence problem ($[t] = [s]$?).

In the sequel, we focus on the equational theory w.r.t. languages and investigate its fragments. We prepare a useful tool (Lem. 2), which enables us to decompose terms into languages of words.

2.3 Languages for KA terms with variable complements

Let $\mathbf{V}' = \{x, x^- \mid x \in \mathbf{V}\}$. For a term t, we write $[t]_{\mathbf{V}'}$ for the language of t where t is viewed as the regular expression over $\mathbf{V}'$. Each word over $\mathbf{V}'$ is a term such that both the union ($\cup$) and the Kleene-star ($\_^*$) do not occur. Note that $[x^-]_{\mathbf{V}'} = \{x^-\}$, cf. $[x^-] = \mathbf{V}^* \setminus \{x\}$. For a language valuation $\mathfrak{v}$ and a language L over $\mathbf{V}'$, we define

$$\hat{\mathfrak{v}}(L) \quad \triangleq \quad \bigcup_{w \in L} \hat{\mathfrak{v}}(w).$$

By using the distributive law of $\cdot$ w.r.t. $\cup$, for all languages L, K and language valuations $\mathfrak{v}$, we have:

$$\hat{\mathfrak{v}}(L \cup K) = \hat{\mathfrak{v}}(L) \cup \hat{\mathfrak{v}}(K) \qquad \hat{\mathfrak{v}}(L \cdot K) = \hat{\mathfrak{v}}(L) \cdot \hat{\mathfrak{v}}(K) \qquad \hat{\mathfrak{v}}(L^*) = \hat{\mathfrak{v}}(L)^*.$$

We can decompose each term t to the set $[t]_{\mathbf{V}'}$ of words over $\mathbf{V}'$ as follows:

Lemma 2. *Let $\mathfrak{v}$ be a language valuation. For all terms t, we have*

$$\hat{\mathfrak{v}}(t) \quad = \quad \hat{\mathfrak{v}}([t]_{\mathbf{V}'}).$$

Proof. By easy induction on t using the equations above. Case $t = x, x^-, \mathsf{I}$: Clear, by $[t]_{\mathbf{V}'} = \{t\}$. Case $t = \bot$: By $\hat{\mathfrak{v}}(\bot) = \emptyset = \hat{\mathfrak{v}}([\bot]_{\mathbf{V}'})$. Case $t = s \cup u$, Case $t = s \cdot u$, Case $t = s^*$: By IH with the equations above. For example, when $t = s \cdot u$, we have

$$\begin{aligned} \hat{\mathfrak{v}}(s \cdot u) = \hat{\mathfrak{v}}(s) \cdot \hat{\mathfrak{v}}(u) &= \hat{\mathfrak{v}}([s]_{\mathbf{V}'}) \cdot \hat{\mathfrak{v}}([u]_{\mathbf{V}'}) && \text{(IH)} \\ &= \hat{\mathfrak{v}}([s]_{\mathbf{V}'} \cdot [u]_{\mathbf{V}'}) = \hat{\mathfrak{v}}([s \cdot u]_{\mathbf{V}'}). && \square \end{aligned}$$

3 The identity inclusion problem

We first consider the *identity inclusion problem* w.r.t. languages:

Given a term t, does $\mathsf{LANG} \models \mathsf{I} \leq t$?

This problem is relatively easily solvable. Since $\mathsf{LANG} \models \mathsf{I} \leq t$ iff $\mathsf{I} \in \hat{\mathfrak{v}}(t)$ for all language valuation $\mathfrak{v}$, it suffices to consider the membership of the empty word I. We use the following facts.

Proposition 3. *For all languages L, K, we have:*

$$\begin{array}{rcl} \mathsf{I} \in L \cup K & \iff & \mathsf{I} \in L \vee \mathsf{I} \in K \\ \mathsf{I} \in L \cdot K & \iff & \mathsf{I} \in L \wedge \mathsf{I} \in K \\ \mathsf{I} \in L^* & \iff & \mathsf{True}. \end{array}$$

Proof. Clear, by definition. □

Lemma 4. *Let $\mathfrak{v}, \mathfrak{v}'$ be language valuations. Assume that for all variables x, $\mathsf{I} \in \mathfrak{v}(x)$ iff $\mathsf{I} \in \mathfrak{v}'(x)$. For all terms t,*

$$\mathsf{I} \in \hat{\mathfrak{v}}(t) \iff \mathsf{I} \in \hat{\mathfrak{v}}'(t).$$

Proof. By easy induction on t using Prop. 3. Case $t = x, x^-$: Clear by the assumption. Case t is a constant: Trivial. For inductive cases, e.g., Case $t = s \cup u$: By using Prop. 3, we have

$$\begin{array}{rcl} \mathsf{I} \in \hat{\mathfrak{v}}(s \cup u) \iff \mathsf{I} \in \hat{\mathfrak{v}}(s) \vee \mathsf{I} \in \hat{\mathfrak{v}}(u) & \iff & \mathsf{I} \in \hat{\mathfrak{v}}'(s) \vee \mathsf{I} \in \hat{\mathfrak{v}}'(u) \\ & \iff & \mathsf{I} \in \hat{\mathfrak{v}}'(s \cup u). \end{array} \quad \text{(IH)}$$

(Similarly for the other inductive cases.) □

By using Lem. 4, it suffices to consider a finite number of valuations, as follows.

Theorem 5. *For all terms t, the following are equivalent:*

1. $\mathsf{LANG} \models \mathsf{I} \leq t$ *(i.e., $\hat{\mathfrak{v}}(\mathsf{I}) \subseteq \hat{\mathfrak{v}}(t)$, for all language valuations $\mathfrak{v}$);*
2. $\hat{\mathfrak{v}}(\mathsf{I}) \subseteq \hat{\mathfrak{v}}(t)$, *for all language valuations $\mathfrak{v}$ over the empty set s.t. for all x, $\mathfrak{v}(x) \subseteq \{\mathsf{I}\}$.*

Proof. 1⇒2: Trivial. 2⇒1: We prove the contraposition. By $\mathsf{LANG} \not\models \mathsf{I} \leq t$, there is a language valuation $\mathfrak{v}$ s.t. $\hat{\mathfrak{v}}(\mathsf{I}) \not\subseteq \hat{\mathfrak{v}}(t)$ (i.e., $\mathsf{I} \notin \hat{\mathfrak{v}}(t)$). Let $\mathfrak{v}^{\langle\rangle}$ be the language valuation over the empty set defined by:

$$\mathfrak{v}^{\langle\rangle}(x) \triangleq \{\mathsf{I} \mid \mathsf{I} \in \mathfrak{v}(x)\}.$$

Then by Lem. 4, $\mathsf{I} \notin \hat{\mathfrak{v}}^{\langle\rangle}(t)$ holds; thus, we have $\hat{\mathfrak{v}}^{\langle\rangle}(\mathsf{I}) \not\subseteq \hat{\mathfrak{v}}^{\langle\rangle}(t)$. □

Corollary 6. *The identity inclusion problem (given a term t, does $\mathsf{LANG} \models \mathsf{I} \leq t$?) is decidable and coNP-complete for KA terms with variable complements.*

Proof. (in coNP): Thm. 5 induces the following non-deterministic polynomial algorithm:

1. Pick up a language valuation $\mathfrak{v}$ over the empty set s.t. for all x, $\mathfrak{v}(x) \subseteq \{\mathsf{I}\}$, non-deterministically.
2. If $\hat{\mathfrak{v}}(\mathsf{I}) \not\subseteq \hat{\mathfrak{v}}(t)$, then return True; otherwise return False.

Then $\mathsf{LANG} \not\models \mathsf{I} \leq t$ if some execution returns True; and $\mathsf{LANG} \models \mathsf{I} \leq t$ otherwise. Hence, the identity inclusion problem is decidable in coNP (as its complemented problem is in NP).

(coNP-hard): Because this problem subsumes the validity problem of propositional formulas in disjunctive normal form, which is a well-known coNP-complete problem [4]. More precisely, given a propositional formula φ in disjunctive normal form, let t be the term obtained from φ by replacing each conjunction $\wedge$ with $\cdot$ and each disjunction $\vee$ with $\cup$ (where we map each positive literal x to the variable

x and each negative literal x^- to the complemented variable x^-); for example, if $\varphi = (x \wedge y^-) \vee (y \vee x^-)$, then $t = (x \cdot y^-) \cup (y \cup x^-)$. Then, for all language valuations $\mathfrak{v}$ (over the empty set s.t. for all $x, \mathfrak{v}(x) \subseteq \{\mathsf{I}\}$), we have: $\hat{\mathfrak{v}}(\mathsf{I}) \subseteq \hat{\mathfrak{v}}(t)$ holds iff φ is True on the valuation $\mathfrak{v}'$, where $\mathfrak{v}'$ is the map mapping each x to True if $\mathsf{I} \in \mathfrak{v}(x)$ and False otherwise. Thus by Thm. 5, $\mathsf{LANG} \models \mathsf{I} \leq t$ iff φ is valid. Hence, the identity inclusion problem is coNP-hard. □

Remark 7. Under the standard language valuation, the identity inclusion problem—given a term t, does $[\mathsf{I}] \subseteq [t]$? (i.e., does $\mathsf{I} \in [t]$?)—is decidable in P (because we can compute "$\mathsf{I} \in [t]$?" by induction on t, as $\mathsf{I} \notin [x]$ and $\mathsf{I} \in [x^-]$ for every variable x). Hence, for KA terms with variable complements, the identity inclusion problem w.r.t. languages is strictly harder than that under the standard language valuation unless P = NP.

4 The variable inclusion problem and the universality problem

Next, we consider the *variable inclusion problem*:

Given a variable x and a term t, does $\mathsf{LANG} \models x \leq t$?

In the identity inclusion problem, if $w \in \hat{\mathfrak{v}}(\mathsf{I}) \setminus \hat{\mathfrak{v}}(t)$, then $w = \mathsf{I}$ should hold; so it suffices to consider the membership of the empty word I. However, in the variable inclusion problem, this situation changes: if $w \in \hat{\mathfrak{v}}(x) \setminus \hat{\mathfrak{v}}(t)$, then w is possibly any word.

Nevertheless, we can overcome the problem above for KA terms with variable complements; more precisely, from a language valuation $\mathfrak{v}$ s.t. $w \in \hat{\mathfrak{v}}(x) \setminus \hat{\mathfrak{v}}(t)$ for some word w, we can construct a language valuation $\mathfrak{v}'$ s.t. $\ell \in \hat{\mathfrak{v}}'(x) \setminus \hat{\mathfrak{v}}'(t)$ for some *letter* ℓ. If such $\mathfrak{v}'$ can be constructed from $\mathfrak{v}$, then considering the membership of letters suffices because we have

$$\begin{aligned} \mathsf{LANG} \not\models x \leq t &\iff w \in \hat{\mathfrak{v}}(x) \setminus \hat{\mathfrak{v}}(t) \text{ for some language valuation } \mathfrak{v} \text{ and word } w && \text{(By definition)} \\ &\iff \ell \in \hat{\mathfrak{v}}'(x) \setminus \hat{\mathfrak{v}}'(t) \text{ for some language valuation } \mathfrak{v}' \text{ and } \textit{letter } \ell \\ & && (\Longrightarrow\text{: By the condition of } \mathfrak{v}'. \Longleftarrow\text{: Trivial by letting } \mathfrak{v} = \mathfrak{v}'.) \end{aligned}$$

Such a language valuation $\mathfrak{v}'$ can be defined as follows:

Definition 8. For a language valuation $\mathfrak{v}$ over a set X and a word w over X, the language valuation $\mathfrak{v}^w$ over the set $\{\ell\}$ (where ℓ is a letter) is defined as follows:

$$\mathfrak{v}^w(x) \quad \triangleq \quad \{\mathsf{I} \mid \mathsf{I} \in \mathfrak{v}(x)\} \cup \{\ell \mid w \in \mathfrak{v}(x)\}.$$

In the following, we prove that $\mathfrak{v}^w$ satisfies the condition of $\mathfrak{v}'$ above, that is, the following conditions:

- $w \in \hat{\mathfrak{v}}(x) \Longrightarrow \ell \in \hat{\mathfrak{v}}^w(x)$;
- $w \notin \hat{\mathfrak{v}}(t) \Longrightarrow \ell \notin \hat{\mathfrak{v}}^w(t)$.

The first condition is clear by the definition of $\mathfrak{v}^w$. We prove the second condition in Lem. 10. We prepare the following fact:

Proposition 9 (cf. Prop. 3)**.** *For all languages L, K and letters a, we have:*

$$\begin{aligned} a \in L \cup K &\iff a \in L \vee a \in K \\ a \in L \cdot K &\iff (a \in L \wedge \mathsf{I} \in K) \vee (\mathsf{I} \in L \wedge a \in K) \\ a \in L^* &\iff a \in L. \end{aligned}$$

Proof. Clear, by definition. □

Lemma 10 (cf. Lem. 4)**.** *Let* $\mathfrak{v}$ *be a language valuation and* w *be a word. For all terms* t*, we have:*

$$\ell \in \hat{\mathfrak{v}}^w(t) \quad \Longrightarrow \quad w \in \hat{\mathfrak{v}}(t).$$

Proof. By induction on t.

Case $t = x, x^-$: By the construction of $\mathfrak{v}^w$, $\ell \in \hat{\mathfrak{v}}^w(x)$ iff $w \in \hat{\mathfrak{v}}(x)$. (Hence, we also have $\ell \in \hat{\mathfrak{v}}^w(x^-)$ iff $w \in \hat{\mathfrak{v}}(x^-)$.)

Case $t = \bot, \mathsf{I}$: By $\ell \notin \hat{\mathfrak{v}}^w(t)$.

Case $t = s \cup u$: We have:

$$\begin{aligned} \ell \in \hat{\mathfrak{v}}^w(s) \cup \hat{\mathfrak{v}}^w(u) &\Longleftrightarrow \ell \in \hat{\mathfrak{v}}^w(s) \vee \ell \in \hat{\mathfrak{v}}^w(u) && \text{(Prop. 9)} \\ &\Longrightarrow w \in \hat{\mathfrak{v}}(s) \vee w \in \hat{\mathfrak{v}}(u) && \text{(IH)} \\ &\Longrightarrow w \in \hat{\mathfrak{v}}(s) \cup \hat{\mathfrak{v}}(u). \end{aligned}$$

Case $t = s \cdot u$: We have:

$$\begin{aligned} \ell \in \hat{\mathfrak{v}}^w(s) \cdot \hat{\mathfrak{v}}^w(u) &\Longleftrightarrow (\ell \in \hat{\mathfrak{v}}^w(s) \wedge \mathsf{I} \in \hat{\mathfrak{v}}^w(u)) \vee (\mathsf{I} \in \hat{\mathfrak{v}}^w(s) \wedge \ell \in \hat{\mathfrak{v}}^w(u)) && \text{(Prop. 9)} \\ &\Longrightarrow (w \in \hat{\mathfrak{v}}(s) \wedge \mathsf{I} \in \hat{\mathfrak{v}}(u)) \vee (\mathsf{I} \in \hat{\mathfrak{v}}(s) \wedge w \in \hat{\mathfrak{v}}(u)) && \text{(IH with Lem. 4)} \\ &\Longrightarrow w \in \hat{\mathfrak{v}}(s) \cdot \hat{\mathfrak{v}}(u) \end{aligned}$$

Case $t = s^*$: We have:

$$\begin{aligned} \ell \in \hat{\mathfrak{v}}^w(s)^* &\Longleftrightarrow \ell \in \hat{\mathfrak{v}}^w(s) && \text{(Prop. 9)} \\ &\Longrightarrow w \in \hat{\mathfrak{v}}(s) && \text{(IH)} \\ &\Longrightarrow w \in \hat{\mathfrak{v}}(s)^* \end{aligned}$$

□

Thus we have obtained the expected condition for $\mathfrak{v}^w$ as follows:

Corollary 11. *Let* $\mathfrak{v}$ *be a language valuation and* w *be a word. For all variables* x *and terms* t*,*

$$w \in \hat{\mathfrak{v}}(x) \setminus \hat{\mathfrak{v}}(t) \quad \Longrightarrow \quad \ell \in \hat{\mathfrak{v}}^w(x) \setminus \hat{\mathfrak{v}}^w(t).$$

Proof. For $\ell \in \hat{\mathfrak{v}}^w(x)$: By the construction of $\mathfrak{v}^w$, $\ell \in \hat{\mathfrak{v}}^w(x)$ iff $w \in \hat{\mathfrak{v}}(x)$. For $\ell \notin \hat{\mathfrak{v}}^w(t)$: By Lem. 10. □

Theorem 12. *For all variables* x *and terms* t*, the following are equivalent:*

1. $\mathsf{LANG} \models x \leq t$*;*
2. $\hat{\mathfrak{v}}(x) \subseteq \hat{\mathfrak{v}}(t)$ *for all language valuations* $\mathfrak{v}$ *over the set* $\{\ell\}$ *s.t.* $\mathfrak{v}(y) \subseteq \{\mathsf{I}, \ell\}$ *for all* y*;*
3. $\hat{\mathfrak{v}}^w(x) \subseteq \hat{\mathfrak{v}}^w(t)$ *for all language valuations* $\mathfrak{v}$ *and words* w*.*

Proof. 1⇒2, 2⇒3: Trivial, as $\hat{\mathfrak{v}}^w(y) \subseteq \{\mathsf{I}, \ell\}$ for all y. 3⇒1: The contraposition is shown by Cor. 11. □

Corollary 13. *The variable inclusion problem (given a variable* x *and a term* t*, does* $\mathsf{LANG} \models x \leq t$*?) is decidable and coNP-complete for KA terms with variable complements.*

Proof. (in coNP): By the condition 2 of Thm. 12, we can give an algorithm as with Cor. 6. (coNP-hard): We give a reduction from the validity problem of propositional formulas in disjunctive normal form, as with Cor. 6. Given a propositional formula φ in disjunctive normal form, let t be the term obtained by the translation in Cor. 6; so, we have that φ is valid iff $\mathsf{LANG} \models \mathsf{I} \leq t$. Then we also have that $\mathsf{LANG} \models \mathsf{I} \leq t$ iff $\mathsf{LANG} \models z \leq z \cdot t$ (where z is a fresh variable); the direction $\Longrightarrow$ is shown by the congruence law, and the direction $\Longleftarrow$ is shown by the substitution law. Therefore, we have that φ is valid iff $\mathsf{LANG} \models z \leq z \cdot t$; thus, the variable inclusion problem is coNP-hard. □

4.1 Generalization from variables to composition-free terms

The proof above applies to not only variables but also terms t having the following property: For all language valuations $\mathfrak{v}$,

$$\text{for all non-empty words } w, \qquad w \in \hat{\mathfrak{v}}(t) \Longrightarrow \ell \in \hat{\mathfrak{v}}^w(t). \tag{L_1}$$

(This condition is intended for composition-free terms (Lem. 15). This is generalized to (L_n) in Sect. 5.1.) If t satisfies the condition (L_1), then combining with Lem. 10 (and with Lem. 4 for the empty word I) yields that for all language valuations $\mathfrak{v}$ and words w,

$$w \in \hat{\mathfrak{v}}(t) \setminus \hat{\mathfrak{v}}(s) \quad \Longrightarrow \quad \begin{cases} \ell \in \hat{\mathfrak{v}}^w(t) \setminus \hat{\mathfrak{v}}^w(s) & (\text{if } w \neq \mathsf{I}) \\ \mathsf{I} \in \hat{\mathfrak{v}}^w(t) \setminus \hat{\mathfrak{v}}^w(s) & (\text{if } w = \mathsf{I}) \end{cases}.$$

Hence, we have the following:

Theorem 14 (cf. Thm. 12)**.** *For all terms t, s, if t satisfies (L_1), then the following are equivalent:*

1. $\mathsf{LANG} \models t \leq s$;
2. $\hat{\mathfrak{v}}(t) \subseteq \hat{\mathfrak{v}}(s)$ *for all language valuations* $\mathfrak{v}$ *over the set* $\{\ell\}$ *s.t.* $\mathfrak{v}(x) \subseteq \{\mathsf{I}, \ell\}$ *for all* x;
3. $\hat{\mathfrak{v}}^w(t) \subseteq \hat{\mathfrak{v}}^w(s)$ *for all language valuations* $\mathfrak{v}$ *and words* w.

Proof. As with Thm. 12 (use the above, instead of Cor. 11). □

Thm. 14 can apply to composition-free terms. We say that a term t is *composition-free* if composition $(\cdot)$ nor Kleene-star $(\_^*)$ does not occur in t.

Lemma 15. *Every composition-free terms t satisfies the condition (L_1).*

Proof. By easy induction on t. Case $t = x, x^-$: By the definition of $\mathfrak{v}^w$. Case $t = \mathsf{I}$: By that $w \notin \hat{\mathfrak{v}}(\mathsf{I})$ holds for all non-empty words w. Case $t = \bot$: By $w \notin \hat{\mathfrak{v}}(\bot)$ always. Case $t = s \cup u$: By IH, we have that $w \in \hat{\mathfrak{v}}(s) \cup \hat{\mathfrak{v}}(u) \iff w \in \hat{\mathfrak{v}}(s) \vee w \in \hat{\mathfrak{v}}(u) \Longrightarrow \ell \in \hat{\mathfrak{v}}^w(s) \vee \ell \in \hat{\mathfrak{v}}^w(u) \iff \ell \in \hat{\mathfrak{v}}^w(s) \cup \hat{\mathfrak{v}}^w(u)$. □

Corollary 16. *The following problem is coNP-complete for KA terms with variable complements:*

Given a composition-free term t and a term s, does $\mathsf{LANG} \models t \leq s$ hold?

Proof. (coNP-hard): By Cor. 6, as t is possibly I. (in coNP): By Thm. 14 with Lem. 15, we can give an algorithm (from the condition 2 of Thm. 14) as with Cor. 13. □

Remark 17. Lem. 15 fails for non-composition-free terms. For example, when $\mathfrak{v}(x) = \{a\}$, we have

$$aa \in \hat{\mathfrak{v}}(xx) \qquad\qquad \ell \notin \hat{\mathfrak{v}}^{aa}(xx).$$

(Note that $\hat{\mathfrak{v}}^{aa}(xx) = \emptyset$, as $\hat{\mathfrak{v}}^{aa}(x) = \emptyset$ by $\mathfrak{v}(x) = \{a\}$.)

4.2 The universality problem

The *universality problem* is the following problem:

$$\text{Given a term } t, \text{ does } \mathsf{LANG} \models \top \leq t?$$

As a consequence of Cor. 16, the universality problem is also decidable and coNP-complete.

Corollary 18. *The universality problem is decidable and coNP-complete for KA terms with variable complements.*

Proof. (in coNP): We can apply Cor. 16 because the term $x \cup x^-$ is composition-free and $\mathsf{LANG} \models \top = x \cup x^-$ holds. (coNP-hard): Similar to Cor. 13. Given a propositional formula φ in disjunctive normal form, let t be the term obtained by the translation in Cor. 6; so, we have that φ is valid iff $\mathsf{LANG} \models \mathsf{I} \leq t$. Then we also have that $\mathsf{LANG} \models \mathsf{I} \leq t$ iff $\mathsf{LANG} \models \top \leq \top \cdot t$, which is proved as follows. $\Longrightarrow$: By the congruence law. $\Longleftarrow$: We prove the contraposition. Assume $\mathsf{LANG} \not\models \mathsf{I} \leq t$; then $\mathsf{I} \notin \hat{\mathfrak{v}}(t)$ for some language valuation $\mathfrak{v}$. Then $\mathsf{I} \notin \hat{\mathfrak{v}}(\top \cdot t)$ holds; thus, $\mathsf{LANG} \not\models \top \leq \top \cdot t$. Hence, the universality problem is coNP-complete. □

Remark 19. In the standard language equivalence, because $[\mathbf{V}^*] = [\top]$ (and the constant $\top$ is usually not a primitive symbol of regular expressions), the universality problem is always of the form: $[\mathbf{V}^*] = [t]$. However, $\mathsf{LANG} \models \mathbf{V}^* \leq t$ is different from $\mathsf{LANG} \models \top \leq t$, as $\mathsf{LANG} \not\models \mathbf{V}^* = \top$.

Remark 20. Under the standard language equivalence, the universality problem—given a term t, does $[\top] \subseteq [t]$? (i.e., does $[t] = \mathbf{V}^*$?)—is PSPACE-hard [5, 8, 12]. Hence, for KA terms with variable complements, the universality problem w.r.t. languages is strictly easier (cf. Remark 7) than that under the standard language equivalence unless NP = PSPACE.

5 The word inclusion problem

Let $\mathbf{V}' = \{x, x^- \mid x \in \mathbf{V}\}$. The *word inclusion problem* is the following problem:

$$\text{Given a word } w \text{ over } \mathbf{V}' \text{ and a term } t, \text{ does } \mathsf{LANG} \models w \leq t?$$

As Remark 17 shows, we cannot apply the method given in Sect. 4 straightforwardly. Nevertheless, we can solve this problem by generalizing the language valuation of Def. 8, as follows. The valuations in Defs. 8, 21 are given by the first author.

Definition 21 (*words-to-letters valuations*)**.** For a language valuation $\mathfrak{v}$ over a set X and words $w_0, \ldots, w_{n-1}$ over X (where $n \geq 0$), the language valuation $\mathfrak{v}^{\langle w_0,\ldots,w_{n-1}\rangle}$ over the set $\{\ell_0, \ldots, \ell_{n-1}\}$ (where $\ell_0, \ldots, \ell_{n-1}$ are pairwise distinct letters) is defined as follows:

$$\mathfrak{v}^{\langle w_0,\ldots,w_{n-1}\rangle}(x) \quad \triangleq \quad \{\ell_i \ldots \ell_{j-1} \mid 0 \leq i \leq j \leq n \ \wedge \ w_i \ldots w_{j-1} \in \mathfrak{v}(x)\}.$$

(Note that the language valuation $\mathfrak{v}^w$ (Def. 8) is the case $n = 1$ of Def. 21 and the language valuation $\mathfrak{v}^{\langle\rangle}$ in the proof of Thm. 5 is the case $n = 0$ of Def. 21.)

By using words-to-letters valuations, we can naturally strengthen the results in Sect. 4 from variables to words. We prepare the following fact:

Proposition 22 (cf. Prop. 9)**.** *For all languages L, K and words w,*

$$\begin{aligned} w \in L \cup K &\iff w \in L \vee w \in K \\ w \in L \cdot K &\iff \exists v, v' \ s.t. \ w = vv', \ v \in L \wedge v' \in K \\ w \in L^* &\iff \exists n \in \mathbb{N}, \exists v_0, \ldots, v_{n-1} \ s.t. \ w = v_0 \ldots v_{n-1}, \forall i < n, \ v_i \in L. \end{aligned}$$

Proof. By definition. □

Lemma 23 (cf. Lem. 10)**.** *Let* $\mathfrak{v}$ *be a language valuation and* $w_0,\dots,w_{n-1}$ *be words (where* $n \geq 0$*). For all terms* t *and* $0 \leq i \leq j \leq n$*, we have:*

$$\ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t) \quad \Longrightarrow \quad w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(t).$$

Proof. By induction on t.

Case $t = x, x^-$: By the construction of $\mathfrak{v}^{\langle w_0,\dots,w_{n-1}\rangle}$, $\ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(x)$ iff $w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(x)$. (Hence, we also have $\ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(x^-)$ iff $w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(x^-)$.)

Case $t = \bot$, Case $t = \mathsf{l}$ where $i < j$: By $\ell_i \dots \ell_{j-1} \notin \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t)$.

Case $t = \mathsf{l}$ where $i = j$: By $\mathsf{l} \in \hat{\mathfrak{v}}(\mathsf{l})$.

Case $t = s \cup u$: We have

$$\begin{aligned}
\ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s \cup u) &\Longleftrightarrow \ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s) \vee \ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(u) && \text{(Prop. 22)}\\
&\Longrightarrow w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(s) \vee w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(u) && \text{(IH)}\\
&\Longrightarrow w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(s \cup u).
\end{aligned}$$

Case $t = s \cdot u$: We have

$$\begin{aligned}
\ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s \cdot u) &\Longleftrightarrow \bigvee_{i \leq k \leq j} (\ell_i \dots \ell_{k-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s) \wedge \ell_k \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(u)) && \text{(Prop. 22)}\\
&\Longrightarrow \bigvee_{i \leq k \leq j} (w_i \dots w_{k-1} \in \hat{\mathfrak{v}}(s) \wedge w_k \dots w_{j-1} \in \hat{\mathfrak{v}}(u)) && \text{(IH)}\\
&\Longrightarrow w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(s \cdot u).
\end{aligned}$$

Case $t = s^*$: We have

$$\begin{aligned}
\ell_i \dots \ell_{j-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s^*) &\Longleftrightarrow \exists m \in \mathbb{N}, \bigvee_{i=k_0 \leq k_1 \leq \dots \leq k_m = j} \bigwedge_{l=1}^{m} (\ell_{k_{l-1}} \dots \ell_{k_l-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s)) && \text{(Prop. 22)}\\
&\Longrightarrow \exists m \in \mathbb{N}, \bigvee_{i=k_0 \leq k_1 \leq \dots \leq k_m = j} \bigwedge_{l=1}^{m} (w_{k_{l-1}} \dots w_{k_l-1} \in \hat{\mathfrak{v}}(s)) && \text{(IH)}\\
&\Longrightarrow w_i \dots w_{j-1} \in \hat{\mathfrak{v}}(s^*).
\end{aligned}$$

□

Corollary 24 (cf. Cor. 11)**.** *Let* $\mathfrak{v}$ *be a language valuation,* w *be a word,* $w_0,\dots,w_{n-1}$ *be words s.t.* $w = w_0 \dots w_{n-1}$*. For all words* v *over* $\mathbf{V}'$ *of length* n *and all terms* t*,*

$$w \in \hat{\mathfrak{v}}(v) \setminus \hat{\mathfrak{v}}(t) \quad \Longrightarrow \quad \ell_0 \dots \ell_{n-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(v) \setminus \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t).$$

Proof. For $\ell_0 \dots \ell_{n-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(v)$: Let $v = x_0 \dots x_{n-1}$. For each $i < n$, by the construction of $\mathfrak{v}^{\langle w_0,\dots,w_{n-1}\rangle}$, $\ell_i \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(x_i)$ iff $w_i \in \hat{\mathfrak{v}}(x_i)$. Thus, we have that $\ell_0 \dots \ell_{n-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(v)$. For $\ell_0 \dots \ell_{n-1} \notin \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t)$: By Lem. 23. □

Theorem 25 (cf. Thm. 12)**.** *For all words* v *over* $\mathbf{V}'$ *of length* n *and all terms* t*, the following are equivalent:*

1. $\mathsf{LANG} \models v \leq t$;
2. $\hat{\mathfrak{v}}(v) \subseteq \hat{\mathfrak{v}}(t)$ *for all language valuations* $\mathfrak{v}$ *s.t.* $\mathfrak{v}(x) \subseteq \{\ell_i \dots \ell_j \mid 0 \leq i \leq j \leq n\}$ *for all* x;
3. $\hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(v) \subseteq \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t)$ *for all language valuations* $\mathfrak{v}$ *and words* $w_0,\dots,w_{n-1}$.

Proof. 1⇒2, 2⇒3: Trivial. 3⇒1: The contraposition is shown by Cor. 24. □

Corollary 26 (cf. Cor. 13)**.** *The word inclusion problem (given a word w and a term t, does $\mathsf{LANG} \models w \leq t$?) is decidable and coNP-complete for KA terms with variable complements.*

Proof. (coNP-hard): By Cor. 6, as w is possibly l. (in coNP): By the condition 2 of Thm. 25, we can give an algorithm as with Cor. 13. □

5.1 Generalization from words to star-free terms

We can apply Thm. 25 to not only words over $\mathbf{V}'$ but also terms t having the following property:

For all language valuations $\mathfrak{v}$ and non-empty words w, for some $w_0,\dots,w_{n-1}$ s.t. $w = w_0 \dots w_{n-1}$,
$w \in \hat{\mathfrak{v}}(t) \Longrightarrow \ell_0 \dots \ell_{n-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t)$. (L_n)

If t satisfies the condition (L_n), then combining with Lem. 23 (and with Lem. 4 for the empty word l) yields that for all language valuations $\mathfrak{v}$ and words w, for some words $w_0,\dots,w_{n-1}$ s.t. $w = w_0 \dots w_{n-1}$, we have:

$$w \in \hat{\mathfrak{v}}(t) \setminus \hat{\mathfrak{v}}(s) \quad \Longrightarrow \quad \begin{cases} \ell_0 \dots \ell_{n-1} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t) \setminus \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s) & (\text{if } w \neq \mathsf{l}) \\ \mathsf{l} \in \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t) \setminus \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s) & (\text{if } w = \mathsf{l}) \end{cases}.$$

Hence, we have the following:

Theorem 27 (cf. Thm. 14)**.** *For all terms t, s, if t satisfies (L_n), the following are equivalent:*

1. $\mathsf{LANG} \models t \leq s$;
2. $\hat{\mathfrak{v}}(t) \subseteq \hat{\mathfrak{v}}(s)$ *for all language valuations* $\mathfrak{v}$ *over the set* $\{\ell_0,\dots,\ell_{n-1}\}$ *s.t.* $\mathfrak{v}(x) \subseteq \{\ell_i \dots \ell_j \mid 0 \leq i \leq j \leq n\}$ *for all* x;
3. $\hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(t) \subseteq \hat{\mathfrak{v}}^{\langle w_0,\dots,w_{n-1}\rangle}(s)$ *for all language valuations* $\mathfrak{v}$ *and words* $w_0,\dots,w_{n-1}$.

Proof. As with Thm. 25 (use the above, instead of Cor. 24). □

By using Thm. 27, we can generalize Cor. 26 from words to star-free terms. We say that a term t is *star-free* if the Kleene-star $(\_^*)$ does not occur in t.

Lemma 28 (cf. Lem. 15)**.** *Every star-free term t satisfies (L_n) for some n.*

Proof. Because the set $[t]_{\mathbf{V}'}$ is finite as t is star-free, let n be the maximal length among words in $[t]_{\mathbf{V}'}$. Let $\mathfrak{v}$ be a language valuation and let w be a non-empty word such that $w \in \hat{\mathfrak{v}}(t)$. Since $\hat{\mathfrak{v}}(t) = \hat{\mathfrak{v}}([t]_{\mathbf{V}'})$ (Lem. 2), there is a word $v \in [t]_{\mathbf{V}'}$ such that $w \in \hat{\mathfrak{v}}(v)$. Let $v = x_0 \dots x_{m-1}$ (note that $m \geq 1$, as w is non-empty and $w \in \hat{\mathfrak{v}}(v)$). Since $w \in \hat{\mathfrak{v}}(x_0 \dots x_{m-1})$, there are $w_0,\dots,w_{m-1}$ of $w = w_0 \dots w_{m-1}$ such that $w_i \in \hat{\mathfrak{v}}(x_i)$ for every i. Let $\mathfrak{v}' \triangleq \mathfrak{v}^{\langle w_0,\dots,w_{m-1},\mathsf{l},\dots,\mathsf{l}\rangle}$, where the length of the sequence is n. Then, we have $\ell_i \in \hat{\mathfrak{v}}'(x_i)$ for every $0 \leq i \leq m-2$ and $\ell_{m-1}\ell_m \dots \ell_{n-1} \in \hat{\mathfrak{v}}'(x_{m-1})$; thus, $\ell_0 \dots \ell_{n-1} \in \hat{\mathfrak{v}}'(x_0 \dots x_{m-1}) = \hat{\mathfrak{v}}'(v) \subseteq \hat{\mathfrak{v}}'(t)$. Hence, this completes the proof. □

Corollary 29. *The following problem is coNP-complete for KA terms with variable complements:*

Given a star-free term t and a term s, does $\mathsf{LANG} \models t \leq s$*?*

Proof. (coNP-hard): By Cor. 6, as t is possibly I. (in coNP): By Lem. 28, we can give an algorithm as with Cor. 26. □

5.2 words-to-letters valuation property

Finally, we show the following property; thus, we have that words-to-letters valuations are sufficient for the equational theory of (full) KA terms with variable complements.

Corollary 30 (words-to-letters valuation property)**.** *For all terms t, s, the following are equivalent:*

1. $\mathsf{LANG} \not\models t \leq s$*;*
2. *there is a words-to-letters valuation $\mathfrak{v}$ such that $\hat{\mathfrak{v}}(t) \not\subseteq \hat{\mathfrak{v}}(s)$.*

Proof. 2⇒1: Trivial. 1⇒2: Since $\mathsf{LANG} \not\models t \leq s$, there is a language valuation $\mathfrak{v}$ such that $\hat{\mathfrak{v}}(t) \not\subseteq \hat{\mathfrak{v}}(s)$. Since $\hat{\mathfrak{v}}(t) = \hat{\mathfrak{v}}([t]_{\mathbf{V}'})$ (Lem. 2), there is a word $v \in [t]_{\mathbf{V}'}$ such that $\hat{\mathfrak{v}}(v) \not\subseteq \hat{\mathfrak{v}}(s)$ (i.e., $\mathsf{LANG} \not\models v \leq s$). Let n be the length of v. By Thm. 25, there are a words-to-letters valuation $\mathfrak{v}$ and words $w_0, \ldots, w_{n-1}$ such that $\hat{\mathfrak{v}}^{\langle w_0,\ldots,w_{n-1}\rangle}(v) \not\subseteq \hat{\mathfrak{v}}^{\langle w_0,\ldots,w_{n-1}\rangle}(s)$. Thus $\hat{\mathfrak{v}}^{\langle w_0,\ldots,w_{n-1}\rangle}(t) \not\subseteq \hat{\mathfrak{v}}^{\langle w_0,\ldots,w_{n-1}\rangle}(s)$, as $v \in [t]_{\mathbf{V}'}$ (Lem. 2). Hence this completes the proof. □

6 On the equational theory of words with variable complements

We prove that the equational theory of words over $\mathbf{V}'$ coincides with the word equivalence (Thm. 32). We give language valuations for separating two distinct words based on words-to-letters valuations.

Lemma 31. *Let $w = x_0 \ldots x_{n-1}$ and $v = y_0 \ldots y_{m-1}$ be words over $\mathbf{V}'$, where $n \leq m$. Let $\mathfrak{v}$ be a language valuation over $\{\ell_0, \ldots, \ell_{n-1}\}$ such that*

- *for all $i < n$, $\ell_i \in \hat{\mathfrak{v}}(x_i)$;*
- *for all $i < m$ and $i \leq j \leq n$, $\ell_i \ldots \ell_{j-1} \in \hat{\mathfrak{v}}(y_i)$ iff $(y_i = x_i \wedge j = i+1)$.*

If $w \neq v$, then $\ell_0 \ldots \ell_{n-1} \in \hat{\mathfrak{v}}(w) \setminus \hat{\mathfrak{v}}(v)$. Such a language valuation $\mathfrak{v}$ always exists.

Proof. Since $\ell_i \in \mathfrak{v}(x_i)$ for all i, we have $\ell_0 \ldots \ell_{n-1} \in \hat{\mathfrak{v}}(w)$. Assume that $\ell_0 \ldots \ell_{n-1} \in \hat{\mathfrak{v}}(y_0 \ldots y_{m-1})$. For $i = 0$, by the condition of $\hat{\mathfrak{v}}(y_i)$ (i.e., $\ell_i \ldots \ell_{j-1} \notin \hat{\mathfrak{v}}(y_i)$ unless $j = i+1$), we should have $\ell_i \in \hat{\mathfrak{v}}(y_i)$, $\ell_{i+1} \ldots \ell_{n-1} \in \hat{\mathfrak{v}}(y_{i+1} \ldots y_{m-1})$, and $y_i = x_i$. By using the same argument iteratively, the condition above should hold for all $i < n$; thus, we have $\mathsf{I} \in \hat{\mathfrak{v}}(y_n \ldots y_{m-1})$ and $y_0 \ldots y_{n-1} = x_0 \ldots x_{m-1}$. Since $\mathsf{I} \notin \hat{\mathfrak{v}}(y_n)$, we have $y_n \ldots y_{m-1} = \mathsf{I}$; thus, $m = n$. However this yields $w = x_0 \ldots x_{n-1} = y_0 \ldots y_{m-1} = v$, which contradicts the assumption. Hence $\ell_0 \ldots \ell_{n-1} \in \hat{\mathfrak{v}}(w) \setminus \hat{\mathfrak{v}}(v)$. Additionally, such a language valuation $\mathfrak{v}$ always exists as follows. If some conditions conflict, then the first condition ($\ell_i \in \hat{\mathfrak{v}}(x_i)$) and the second condition when $j = i+1$ ($\ell_i \in \hat{\mathfrak{v}}(y_i)$) are for some i. If $y_i = x_i$, then $\ell_i \in \hat{\mathfrak{v}}(y_i) = \hat{\mathfrak{v}}(x_i)$, so it does not conflict to the condition $\ell_i \in \hat{\mathfrak{v}}(x_i)$; If $y_i = x_i^-$ (or $y_i^- = x_i$), then $\ell_i \notin \hat{\mathfrak{v}}(y_i)$, so it does not conflict to the condition $\ell_i \in \hat{\mathfrak{v}}(x_i)$. Otherwise, they are not conflicted, as the variables occurring in x_i and y_i are different. Thus, in either case, conditions are not conflicted. Hence, this completes the proof. □

Theorem 32 (Completeness for words with variable complements)**.** *For all words w, v over $\mathbf{V}'$,*

$$\mathsf{LANG} \models w = v \quad \Longleftrightarrow \quad w = v.$$

Proof. $\Longleftarrow$: Clear. $\Longrightarrow$: The contraposition is shown by Lem. 31. □

Remark 33. Since $[w]_{\mathbf{V}'} = \{w\}$, Thm. 32 also shows that: for all words w, v over $\mathbf{V}'$,

$$[w]_{\mathbf{V}'} = [v]_{\mathbf{V}'} \quad \Longleftrightarrow \quad \mathsf{LANG} \models w = v.$$

However, for general terms, the direction $\Longleftarrow$ fails: For example, when $x \neq y$,

$$\mathsf{LANG} \models x \cup x^- = y \cup y^- \qquad\qquad [x \cup x^-]_{\mathbf{V}'} \neq [y \cup y^-]_{\mathbf{V}'}.$$

(The direction $\Longrightarrow$ always holds by Lem. 2.) Thus, we need more axioms to characterize the equational theory.

Remark 34. As (‡) and Remark 1, for all words w, v over $\mathbf{V}'$,

$$\mathsf{LANG} \models w = v \quad \Longrightarrow \quad [w] = [v].$$

However, the converse direction fails even for words (e.g., $w = x^-$ and $v = x^- x^-$).

6.1 Separating one-variable words with small number of letters

We write LANG_n for the class of language models over a set of cardinality at most n. We write

$$\mathsf{LANG}_n \models t = s \quad \overset{\Delta}{\Longleftrightarrow} \quad \hat{\mathfrak{v}}(t) = \hat{\mathfrak{v}}(s) \text{ holds for all (language) valuations } \mathfrak{v} \text{ on } S\text{-algebras in } \mathsf{LANG}_n.$$

Notice that words-to-letters valuations need an unbounded number of letters; so the proof of Thm. 32 cannot directly apply to the class LANG_n. Nevertheless, for *one-variable* words (i.e., words over the set $\{z, z^-\}$ where z is a variable), we can show completeness theorems (cf. Thm. 32) of the equational theory over LANG_n, as Thms. 35, 36. The valuation in the proof of Thm. 36 is given by the second author.

For a word $w = x_0 \dots x_{n-1} \in \{z, z^-\}^*$ and $x \in \{z, z^-\}$, we write $\|w\|_x$ for the number $\#(\{0 \leq i < n \mid x_i = x\})$. For a letter a and $n \in \mathbb{N}$, we write a^n for the word $a \dots a$ of length n.

Theorem 35. *For all words $w, v \in \{z, z^-\}^*$, we have:*

$$\mathsf{LANG}_1 \models w = v \quad \Longleftrightarrow \quad \|w\|_z = \|v\|_z \wedge \|w\|_{z^-} = \|v\|_{z^-}.$$

Proof. $\Longleftarrow$: By the following commutative law: for all language valuations $\mathfrak{v}$ over a set of cardinality at most 1, $\hat{\mathfrak{v}}(zz^-) = \hat{\mathfrak{v}}(z^- z)$. $\Longrightarrow$: If $\|w\|_z < \|v\|_z$, then let $\mathfrak{v}$ be the language valuation defined by $\mathfrak{v}(z) = \{a\}$. Then $a^{\|w\|_z} \in \hat{\mathfrak{v}}(w) \setminus \hat{\mathfrak{v}}(v)$; thus $\mathsf{LANG}_1 \not\models w = v$. If $\|w\|_{z^-} < \|v\|_{z^-}$, then let $\mathfrak{v}$ be the language valuation defined by $\mathfrak{v}(z) = \{a\}^* \setminus \{a\}$. Then $a^{\|w\|_{z^-}} \in \hat{\mathfrak{v}}(w) \setminus \hat{\mathfrak{v}}(v)$. If $\|w\|_z > \|v\|_z$ (resp. $\|w\|_{z^-} > \|v\|_{z^-}$), then similarly to the cases above. □

Theorem 36. *For all words $w, v \in \{z, z^-\}^*$, we have:*

$$\mathsf{LANG}_2 \models w = v \quad \Longleftrightarrow \quad \mathsf{LANG} \models w = v \quad \Longleftrightarrow \quad w = v.$$

Proof. The two $\Longleftarrow$ are clear by definition. We prove $w \neq v \Longrightarrow \mathsf{LANG}_2 \not\models w = v$. By Thm. 35, it suffices to show the case when $\|w\|_z = \|v\|_z$ and $\|w\|_{z^-} = \|v\|_{z^-}$. Let $n \overset{\Delta}{=} \|w\|_{z^-} = \|v\|_{z^-}$ and let w, v be as follows:

$$w = z^{c_0} z^- z^{c_1} \dots z^- z^{c_n}$$
$$v = z^{d_0} z^- z^{d_1} \dots z^- z^{d_n}.$$

Since $w \neq v$, there is $i \leq n$ such that $c_j = d_j$ for all $j < i$ and $c_i \neq d_i$. Without loss of generality, we can assume $c_i < d_i$. Now, we consider the following language valuation $\mathfrak{v}$ over $A \triangleq \{a,b\}$:

$$\mathfrak{v}(z) \triangleq [(aA^*) \cap (A^*a)] = \{c_0 \dots c_{n-1} \in \{a,b\}^* \mid n \geq 1, c_0 = a, c_{n-1} = a\}.$$

Then $a^{(\Sigma_{j=0}^{i} c_j)} b a^{(\Sigma_{j=i+1}^{n} c_j)} \in \hat{\mathfrak{v}}(w)$, as $a \in \hat{\mathfrak{v}}(z)$ and $\mathsf{l}, b \in \hat{\mathfrak{v}}(z^-)$. Assume, towards contradiction, that $a^{(\Sigma_{j=0}^{i} c_j)} b a^{(\Sigma_{j=i+1}^{n} c_j)} \in \hat{\mathfrak{v}}(v)$. Each z occurring in v should map to a, as $(\sum_{j=0}^{n} c_j) = (\sum_{j=0}^{n} d_j)$ and every word in $\mathfrak{v}(z)$ except for a has at least two occurrences of a. The $(\sum_{j=0}^{i} c_j)$-th occurrence and $((\sum_{j=0}^{i} c_j)+1)$-th occurrence of z^- are adjacent (since $(\sum_{j=0}^{i} c_j) < (\sum_{j=0}^{i} d_j)$). Combining them yields $b \in \hat{\mathfrak{v}}(\mathsf{l})$, thus reaching a contradiction. Hence, $a^{(\Sigma_{j=0}^{i} c_j)} b a^{(\Sigma_{j=i+1}^{n} c_j)} \in \hat{\mathfrak{v}}(w) \setminus \hat{\mathfrak{v}}(v)$. This completes the proof. □

The proof of Thms. 35, 36 only applies to one-variable words. We leave open Thms. 35, 36 for many variables words (cf. Thm. 32).

7 Conclusion and future work

We have introduced words-to-letters valuations. By using them, we have shown the decidability and complexity of the identity/variable/word inclusion problems (Cors. 6, 13, 26) and the universality problem (Cor. 18) of the equational theory of KA terms with variable complements w.r.t. languages; in particular, the inequational theory $t \leq s$ is coNP-complete when t does not contain Kleene-star (Cor. 29). Additionally, we have proved a completeness theorem for words with variable complements w.r.t. languages (Thm. 32); moreover, for one-variable words, the equational theory over LANG coincides with that over LANG_2 (Thm. 36).

A natural interest is to extend our decidability results, e.g., for full KA terms with variable complements. As Cor. 30 shows, even for full terms, words-to-letters valuations are sufficient valuations in investigating the equational theory. The first author conjectures that the equational theory of KA terms with variable complements is decidable, possibly by combining the technique like *saturable paths* [9] (which were introduced for the equational theory w.r.t. binary relations). Additionally, we leave open the (finite) axiomatizability of the equational theory (including that over sets of bounded cardinality; cf. Sect. 6.1).

Acknowledgement

We would like to thank the anonymous reviewers for their useful comments. This work was supported by JSPS KAKENHI Grant Number JP21K13828 and JST ACT-X Grant Number JPMJAX210B, Japan.

References

[1] Hajnal Andréka, Szabolcs Mikulás & István Németi (2011): *The equational theory of Kleene lattices*. *Theoretical Computer Science* 412(52), pp. 7099–7108, doi:10.1016/J.TCS.2011.09.024.

[2] S. L. Bloom, Z. Ésik & Gh. Stefanescu (1995): *Notes on equational theories of relations*. *algebra universalis* 33(1), pp. 98–126, doi:10.1007/BF01190768.

[3] John H. Conway (1971): *Regular Algebra and Finite Machines*. Chapman and Hall.

[4] Stephen A. Cook (1971): *The complexity of theorem-proving procedures*. In: *STOC*, ACM, p. 151–158, doi:10.1145/800157.805047.

[5] Harry B. Hunt III, Daniel J. Rosenkrantz & Thomas G. Szymanski (1976): *On the equivalence, containment, and covering problems for the regular and context-free languages*. Journal of Computer and System Sciences 12(2), pp. 222–268, doi:10.1016/S0022-0000(76)80038-4.

[6] S. C. Kleene (1951): *Representation of Events in Nerve Nets and Finite Automata*. Technical Report, RAND Corporation. Available at `https://www.rand.org/pubs/research_memoranda/RM704.html`.

[7] Dexter Kozen & Frederick Smith (1996): *Kleene algebra with tests: Completeness and decidability*. In: *CSL*, *LNCS* 1258, Springer, pp. 244–259, doi:10.1007/3-540-63172-0_43.

[8] A. R. Meyer & L. J. Stockmeyer (1972): *The equivalence problem for regular expressions with squaring requires exponential space*. In: *SWAT*, IEEE, pp. 125–129, doi:10.1109/SWAT.1972.29.

[9] Yoshiki Nakamura (2023): *Existential Calculi of Relations with Transitive Closure: Complexity and Edge Saturations*. In: *LICS*, IEEE, pp. 1–13, doi:10.1109/LICS56636.2023.10175811.

[10] Kan Ching Ng (1984): *Relation algebras with transitive closure*. Ph.D. thesis, University of California.

[11] Damien Pous & Jana Wagemaker (2022): *Completeness Theorems for Kleene Algebra with Top*. In: *CONCUR*, 243, Schloss Dagstuhl, pp. 26:1–26:18, doi:10.4230/LIPICS.CONCUR.2022.26.

[12] Larry J. Stockmeyer & Albert R. Meyer (1973): *Word problems requiring exponential time (Preliminary Report)*. In: *STOC*, ACM, pp. 1–9, doi:10.1145/800125.804029.

[13] Alfred Tarski (1941): *On the Calculus of Relations*. The Journal of Symbolic Logic 6(3), pp. 73–89, doi:10.2307/2268577.

[14] Ken Thompson (1968): *Programming Techniques: Regular expression search algorithm*. Communications of the ACM 11(6), pp. 419–422, doi:10.1145/363347.363387.

A A direct proof of the coincidence between the equational theory w.r.t. languages and the language equivalence for KA terms

(In this section, we use the notations of Sect. 2.)

We say that a term t is a *KA term* if the complement $(\_^-)$ does not occur in t. Recall language valuations for languages in Sect. 2.3.

Lemma 37 (cf. Lem. 2)**.** *Let* $\mathfrak{v}$ *be a language valuation. For all KA terms* t*, we have*

$$\hat{\mathfrak{v}}(t) \quad = \quad \hat{\mathfrak{v}}([t]).$$

Proof. By Lem. 2, as $[t] = [t]_{\mathbf{V}'}$ (since KA terms do not contain the complement). □

Theorem 38. *For all KA terms* t, s*,*

$$\mathsf{LANG} \models t = s \quad \Longleftrightarrow \quad [t] = [s].$$

Proof. We have

$$\begin{array}{lll}
\mathsf{LANG} \models t = s & \Longrightarrow & [t] = [s] \qquad ([_] \text{ is an instance of language valuations}) \\
& \Longrightarrow & \text{for all language valuations } \mathfrak{v}, \hat{\mathfrak{v}}([t]) = \hat{\mathfrak{v}}([s]) \\
& \Longleftrightarrow & \text{for all language valuations } \mathfrak{v}, \hat{\mathfrak{v}}(t) = \hat{\mathfrak{v}}(s) \qquad \text{(Lem. 37)} \\
& \Longleftrightarrow & \mathsf{LANG} \models t = s. \qquad \text{(By definition)}
\end{array}$$

□

Solving the Weighted HOM-Problem With the Help of Unambiguity

Andreea-Teodora Nász
Leipzig University
Leipzig, Germany
Faculty of Mathematics and Computer Science
PO box 100 920, 04009 Leipzig, Germany
nasz@informatik.uni-leipzig.de

The *HOM-problem*, which asks whether the image of a regular tree language under a tree homomorphism is again regular, is known to be decidable by [Godoy, Giménez, Ramos, Àlvarez: The HOM problem is decidable. STOC (2010)]. Research on the weighted version of this problem, however, is still in its infancy since it requires customized investigations. In this paper we address the weighted HOM-problem and strive to keep the underlying semiring as general as possible. In return, we restrict the input: We require the tree homomorphism h to be *tetris-free*, a condition weaker than injectivity, and for the given weighted tree automaton, we propose an ambiguity notion with respect to h. These assumptions suffice to ensure decidability of the thus restricted HOM-problem for all zero-sum free semirings by allowing us to reduce it to the (decidable) unweighted case.

1 Introduction

Over the past decades, various extensions to the well-known model of finite-state automata have been proposed. These acceptors were taken to the next level when their qualitative evaluation was generalized to a quantitative one, which led to the concept of *weighted automata* [29]. Such devices assign a weight to each input word, thus computing so-called *formal power series*. Weighted automata are commonly used to model numerical factors related to the input, such as costs, probabilities and consumption of resources or time, and enjoy consistent attention from the research community focused on automata theory [8, 28]. The favored algebraic structure for performing weight calculations are semirings [14, 16], as they are quite general while still being computationally efficient due to their distributivity.

Another dimension for generalizing finite-state automata lifts their input to more complex data structures such as infinite words [20, 26], trees [2], graphs [1] and pictures [11, 27]. Particularly, *finite-state tree automata* were introduced independently in [5, 31, 32]. The so-called *regular tree languages* they recognize have been studied extensively [2], and find applications in a variety of areas like natural language processing [17], picture generation [6] and compiler construction [33]. In many cases, applications require both types of generalizations, and so several models of *weighted tree automata* (WTA) and the *regular tree series* they recognize continue to be studied [9].

The price to pay for the simplicity of tree automata lies in their significant limitations. For instance, they cannot ensure that certain subtrees of input trees are equal [10], much like the classical (string) automata cannot ensure that the number of a's and b's in a word is equal. This defect was tackled with extensions proposed in [25] and [3, 12, 13] where *tree automata with constraints* can explicitly require or forbid certain subtrees to be equal. Such devices have played a crucial part in deciding the *HOM-problem*: This long-standing open question [2] asks, given a regular tree language and a tree homomorphism, whether the image is again regular. A tree homomorphism performs a transformation on trees

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 200–214, doi:10.4204/EPTCS.386.16

and can duplicate subtrees, therefore the trees in the homomorphic image might have certain identical subtrees, which calls for the constraints mentioned above. In [12], the authors first represent this homomorphic image of a regular tree language by a tree automaton with explicit constraints, and then decide algorithmically if the language it recognizes is regular despite the constraints it imposes.

The nature of the *weighted* HOM-problem, where a regular tree series and a tree homomorphism are given as input, requires an individual investigation for different semirings. Recently, the approach from [12] was adjusted to the special case of nonnegative integers [23], but so far, the question remains open for other semirings. In this paper, we reverse the strategy and impose conditions on the input in order to decide the thus restricted HOM-problem for a larger class of semirings. More precisely, we require our protagonist – the weighted tree automaton with constraints – to be unambiguous, and reduce the question of its regularity to the unweighted case from [12] for any zero-sum free (commutative) semiring. Afterwards, we phrase a condition on the input of the HOM-problem which ensures that our reduction is applicable.

This article consists of five sections including its introduction. Our main contributions can be summarized as follows:

- In Section 2 we establish notations and recall the main objects that will play a role throughout the paper, primarily the *weighted tree automata with hom-constraints (WTAh)* which are used to represent homomorphic images of regular tree series.
- In Section 3 we prove that regularity is decidable for the unambiguous devices of this type over zero-sum free semirings. We achieve this by reducing the question to the unweighted case where regularity is known to be decidable [12].
- In Section 4 we integrate this decidability result into the HOM-problem. To this end, we phrase a condition on the input of the HOM-problem which guarantees that the WTAh constructed for this instance is unambiguous. Thus, the HOM-problem with input restricted accordingly is decidable for any zero-sum free semiring.
- Finally, in Section 5 we briefly summarize our results and discuss further research that will extend the present work.

2 Preliminaries and Technical Background

We begin as usual with the necessary background for this paper.

General Notation

We denote the set $\{0,1,2,\dots\}$ of nonnegative integers by $\mathbb{N}$, and we let $[k] = \{1,\dots,k\}$ for every $k \in \mathbb{N}$. Let A and B be sets. We write $|A|$ for the cardinality of A, and A^* for the set of finite strings over A. The empty string is ε and the length of a string w is $|w|$. For a mapping $f\colon A \to B$ and $S \subseteq B$ we denote the inverse image of S under f by $f^{-1}(S)$, and we write $f^{-1}(b)$ instead of $f^{-1}(\{b\})$ for every $b \in B$.

Trees

A *ranked alphabet* is a pair (Σ, rk) that consists of a finite set Σ and a rank mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$. For every $k \geq 0$, we define $\Sigma_k = \mathrm{rk}^{-1}(k)$, and we sometimes write $\sigma^{(k)}$ to indicate that $\sigma \in \Sigma_k$. We often abbreviate (Σ, rk) by Σ leaving rk implicit. Let Z be a set disjoint with Σ. The set of Σ-*trees over* Z,

denoted $T_\Sigma(Z)$, is the smallest set T such that (i) $\Sigma_0 \cup Z \subseteq T$ and (ii) $\sigma(t_1,\dots,t_k) \in T$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$ and $t_1,\dots,t_k \in T$. We abbreviate $T_\Sigma(\emptyset)$ simply to T_Σ, and call any subset $L \subseteq T_\Sigma$ a *tree language*. Consider $t \in T_\Sigma(Z)$. The set $\mathrm{pos}(t) \subseteq \mathbb{N}^*$ of *positions of* t is defined inductively by $\mathrm{pos}(t) = \varepsilon$ for every $t \in \Sigma_0 \cup Z$, and by

$$\mathrm{pos}\big(\sigma(t_1,\dots,t_k)\big) = \{\varepsilon\} \cup \bigcup_{i\in[k]} \{ip \mid p \in \mathrm{pos}(t_i)\}$$

for all $k \in \mathbb{N}$, $\sigma \in \Sigma_k$ and $t_1,\dots,t_k \in T_\Sigma(Z)$. The set of positions of t inherits the lexicographic order $\leq_{\mathrm{lex}}$ from $\mathbb{N}^*$. The *size* $|t|$ and *height* $\mathrm{ht}(t)$ *of* t are defined as

$$|t| \;=\; |\mathrm{pos}(t)| \quad \text{and} \quad \mathrm{ht}(t) = \max_{p\in\mathrm{pos}(t)} |p| \,.$$

For $p \in \mathrm{pos}(t)$, the *label* $t(p)$ *of* t *at* p, the *subtree* $t|_p$ *of* t *at* p and the *substitution* $t[t']_p$ *of* t' *into* t *at* p are defined

- for $t \in \Sigma_0 \cup Z$ by $t(\varepsilon) = t|_\varepsilon = t$ and $t[t']_\varepsilon = t'$, and
- for $t = \sigma(t_1,\dots,t_k)$ by $t(\varepsilon) = \sigma,\ t(ip') = t_i(p'),\ t|_\varepsilon = t,\ t|_{ip'} = t_i|_{p'},\ t[t']_\varepsilon = t'$, and

  $$t[t']_{ip'} = \sigma(t_1,\dots,t_{i-1},t_i[t']_{p'},t_{i+1},\dots,t_k)$$

  for all $k \in \mathbb{N},\ \sigma \in \Sigma_k,\ t_1,\dots,t_k \in T_\Sigma(Z),\ i \in [k]$ and $p' \in \mathrm{pos}(t_i)$.

For every subset $S \subseteq \Sigma \cup Z$, we let $\mathrm{pos}_S(t) = \{p \in \mathrm{pos}(t) \mid t(p) \in S\}$ and we abbreviate $\mathrm{pos}_{\{s\}}(t)$ by $\mathrm{pos}_s(t)$ for every $s \in \Sigma \cup Z$. Let $X = \{x_1,x_2,\dots\}$ be a fixed, countable set of formal variables. For $k \in \mathbb{N}$ we denote by X_k the subset $\{x_1,\dots,x_k\}$. For any $t \in T_\Sigma(X)$ we let

$$\mathrm{var}(t) = \{x \in X \mid \mathrm{pos}_x(t) \neq \emptyset\} \,.$$

Finally, for $t \in T_\Sigma(Z)$, a subset $V \subseteq Z$ and a mapping $\theta : V \to T_\Sigma(Z)$, we define the *substitution* $t\theta$ *applied to* t by $v\theta = \theta(v)$ for $v \in V,\ z\theta = z$ for $z \in Z \setminus V$, and

$$\sigma(t_1,\dots,t_k)\theta = \sigma\big(t_1\theta,\dots,t_k\theta\big)$$

for all $k \in \mathbb{N}$, $\sigma \in \Sigma_k$ and $t_1,\dots,t_k \in T_\Sigma(Z)$. If $V = \{v_1,\dots,v_n\}$, we write the substitution θ explicitly as $[v_1 \leftarrow \theta(v_1),\dots,v_n \leftarrow \theta(v_n)]$, and abbreviate it further to $[\theta(x_1),\dots,\theta(x_n)]$ if $V = X_n$.

Semirings and Tree Series

A *(commutative) semiring* [14, 15] is a tuple $(\mathbb{S},+,\cdot,0,1)$ such that $(\mathbb{S},+,0)$ and $(\mathbb{S},\cdot,1)$ are commutative monoids, $\cdot$ distributes over $+$, and $0 \cdot s = 0$ for all $s \in \mathbb{S}$. Examples include

- the Boolean semiring $\mathbb{B} = \big(\{0,1\},\vee,\wedge,0,1\big)$,
- the semiring $\mathbb{N} = \big(\mathbb{N},+,\cdot,0,1\big)$,
- the semiring $\mathbb{Z} = \big(\mathbb{Z},+,\cdot,0,1\big)$,
- the tropical semiring $\mathbb{T} = \big(\mathbb{N}\cup\{\infty\},\min,+,\infty,0\big)$, and
- the arctic semiring $\mathbb{A} = \big(\mathbb{N}\cup\{-\infty\},\max,+,-\infty,0\big)$.

When there is no risk of confusion, we refer to a semiring $(\mathbb{S},+,\cdot,0,1)$ simply by its carrier set $\mathbb{S}$. We call $\mathbb{S}$ *zero-sum free* if $a+b=0$ implies $a=b=0$ for all $a,b \in \mathbb{S}$. All semirings listed above except for $\mathbb{Z}$ are zero-sum free. Let Σ be a ranked alphabet and Z a set. Any mapping $\varphi : T_\Sigma(Z) \to \mathbb{S}$ is called a *tree series* or *weighted tree language* over $\mathbb{S}$, and its *support* is the set $\mathrm{supp}(\varphi) = \{t \in T_\Sigma(Z) \mid \varphi(t) \neq 0\}$.

Tree Homomorphisms

Given ranked alphabets Σ and Δ, let $h' \colon \Sigma \to T_\Delta(X)$ be a mapping that satisfies $h'(\sigma) \in T_\Delta(X_k)$ for all $k \in \mathbb{N}$ and $\sigma \in \Sigma_k$. We extend h' to $h \colon T_\Sigma \to T_\Delta$ by $h(\alpha) = h'(\alpha) \in T_\Delta(X_0) = T_\Delta$ for all $\alpha \in \Sigma_0$ and

$$h(\sigma(s_1, \ldots, s_k)) = h'(\sigma)[x_1 \leftarrow h(s_1), \ldots, x_k \leftarrow h(s_k)]$$

for all $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $s_1, \ldots, s_k \in T_\Sigma$. The mapping h is called the *tree homomorphism induced by* h', and we identify h' and its induced tree homomorphism h. We call h

- *nonerasing* if $h(\sigma) \notin X$ for all $\sigma \in \Sigma$,
- *nondeleting* if $\sigma \in \Sigma_k$ implies $\mathrm{var}(h'(\sigma)) = X_k$ for all $k \in \mathbb{N}$, and
- *input-finitary* if the preimage $h^{-1}(t)$ is finite for every $t \in T_\Delta$.

If a tree homomorphism $h \colon T_\Sigma \to T_\Delta$ is nonerasing and nondeleting, then for every $s \in h^{-1}(t)$, it is $|s| \leq |t|$. In particular, h is then input finitary.

Consider a tree series $A \colon T_\Sigma \to \mathbb{S}$. Its *homomorphic image under* h is the tree series $h_A \colon T_\Delta \to \mathbb{S}$ defined for every $t \in T_\Delta$ by

$$h_A(t) = \sum_{s \in h^{-1}(t)} A(s) \,.$$

This definition relies on the tree homomorphism to be input-finitary, otherwise the above sum is not finite, so the value $h_A(t)$ might not be well-defined. For this reason, we will only consider nondeleting and nonerasing tree homomorphisms.

Weighted Tree Automata with Constraints

Recently it was shown [21, 22] that such homomorphic images of regular tree languages can be represented efficiently using *weighted tree automata with hom-constraints* (WTAh). These devices were first introduced for the unweighted case in [12] and defined for zero-sum free commutative semirings in [21].

Definition 1 (cf. [22, Definition 1])**.** *Let $\mathbb{S}$ be a commutative semiring. A* weighted tree automaton over $\mathbb{S}$ with hom-constraints (WTAh) *is a tuple $\mathscr{A} = \big(Q, \Sigma, F, R, \mathrm{wt}\big)$ such that Q is a finite set of states, Σ is a ranked alphabet, $F \subseteq Q$ is the set of final states, R is a finite set of rules of the form (ℓ, q, E) such that $\ell \in T_\Sigma(Q) \setminus Q$, $q \in Q$ and E is an equivalence relation on $\mathrm{pos}_Q(\ell)$, and $\mathrm{wt} \colon R \to \mathbb{S}$ assigns a weight to each rule.*

Rules of a WTAh are typically depicted as $r = \ell \xrightarrow{E}_{\mathrm{wt}(r)} q$. The components of such a rule are the *left-hand side* ℓ, the *target state* q, the set E of *hom-constraints* and the *weight* $\mathrm{wt}(r)$. A hom-constraint $(p, p') \in E$ is listed as "$p = p'$", and if p and p' are distinct, then p, p' are called *constrained positions*. The equivalence class of p in E is denoted $[p]_{\equiv_E}$. We typically omit the trivial constraints $(p, p) \in E$.

Example 2. *Let Σ be the ranked alphabet $\{a^{(0)}, g^{(1)}, k^{(2)}\}$. Consider the WTAh $\mathscr{A} = \big(Q, \Sigma, F, R, \mathrm{wt}\big)$ over $\mathbb{Z}$ with $Q = \{q, q_f\}$, $F = \{q_f\}$ and the set of rules and weights*

$$R \;=\; \big\{\, a \to_1 q, \quad g(q) \to_2 q, \quad k\big(q, g(q)\big) \xrightarrow{1=21}_1 q_f \,\big\}.$$

The only constrained positions are 1 *and* 21 *in the rule with left-hand side $k\big(q, g(q)\big)$.*

The WTAh is a *weighted tree grammar* (WTG) if E is the identity relation for every rule $\ell \xrightarrow{E} q$, and a WTA in the classical sense [2] if additionally $\mathrm{pos}_\Sigma(\ell) = \{\varepsilon\}$. WTG and WTA are equally expressive, as WTG can be translated straightforwardly into WTA using additional states.

In this work, we are particularly interested in a specific subclass of WTAh, namely the *eq-restricted* WTAh [22]. In such a device, there is a designated *sink-state* whose sole purpose is to neutrally process copies of identical subtrees. More precisely, whenever different subtrees are mutually constrained, there is one leading copy among them that can be processed with arbitrary states and weights, while every other copy is handled exclusively by the weight-neutral sink-state.

Definition 3. *A WTAh* $\big(Q,\Sigma,F,R,\mathrm{wt}\big)$ *is* eq-restricted *if it has a* sink state $\bot \in Q \setminus F$ *such that*

- *for all* $\sigma \in \Sigma$*, the rule* $\sigma(\bot,\dots,\bot) \to_1 \bot$ *belongs to* R*, and no other rule targets* $\bot$*, and*
- *for every rule* $\ell \xrightarrow{E} q$ *with* $q \neq \bot$*, the following conditions hold:*
  Let $\mathrm{pos}_Q(\ell) = \{p_1,\dots,p_n\}$ *and* $q_i = \ell(p_i)$ *for* $i \in [n]$*.*
  1. *For each* $i \in [n]$*, there exists* $q' \in Q \setminus \{\bot\}$ *with* $\{q_j \mid p_j \in [p_i]_{\equiv_E}\} \setminus \{\bot\} = \{q'\}$*.*
  2. *There exists exactly one* $p_j \in [p_i]_{\equiv_E}$ *such that* $q_j = q'$*.*

In other words, among each E-equivalence class of positions of a left-hand side ℓ, there is only one occurrence of a state different from $\bot$, every other related position is labelled by $\bot$. Moreover, $\bot$ processes every possible tree with weight 1. Whenever we consider an eq-restricted WTAh, we denote its state set by $Q \dot\cup \{\bot\}$ instead of $Q \ni \bot$ to point out the sink-state.

Example 4. *Recall the WTAc* $\mathscr{A}$ *from Example 2. It is not eq-restricted since the constrained positions* 1 *and* 21 *are both labeled by the same state, which is not a sink state. Instead, let us add a non-final state* $\bot$ *to* Q*, replace the rule* $k\big(q,g(q)\big) \xrightarrow{1=21}_1 q_f$ *with* $k\big(q,g(\bot)\big) \xrightarrow{1=21}_1 q_f$ *and add the required rules targeting* $\bot$ *to obtain an eq-restricted WTAh* $\mathscr{A}'$*. More precisely, we have the eq-restricted WTAh* $\mathscr{A}' = \big(\{q,q_f,\bot\},\Delta,\{q_f\},R',\mathrm{wt}'\big)$ *with the set of rules and weights*

$$\begin{aligned} R' = \big\{\, a \to_1 q, \quad g(q) \to_2 q, \quad k\big(q,g(\bot)\big) \xrightarrow{1=21}_1 q_f \,\big\} \\ \cup \big\{\, a \to_1 \bot, \quad g(\bot) \to_1 \bot, \quad k(\bot,\bot) \to_1 \bot \,\big\}\,. \end{aligned}$$

Next, let us recall the semantics of WTAh from [22, Definitions 2 and 3].

Definition 5. *Let* $\mathscr{A} = \big(Q,\Sigma,F,R,\mathrm{wt}\big)$ *be a WTAh. A* run *of* $\mathscr{A}$ *is a tree over the ranked alphabet* $\Sigma \cup R$ *where the rank of a rule is* $\mathrm{rk}(\ell \xrightarrow{E} q) = |\mathrm{pos}_Q(\ell)|$*, and it is defined inductively. Consider* $t_1,\dots,t_n \in T_\Sigma$*,* $q_1,\dots,q_n \in Q$ *and suppose that* ρ_i *is a run of* $\mathscr{A}$ *for* t_i *to* q_i *with weight* $\mathrm{wt}(\rho_i) = a_i$ *for each* $i \in [n]$*. Assume that there is a rule of the form* $\ell \xrightarrow{E}_a q$ *in* R *such that* $\ell = \sigma(\ell_1,\dots,\ell_m)$*,* $\mathrm{pos}_Q(\ell) = \{p_1,\dots,p_n\}$ *with* $\ell(p_i) = q_i$ *and that for all* $p_i = p_j \in E$*, it is* $t_i = t_j$*. Then the following is a run of* $\mathscr{A}$ *for the tree* $t = \ell[t_1]_{p_1} \cdots [t_n]_{p_n}$ *to* q*:*

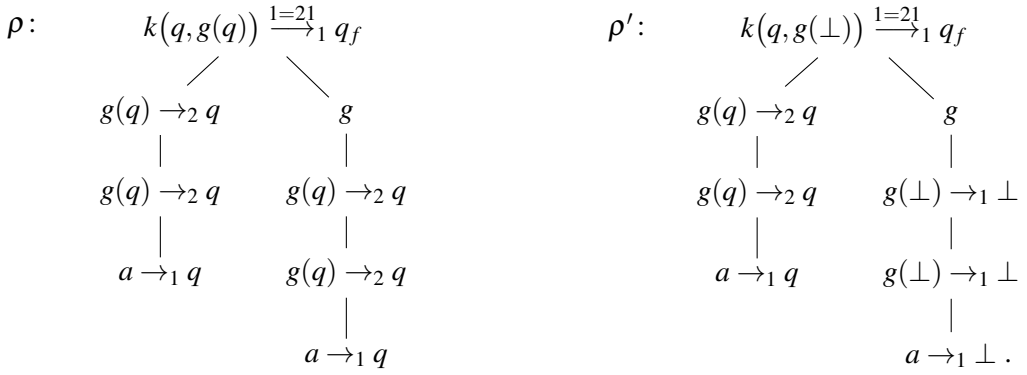
$$\rho = \big(\ell \xrightarrow{E}_a q\big)(\ell_1,\dots,\ell_m)[\rho_1]_{p_1} \cdots [\rho_n]_{p_n}\,.$$

Its weight $\mathrm{wt}(\rho)$ *is computed as* $a \cdot \prod_{i\in[n]} a_i$*. If* $\mathrm{wt}(\rho) \neq 0$*, then* ρ *is* valid*, and if in addition,* $q \in F$ *for its* target state q*, then* ρ *is* accepting*. We call* $\mathscr{A}$ unambiguous *if for every* $t \in T_\Sigma$ *there is at most one accepting run. The value* $\mathrm{wt}^q(t)$ *is the sum of all weights* $\mathrm{wt}(\rho)$ *of runs of* $\mathscr{A}$ *for* t *to* q*. Finally, the tree series recognized by* $\mathscr{A}$ *is defined simply by*

$$[\![\mathscr{A}]\!]\colon\ T_\Sigma \to \mathbb{S}, \quad t \mapsto \sum_{q\in F} \mathrm{wt}^q(t)\,.$$

Since the weights of rules are multiplied, we can assume wlog. that $\mathrm{wt}(r) \neq 0$ for all $r \in R$, which we will do from now on. Finally, two WTAh are said to be *equivalent* if they recognize the same tree series.

Example 6. *Recall the WTAh $\mathscr{A}$ and $\mathscr{A}'$ from Examples 2 and 4 and consider the tree $k\big(g^2(a), g^3(a)\big)$. The accepting runs ρ and ρ' of $\mathscr{A}$ and $\mathscr{A}'$, respectively, for it are the following:*



It is $\mathrm{wt}(\rho) = 2^4$ *while* $\mathrm{wt}'(\rho') = 2^2$ *because in the eq-restricted WTAh $\mathscr{A}'$, every constrained subtree except for one (pending from position* 1*) is processed exclusively in the state $\bot$ with weight* 1*.*

Both WTAh are unambiguous, so it is impossible for different accepting runs with complementary weights to cancel out. Thus for a tree $t \in T_\Sigma$ it is $t \in \mathrm{supp}[\![\mathscr{A}]\!]$ iff. $\mathscr{A}$ has an accepting run for t, and the same is true for $\mathscr{A}'$. In fact, it is

$$\mathrm{supp}[\![\mathscr{A}]\!] \;=\; \mathrm{supp}[\![\mathscr{A}']\!] \;=\; \big\{k\big(g^n a, g^{n+1}(a)\big) \mid n \in \mathbb{N}\big\}\,.$$

If a tree series is recognized by a WTA, it is called *regular*, if it is recognized by some WTAh, then it is called *constraint-regular*, and if it is recognized by an eq-restricted WTAh, then it is called *hom-regular*. This choice of name hints at the fact that eq-restricted WTAh are tailored to represent homomorphic images of regular tree series. For an illustration of this feature, consider the following example.

Example 7. *Let $\Sigma = \{a^{(0)}, g^{(1)}, f^{(1)}\}$ and $A\colon T_\Sigma \to \mathbb{N}$ defined for every $s \in T_\Sigma$ by*

$$A(s) = \begin{cases} 2^n & \text{if } s = f\big(g^n(a)\big) \\ 0 & \text{else.} \end{cases}$$

A simple WTA recognizing the tree series A is $\mathscr{A} = \big(\{q, q_f\}, \Sigma, \{q_f\}, R, \mathrm{wt}\big)$ with the rules and weights $R = \big\{a \to_1 q,\ g(q) \to_2 q,\ f(q) \to_1 q_f\big\}$. Consider $\Delta = \{a^{(0)}, g^{(1)}, k^{(2)}\}$ and the input-finitary tree homomorphism $h\colon T_\Sigma \to T_\Delta$ induced by the mapping $h(a) = a$, $h(g) = g(x_1)$ and $h(f) = k\big(x_1, g(x_1)\big)$. The homomorphic image h_A is the tree series given for all $t \in T_\Delta$ by

$$h_A(t) = \begin{cases} 2^n & \text{if } t = k\big(g^n(a), g^{n+1}(a)\big) \\ 0 & \text{else.} \end{cases}$$

The natural eq-restricted WTAh that recognizes h_A is $\mathscr{A}' = \big(\{q, q_f, \bot\}, \Delta, \{q_f\}, R', \mathrm{wt}'\big)$ from Example 4 with

$$\begin{aligned} R' = \big\{ & a \to_1 q, \quad g(q) \to_2 q, \quad k\big(q, g(\bot)\big) \xrightarrow{1=21}_1 q_f \big\} \\ \cup \big\{ & a \to_1 \bot, \quad g(\bot) \to_1 \bot, \quad k(\bot, \bot) \to_1 \bot \big\}\,. \end{aligned}$$

The new rules in R' are obtained from the rules in R by applying the tree homomorphism to their left-hand sides. The duplicated subtree below k targets the sink state $\bot$ instead of q to avoid distorting the weight with an additional factor 2^n.

More formally, the following statement was shown in [22]. We include a condensed version of the proof as we will refer to a technical detail below.

Lemma 8 (cf. [22, Theorem 5])**.** *Let $\mathbb{S}$ be a commutative semiring, $\mathscr{A} = \big(Q, \Sigma, F, R, \mathrm{wt}\big)$ a WTA over $\mathbb{S}$ and $h\colon T_\Sigma \to T_\Delta$ a nondeleting and nonerasing tree homomorphism. There is an eq-restricted WTAh $\mathscr{A}'$ that recognizes $h_{[\![\mathscr{A}]\!]}$.*

Proof. An eq-restricted WTAh $\mathscr{A}'$ for $h_{[\![\mathscr{A}]\!]}$ is constructed in two stages.

First, we define $\mathscr{A}'' = \big(Q \dot\cup \{\bot\}, \Delta \cup \Delta \times R, F'', R'', \mathrm{wt}''\big)$ such that for every $r = \sigma(q_1, \dots, q_k) \to_{\mathrm{wt}(r)} q$ in R and $h(\sigma) = u = \delta(u_1, \dots, u_n)$, we include

$$r'' \;=\; \Big(\langle \delta, r\rangle(u_1, \dots, u_n)[\![q_1, \dots, q_k]\!] \xrightarrow{E}_{\mathrm{wt}''(r'')} q\Big) \;\in\; R'' \qquad \text{with} \qquad E \;=\; \bigcup_{i \in [k]} \mathrm{pos}_{x_i}(u)^2$$

where the substitution $\langle \delta, r\rangle(u_1, \dots, u_n)[\![q_1, \dots, q_k]\!]$ replaces for every $i \in [k]$ only the $\leq_{\mathrm{lex}}$-minimal occurrence of x_i in $\langle \delta, r\rangle(u_1, \dots, u_n)$ by q_i and all other occurrences by $\bot$. We set $\mathrm{wt}''(r'') = \mathrm{wt}(r)$. Additionally, we let $r''_\delta = \delta(\bot, \dots, \bot) \to \bot \in R''$ with $\mathrm{wt}''(r''_\delta) = 1$ for every $k \in \mathbb{N}$ and $\delta \in \Delta_k$. No other productions are in R''. Finally, we let $F'' = F$.

We can now delete the annotation: We use a deterministic relabeling to remove the second components of labels of $\Delta \times R$, adding up the weights of now identical rules. Since hom-regular languages are closed under relabelings [22, Theorem 4], we obtain an eq-restricted WTAh $\mathscr{A}' = \big(Q \dot\cup \{\bot\}, \Delta, F', R', \mathrm{wt}'\big)$ recognizing $h_{[\![\mathscr{A}]\!]}$. □

The WTAh constructed for the homomorphic image of a WTA preserves the original state behaviour in its leading copies of duplicated subtrees. Using the notation from the proof of Lemma 8, we want to define a mapping that traces the runs of the input WTA to its homomorphic image.

Definition 9. *Let $\mathscr{A}, h$ and $\mathscr{A}'$ be as in Lemma 8, let $r = \sigma(q_1, \dots, q_k) \to q \in R$ and $h(\sigma) = \delta(u_1, \dots, u_n)$. We let $h^R(r)$ be the rule $\delta(u_1, \dots, u_n)[\![q_1, \dots, q_k]\!] \xrightarrow{E} q$ of the WTAh $\mathscr{A}'$.*

The assignment h^R extends naturally to the runs of $\mathscr{A}$: For a run of the form $\rho = r = (\alpha \to q)$ with $\alpha \in \Sigma^0$, we set $h^R(\rho) = h^R(r)$. For a run of $\mathscr{A}$ of the form $\rho = r(\rho_1, \dots, \rho_k)$ with $r = \sigma(q_1, \dots, q_k) \to q$ and $h(\sigma) = \delta(u_1, \dots, u_n)$ we set

$$h^R(\rho) = \big(h^R(r)\big)(u_1, \dots, u_n)[\![h^R(\rho_1), \dots, h^R(\rho_k)]\!]\,;$$

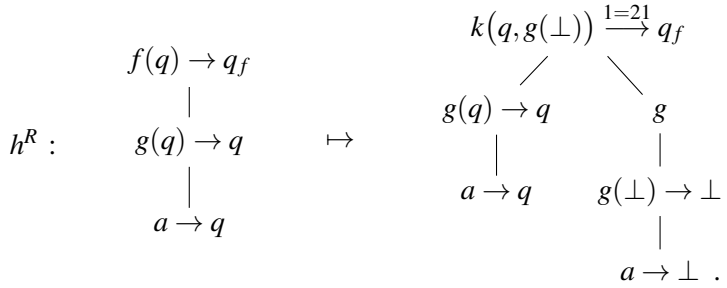
here, the substitution $[\![h^R(\rho_1), \dots, h^R(\rho_k)]\!]$ replaces for every $i \in [k]$ only the $\leq_{lex}$-minimal occurrence of x_i in $\big(h^R(r)\big)(u_1, \dots, u_n)$ by $h^R(\rho_i)$ and all other occurrences by the respective unique run to $\bot$ for the unique tree that satisfies the constraint E.

Using the notation above, the assignment $h^R\colon R \to R'$ is well-defined, but not necessarily injective, and its image is $h^R(R) = \{r' \in R' \mid r' \text{ targets some } q \neq \bot\}$. Let us see how it acts on our running example.

Example 10. *Recall the WTA $\mathscr{A}$ and WTAh $\mathscr{A}'$ from Example 7. The mapping h^R assigns*

$$h^R\colon \quad f(q) \to q_f \quad \mapsto \quad k\big(q, g(\bot)\big) \xrightarrow{1=21} q_f\,,$$

and for the unique run of $\mathscr{A}$ for the tree $f\big(g(a)\big)$, it is



When discussing the behaviour of a WTAh $\mathscr{A}$, we often argue with the help of runs ρ, so it is a nuisance that we might have $\mathrm{wt}(\rho) = 0$. This anomaly can occur even if $\mathrm{wt}(r) \neq 0$ for all rules r of $\mathscr{A}$ due to the presence of zero-divizors, that is, elements $s, s' \in \mathbb{S} \setminus \{0\}$ such that $s \cdot s' = 0$. Fortunately, we can avoid this altogether using a construction of [18], which is based on DICKSON's Lemma [4]. It was first lifted to tree automata in [7] and later to WTAh in [21, 22]. Here, we slightly adjust the proof of Lemma 3 in [22] such that it preserves the eq-restriction of the input WTAh.

Lemma 11. *(cf. [22, Lemma 3]) Let $\mathbb{S}$ be a commutative semiring. For every eq-restricted WTAh $\mathscr{A}$ over $\mathbb{S}$ there exists an eq-restricted WTAh $\mathscr{A}'$ equivalent to $\mathscr{A}$ such that $\mathrm{wt}_{\mathscr{A}'}(\rho') \neq 0$ for all runs ρ' of $\mathscr{A}'$. For each $t \in \mathrm{supp}[\![\mathscr{A}]\!]$, the accepting (i.e. of non-zero weight and targeting a final state) runs of $\mathscr{A}$ for t translate bijectively into the accepting runs ρ' of $\mathscr{A}'$ for t, and the weights are preserved.*

Proof. Let $\mathscr{A}$ be the eq-restricted WTAh $\big(Q \dot\cup \{\bot\}, \Sigma, F, R, \mathrm{wt}\big)$. Obviously, $(\mathbb{S}, \cdot, 1, 0)$ is a commutative monoid with zero. Let $(s_1, \ldots, s_n)$ be an enumeration of the finite set $\mathrm{wt}(R) \setminus \{1\} \subseteq \mathbb{S}$. We consider the monoid homomorphism $h\colon \mathbb{N}^n \to \mathbb{S}$, which is given for every $m_1, \ldots, m_n \in \mathbb{N}$ by

$$h(m_1, \ldots, m_n) = \prod_{i=1}^{n} s_i^{m_i} .$$

According to DICKSON's lemma [4], the set $\min\big(h^{-1}(0)\big)$ is finite, where the partial order is the standard pointwise order on $\mathbb{N}^n$. Hence there is $u \in \mathbb{N}$ such that $\min\big(h^{-1}(0)\big) \subseteq \{0, \ldots, u\}^n = U$. We define the operation $\oplus\colon U^2 \to U$ by $(v \oplus v')_i = \min(v_i + v'_i, u)$ for every $v, v' \in U$ and $i \in [n]$. Moreover, for every $i \in [n]$ we let $1_{s_i} \in U$ be the vector such that $(1_{s_i})_i = 1$ and $(1_{s_i})_a = 0$ for all $a \in [n] \setminus \{i\}$. Let $V = U \setminus h^{-1}(0)$. We construct the equivalent eq-restricted WTAh $\mathscr{A}' = \big(Q' \dot\cup \{\bot\}, \Sigma, F', R', \mathrm{wt}'\big)$ such that $Q' = Q \times V$, $F' = F \times V$, and R' and wt' are given as follows. Consider a rule $r = \ell \xrightarrow{E} q \in R$, let $\mathrm{pos}_Q(\ell) = \{p_1, \ldots, p_k\}$ ordered lexicographically and let $q_i = \ell(p_i)$ for all $i \in [k]$. Note that we do not consider the leaves of ℓ that are labeled by $\bot$. For all choices of $v_1, \ldots, v_k \in V$ such that the value $v = 1_{\mathrm{wt}(r)} \oplus \bigoplus_{i=1}^{k} v_i$ is again in V, the production

$$\ell[\langle q_1, v_1\rangle]_{p_1} \ldots [\langle q_k, v_k\rangle]_{p_k} \xrightarrow{E} \langle q, v\rangle$$

belongs to R' and its weight is $\mathrm{wt}'(p') = \mathrm{wt}(r)$. No further rules are in R'.

By annotating the power vectors v_i to the states $q \neq \bot$, we suitably (for the purpose of zero-divisors) track the weight of runs as v. If attaching another rule adopted from R to so far valid runs of $\mathscr{A}'$ would evaluate the overall weight to zero, then we exclude this rule from R'. Consequently, every run of $\mathscr{A}'$ is valid. To preserve the eq-restriction, we only annotate power vectors v_i to the non-sink states. It is safe to omit $\bot$ in this construction since $\bot$ only ever processes the neutral weight 1. □

From here on, we silently assume that each WTAh avoids zero-divizors.

A main result proved in this article is deciding regularity for unambiguous WTAh over zero-sum free commutative semirings. We achieve this by reducing the problem to the unweigted (i.e. boolean) case solved in [12]. For this, we must relate our WTAh model to the *tree automata with HOM equality constraints* used by [12] which differ slightly from our WTAh over the boolean semiring. Fortunately, the two are closely related and the translation is rather simple: We merely eliminate the sink state and drop the weight assignment.

Lemma 12. *Let $\mathbb{S}$ be a commutative semiring and $\mathscr{A} = \big(Q\dot\cup\{\bot\},\Sigma,F,R,\mathrm{wt}\big)$ an eq-restricted WTAh over $\mathbb{S}$. If $\mathscr{A}$ is unambiguous or $\mathbb{S}$ is zero-sum free, then there is a* tree automaton with HOM equality constraints (TA_{hom}) *[12] $\mathscr{A}^{\mathbb{B}}$ that recognizes the tree language* $\mathrm{supp}[\![\mathscr{A}]\!]$. *If $\mathscr{A}$ is a WTA (i.e. without constraints), then $\mathscr{A}^{\mathbb{B}}$ is also a TA without constraints.*

Proof. Let $q \in Q$ and consider a rule $\ell \xrightarrow{E}_a q$ of $\mathscr{A}$. Suppose that $\{p_1^1,\dots,p_{n_1}^1\},\ \dots\ ,\{p_1^m,\dots,p_{n_m}^m\}$ are the equivalence classes of E, and wlog. let p_1^i be the unique representative such that $\ell(p_1^i) \neq \bot$ for each $i \in [m]$. Then we include the unweighted rule

$$\ell\,[\ell(p_1^1)]_{p_2^1}\cdots[\ell(p_1^1)]_{p_{n_1}^1}\ \cdots\ [\ell(p_1^m)]_{p_2^m}\cdots[\ell(p_1^m)]_{p_{n_m}^m} \xrightarrow{E} q$$

in $R^{\mathbb{B}}$, that is, we replace every occurrence of $\bot$ by the unique state from Q that labels a related position. This is necessary because the definition of TA_{hom} requires E-related positions to be labelled with the same state. We proceed this way for every rule of $\mathscr{A}$, discarding the rules that target $\bot$, and obtain the (unweighted) TA_{hom} $\mathscr{A}^{\mathbb{B}} = \big(Q,\Sigma,F,R^{\mathbb{B}}\big)$. Since $\mathscr{A}$ avoids zero-divizors, the conditions in the statement are each sufficient to ensure that $t \in \mathrm{supp}[\![\mathscr{A}]\!]$ iff. there exists an run of $\mathscr{A}$ for t to a final state, so $\mathscr{A}^{\mathbb{B}}$ recognizes $\mathrm{supp}[\![\mathscr{A}]\!]$. □

Example 13. *Reconsider the WTAh $\mathscr{A}'$ from Example 7. To obtain the TA_{hom} $(\mathscr{A}')^{\mathbb{B}}$, we remove the sink state $\bot$, all rules that target $\bot$ and the weight assignment, and replace the rule $k\big(q,g(\bot)\big) \xrightarrow{1=21}_1 q_f$ with the unweighted rule $k\big(q,g(q)\big) \xrightarrow{1=21} q_f$.*

3 Deciding Regularity for Unambiguous WTAh

In this section, we prove that regularity is decidable for unambiguous eq-restricted WTAh over zero-sum free semirings. To this end, we reduce this problem to regularity in the unweighted case, which was proved decidable in [12].

We begin by defining the *linearization* of eq-restricted WTAh, which was introduced for the boolean case in [12] and adapted to the weighted model in [23]. The linearization of a WTAh $\mathscr{A}$ by the number $\mathtt{h}$ is a WTG $\mathtt{lin}(\mathscr{A},\mathtt{h})$ that approximates $\mathscr{A}$: It simulates all runs of $\mathscr{A}$ which only enforce the equality of subtrees of height at most $\mathtt{h}$. This is achieved by instantiating the constrained Q-positions of every rule $\ell \xrightarrow{E} q$ in $\mathscr{A}$ with compatible trees of height at most $\mathtt{h}$, while the Q-positions of ℓ that are unconstrained by E remain unchanged.

Formally, the linearization is defined following [12, Definition 7.1].

Definition 14 (cf. [23, Definition 12])**.** *Let $\mathbb{S}$ be a commutative semiring. Consider an eq-restricted WTAh $\mathscr{A} = (Q\dot\cup\{\bot\},\Sigma,F,R,\mathrm{wt})$ over $\mathbb{S}$, and let $\mathtt{h} \in \mathbb{N}$ be a nonnegative integer. The* linearization of $\mathscr{A}$ by $\mathtt{h}$ *is the WTG $\mathtt{lin}(\mathscr{A}',\mathtt{h}) = \big(Q,\Sigma,F,R_{\mathtt{h}},\mathrm{wt}_{\mathtt{h}}\big)$, where $R_{\mathtt{h}}$ and $\mathrm{wt}_{\mathtt{h}}$ are defined as follows.*

For $\ell' \in T_\Sigma(Q \dot\cup \{\bot\})$ and $q \in Q$, we include the rule $(\ell' \to q)$ in $R_\mathtt{h}$ iff. there exist a rule $(\ell \xrightarrow{E} q) \in R$, positions $p_1, \dots, p_k \in \mathrm{pos}_{Q\dot\cup\{\bot\}}(\ell)$, and trees $t_1, \dots, t_k \in T_\Sigma$ such that

- $\{p_1, \dots, p_k\} = \bigcup_{p \in \mathrm{pos}_\bot(\ell)} [p]_E$, *that is, E constrains exactly the positions $p_1, \dots, p_k$,*
- $(p_i, p_j) \in E$ *implies* $t_i = t_j$ *for all* $i, j \in [k]$,
- $\ell' = \ell[t_1]_{p_1} \cdots [t_k]_{p_k}$, *and*
- $\mathrm{wt}^{\ell(p_i)}(t_i) \neq 0$ *and* $\mathrm{ht}(t_i) \leq \mathtt{h}$ *for all* $i \in [k]$.

For every such production $\ell' \to q$ we set $\mathrm{wt}_\mathtt{h}(\ell' \to q)$ as the sum over all weights

$$\mathrm{wt}(\ell \xrightarrow{E} q) \cdot \prod_{i \in [k]} \mathrm{wt}^{\ell(p_i)}(t_i)$$

for all $(\ell \xrightarrow{E} q) \in R$, $p_1, \dots, p_k \in \mathrm{pos}_{Q\dot\cup\{\bot\}}(\ell)$ and $t_1, \dots, t_k \in T_\Sigma$ as above.

Note that the linearization is a WTG without constraints, so it recognizes a regular tree series. Let us apply this construction to our running example.

Example 15. *We recall the WTAh $\mathscr{A}'$ from Example 7 and set $\mathtt{h} = 2$. The linearization of $\mathscr{A}'$ by 2 instantiates every constrained position by compatible trees of maximal height 2, keeping track of the weights, and removes $\bot$ and the rules that target it. More precisely, $\mathtt{lin}(\mathscr{A}', 2) = \big(\{q, q_f\}, \Delta, \{q_f\}, R_2, \mathrm{wt}_2\big)$ with the set of rules and weights*

$$\begin{aligned} R_2 \;=\; \big\{\, & a \to_1 q, \qquad g(q) \to_2 q, \qquad k\big(a, g(a)\big) \to_1 q_f, \\ & k\big(g(a), g(g(a))\big) \to_2 q_f, \qquad k\big(g(g(a)), g(g(g(a)))\big) \to_4 q_f \,\big\}. \end{aligned}$$

This example illustrates that the larger we choose $\mathtt{h}$, the better $\mathtt{lin}(\mathscr{A}', \mathtt{h})$ approximates $[\![\mathscr{A}']\!]$. In this particular case however, there will always be a tree t such that $[\![\mathscr{A}']\!](t) \neq [\![\mathtt{lin}(\mathscr{A}', \mathtt{h})]\!](t)$, say, the tree $k\big(g^{\mathtt{h}+1}(a), g^{\mathtt{h}+2}(a)\big)$. For eq-restricted WTAh $\mathscr{A}$ over $\mathbb{B}$ or $\mathbb{N}$ it is known [12, 22] that $[\![\mathscr{A}]\!]$ is regular iff. $[\![\mathtt{lin}(\mathscr{A}, \mathtt{h})]\!] = [\![\mathscr{A}]\!]$ for a certain parameter $\mathtt{h}$. For other semirings, a customized investigation is necessary, but unambiguous WTAh allow us to decide regularity by applying the boolean case directly. To this end, the following lemma is fundamental.

Lemma 16. *Let $\mathbb{S}$ be a commutative semiring, $\mathscr{A}$ an eq-restricted WTAh over $\mathbb{S}$ and $h \in \mathbb{N}$. For each $t \in \mathrm{supp}[\![\mathscr{A}]\!]$, there are at most as many accepting runs of $\mathtt{lin}(\mathscr{A}, \mathtt{h})$ for t as there are accepting runs of $\mathscr{A}$ for t. In particular, if $\mathscr{A}$ is unambiguous, then so is its linearization, and for every $t \in \mathrm{supp}[\![\mathscr{A}]\!]$ it is either $[\![\mathtt{lin}(\mathscr{A}, \mathtt{h})]\!](t) = [\![\mathscr{A}]\!](t)$, or there are no accepting runs of $\mathtt{lin}(\mathscr{A}, \mathtt{h})$ for t.*

Proof. The linearization $\mathtt{lin}(\mathscr{A}, \mathtt{h})$ is defined in such a way that it simulates every run ρ of $\mathscr{A}$ with the following property: Say ρ processes $t \in T_\Sigma$, then for every rule $\ell \xrightarrow{E} q$ used in ρ at position p (that is, at the root of $t|_p$), and for every position $\bar p$ constrained by E, it is $\mathrm{ht}(t|_{p\bar p}) \leq \mathtt{h}$. Different runs of $\mathscr{A}$ might be merged into the same run of $\mathtt{lin}(\mathscr{A}, \mathtt{h})$, but for a particular run of $\mathscr{A}$ it is uniquely determined which run of $\mathtt{lin}(\mathscr{A}, \mathtt{h})$ will incorporate it. □

We need yet another technical ingredient for the reduction to the unweighted case: to interchange the linearization of a WTAh and its projection onto the boolean TA_{hom}. The linearization for TA_{hom} was defined in [12, Definition 7] and indeed, the following holds.

Lemma 17. *Consider an unambiguous, eq-restricted WTAh $\mathscr{A}$ over a commutative semiring. Let $\mathscr{A}^\mathbb{B}$ the TA_{hom} for $\mathrm{supp}[\![\mathscr{A}]\!]$ defined in Lemma 12 and $\mathrm{linearize}(\mathscr{A}^\mathbb{B}, \mathtt{h})$ in turn the linearization of $\mathscr{A}^\mathbb{B}$ by $\mathtt{h}$ as introduced in [12, Definition 7.1]. Then it is $\mathtt{lin}(\mathscr{A}, \mathtt{h})^\mathbb{B} = \mathrm{linearize}(\mathscr{A}^\mathbb{B}, \mathtt{h})$.*

We are now ready for the main result of this section: the reduction of regularity for eq-restricted WTAh over zero-sum free semirings to the unweighted case.

Theorem 18. *Let $\mathbb{S}$ be a zero-sum free commutative semiring and $\mathscr{A}$ an unambiguous eq-restricted WTAh over $\mathbb{S}$. The tree series $[\![\mathscr{A}]\!]$ is regular iff.* $\operatorname{supp}[\![\mathscr{A}]\!]$ *is a regular tree language.*

Proof. Suppose first that $[\![\mathscr{A}]\!]$ is regular, thus there is a WTA $\mathscr{B}$ equivalent to $\mathscr{A}$. Since $\mathbb{S}$ is zero-sum free, we can apply Lemma 12 to $\mathscr{B}$ and obtain that $\operatorname{supp}[\![\mathscr{B}]\!] = \operatorname{supp}[\![\mathscr{A}]\!]$ is regular.

Next, suppose that $[\![\mathscr{A}]\!]$ is not regular. In particular, the regular WTG $\mathtt{lin}(\mathscr{A},\mathtt{h})$ is not equivalent to $\mathscr{A}$ for any $h \in \mathbb{N}$. Thus by Lemma 16, it is $\operatorname{supp}[\![\mathscr{A}]\!] \neq \operatorname{supp}[\![\mathtt{lin}(\mathscr{A},\mathtt{h})]\!]$. By Lemma 12, $\mathtt{lin}(\mathscr{A},\mathtt{h})^{\mathbb{B}}$ recognizes the regular language $\operatorname{supp}[\![\mathtt{lin}(\mathscr{A},\mathtt{h})]\!]$, and together with Lemma 17, it is

$$[\![\mathscr{A}^{\mathbb{B}}]\!] = \operatorname{supp}[\![\mathscr{A}]\!] \neq [\![\mathtt{lin}(\mathscr{A},\mathtt{h})^{\mathbb{B}}]\!] = [\![\mathit{linearize}(\mathscr{A}^{\mathbb{B}},\mathtt{h})]\!],$$

that is, the boolean linearization of the TA_{hom} $\mathscr{A}^{\mathbb{B}}$ is not equivalent to it for any $h \in \mathbb{N}$. This, however, implies that $[\![\mathscr{A}^{\mathbb{B}}]\!] = \operatorname{supp}[\![\mathscr{A}]\!]$ is not regular as proved in [22, Lemma 7.3]. □

Note that we only used zero-sum freeness of the semiring for the first part of the statement, as Lemma 12 holds for unambiguous WTAh over arbitrary commutative semirings. With this result, regularity of eq-restricted WTAh is decidable.

Corollary 19. *Let $\mathbb{S}$ be a zero-sum free commutative semiring. Given an unambiguous eq-restricted WTAh $\mathscr{A}$ over $\mathbb{S}$ as input, it is decidable whether $[\![\mathscr{A}]\!]$ is regular.*

Proof. By Theorem 18, $[\![\mathscr{A}]\!]$ is regular iff. $\operatorname{supp}[\![\mathscr{A}]\!]$ is regular. A TA_{hom} recognizing the latter can be constructed with Lemma 12, for which, in turn, regularity is decidable [12, Section 7]. □

4 A Sufficient Condition and the HOM-Problem

So far, the assumption we make for deciding regularity is imposed on the WTAh. Meanwhile the HOM-problem has a WTA $\mathscr{A}$ and a tree homomorphism h as input. In this section, we propose conditions on $\mathscr{A}$ and h which ensure that the strategy of the previous section is applicable to the corresponding instance of the HOM-problem. We begin with a condition for h which generalizes injectivity.

Definition 20. *Let Σ and Δ be ranked alphabets and $h\colon T_\Sigma \to T_\Delta$ a nondeleting and nonerasing tree homomorphism. We call h* tetris-free *if for all $s,s' \in T_\Sigma$ with $h(s) = h(s')$, it is $\operatorname{pos}(s) = \operatorname{pos}(s')$ and for all $p \in \operatorname{pos}(s)$, we have $h\big(s(p)\big) = h\big(s'(p)\big)$.*

In other words, $h\colon T_\Sigma \to T_\Delta$ is tetris-free if we cannot combine the building blocks $h(\sigma)$, $\sigma \in \Sigma$ in different ways to build the same tree. In contrast, Figure 1 below shows the well-known *Tetriminos*®© [19] violating (and thus naming) the tetris-free condition.

Let us discuss a short example and counter-example.

Example 21. *Let $\Sigma = \{a^{(0)}, b^{(0)}, g^{(1)}\}$ and $\Delta = \{c^{(0)}, k^{(2)}\}$. Consider the tree homomorphism $h\colon T_\Sigma \to T_\Delta$ induced by $h(a) = h(b) = c$ and $h(g) = k(x_1, x_1)$. While h is not injective, it is tetris-free. However, the tree homomorphism $\hat{h}\colon T_\Sigma \to T_\Delta$ induced by $\hat{h}(a) = c$, $\hat{h}(b) = k(c,c)$ and $\hat{h}(g) = k(x_1, c)$ is not: The trees $g(a)$ and b violate the tetris-free condition.*

Intuitively, if a tree homomorphism h is tetris-free, then any non-injective behaviour of h is located entirely at the symbol level. This allows the construction of the WTAh to cancel the non-injectivity of h. For this, however, we also need to make an assumption on the input WTA $\mathscr{A}$, which leads us to this augmented version of unambiguity for $\mathscr{A}$.

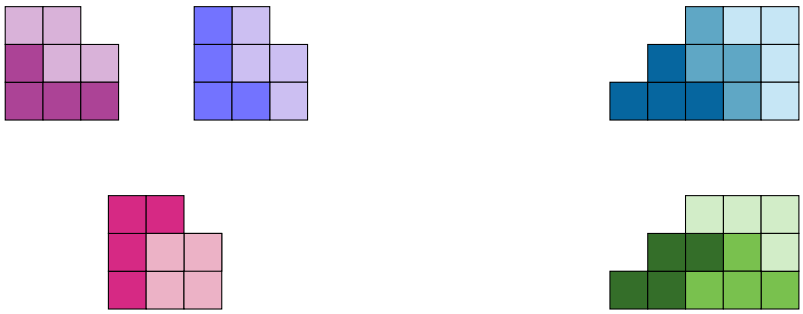


Figure 1: The game of *Tetris*®© [19] being non-tetris-free by nature.

Definition 22. *Let $\mathscr{A}$ be a WTA over a commutative semiring $\mathbb{S}$ and Σ, and $h\colon T_\Sigma \to T_\Delta$ a nondeleting and nonerasing, tetris-free tree homomorphism. We say that $\mathscr{A}$ is h-*unambiguous *if for all trees $s, s' \in T_\Sigma$ such that $h(s) = h(s')$, all accepting runs ρ, ρ' of $\mathscr{A}$ for s and s', respectively, and all $p \in \mathrm{pos}(s)$, the target states of the rules applied in ρ and ρ' at p, respectively, coincide.*

Remark 23. *The condition of h-unambiguity is stronger than unambiguity: For $s = s' \in \mathrm{supp}[\![\mathscr{A}]\!]$ we obtain that $\mathscr{A}$ has at most one accepting run for s (since runs of WTA are uniquely determined by the processed symbol and the target state at every position). A similar reasoning applies if we choose $s \neq s'$ with $h(s) = h(s')$: While the unique runs of $\mathscr{A}$ for s and s' may read different symbols, the states they pass through coincide at every position.*

Imposing these conditions on the input of the HOM-problem allows us to build on it with the arguments from the previous section.

Proposition 24. *Let $\mathscr{A}$ be a WTA over a commutative semiring $\mathbb{S}$ and Σ, and $h\colon T_\Sigma \to T_\Delta$ a nondeleting and nonerasing tree homomorphism. If h is tetris-free and $\mathscr{A}$ is h-unambiguous, then the eq-restricted WTAh $\mathscr{A}'$ for $h_{[\![\mathscr{A}]\!]}$ constructed in Lemma 8 is unambiguous.*

Proof. Let ϑ and ϑ' be accepting runs of $\mathscr{A}'$ for the same $t \in T_\Delta$. We prove the statement by contradiction, so assume that $\vartheta \neq \vartheta'$. Then there are two distinct runs ρ and ρ' of $\mathscr{A}$ such that $\vartheta = h^R(\rho)$ and $\vartheta' = h^R(\rho')$ as introduced in Definition 9. The mapping h^R does not modify the target states of runs, so both ρ and ρ' are accepting as well, and since $\mathscr{A}$ is unambiguous, they must process distinct trees s and s', respectively, with $h(s) = h(s')$. By the premises of the statement, at every $p \in \mathrm{pos}(s) = \mathrm{pos}(s')$ it is $h\big(s(p)\big) = h\big(s'(p)\big)$, and the target states of ρ and ρ' at p coincide, so although $\rho \neq \rho'$, after applying h it is $\vartheta = h^R(\rho) = h^R(\rho') = \vartheta'$, which contradicts our assumption. □

We want to illustrate the role played by our two conditions, the h-unambiguity and the tetris-freeness. Let us discuss this with the help of two counter-examples.

Example 25. *Consider the ranked alphabets $\Sigma = \{a^{(0)}, b^{(0)}, g^{(1)}\}$ and $\Delta = \{c^{(0)}, k^{(2)}\}$. Let $h\colon T_\Sigma \to T_\Delta$ be the tetris-free tree homomorphism from Example 21 induced by $h(a) = h(b) = c$ and $h(g) = k(x_1, x_1)$. Moreover, let $\mathscr{A} = \big(Q, \Sigma, Q, R, \mathrm{wt}\big)$ be the WTA over the arctic semiring $\mathbb{A}$ with $Q = \{q_a, q_b\}$ and the following rules and weights:*

$$R = \big\{\, a \to_0 q_a\,, \quad b \to_0 q_b\,, \quad g(q_a) \to_1 q_a\,, \quad g(q_b) \to_2 q_b \,\big\}\,.$$

The WTA $\mathscr{A}$ is unambiguous, but not h-unambiguous, since the runs for a and b target different states despite $h(a) = h(b)$. Evaluating the weights in $\mathbb{A}$, we obtain the tree series $[\![\mathscr{A}]\!]$ defined by

$$[\![\mathscr{A}]\!] : s \mapsto \begin{cases} n & \text{if } s = g^n(a) \\ 2n & \text{if } s = g^n(b) \end{cases}$$

The WTAh $\mathscr{A}' = \big(Q \dot\cup \{\bot\}, \Delta, Q, R', \mathrm{wt}'\big)$ recognizing $h_{[\![\mathscr{A}]\!]}$ which is obtained from Lemma 8 has the following rules and weights:

$$\begin{aligned} R = {} & \big\{\, c \to_0 q_a\,, \quad c \to_0 q_b\,, \quad k(q_a, \bot) \xrightarrow{1=2}_1 q_a\,, \quad k(q_b, \bot) \xrightarrow{1=2}_2 q_b \,\big\} \\ & \cup \big\{\, c \to_0 \bot\,, \quad k(\bot, \bot) \to_0 \bot \,\big\}\,. \end{aligned}$$

Because of the different target states, the rules $c \to q_a$ and $c \to q_b$ are not merged in $\mathscr{A}'$, therefore $\mathscr{A}'$ is not unambiguous.

On the other hand, let $\hat h$ be the homomorphism from Example 21 induced by $\hat h(a) = c$, $\hat h(b) = k(c,c)$ and $\hat h(g) = k(x_1, c)$. Recall that h is not tetris-free because $h\big(g(a)\big) = h(b)$ although $\mathrm{pos}\,\big(g(a)\big) \neq \mathrm{pos}(b)$. Moreover, consider the WTA $\hat{\mathscr{A}} = \big(\{q\}, \Sigma, \{q\}, \hat R, \hat{\mathrm{wt}}\big)$ over $\mathbb{N}$ with the following rules and weights:

$$\hat R = \big\{\, a \to_2 q\,, \quad b \to_3 q\,, \quad g(q) \to_1 q \,\big\}\,.$$

The WTA $\hat{\mathscr{A}}$ only has one state, so it is deterministic and thus unambiguous. It recognizes the tree series $[\![\hat{\mathscr{A}}]\!]$ defined by

$$[\![\hat{\mathscr{A}}]\!] : s \mapsto 2|\mathrm{pos}_a(s)| + 3|\mathrm{pos}_b(s)|\,.$$

However, the WTAh $\hat{\mathscr{A}}' = \big(\{q, \bot\}, \Delta, \{q\}, \hat R', \hat{\mathrm{wt}}'\big)$ for $h_{[\![\mathscr{A}]\!]}$ has the following rules and weights:

$$\begin{aligned} \hat R' = {} & \big\{\, c \to_2 q\,, \quad k(c,c) \to_3 q\,, \quad k(q,c) \to_1 q \,\big\} \\ & \cup \big\{\, c \to_1 \bot\,, \quad k(\bot, \bot) \to_1 \bot \,\big\}\,. \end{aligned}$$

Since $\hat h$ performs no duplications, the rules targeting $\bot$ are not used in any accepting run, so we can safely ignore them. Although this time, no two rules of $\hat{\mathscr{A}}'$ (that are used in an accepting run) share a left-hand side, the tree $k(c,c) = \hat h\big(g(a)\big) = \hat h(b)$ still has two different runs, which stem directly from the non-tetris-freeness of $\hat h$.

As a concequence of Proposition 24, our restricted version of the HOM-problem is decidable.

Corollary 26. *Let $\mathbb{S}$ be a zero-sum free, commutative semiring. For a nondeleting and nonerasing, tetris-free tree homomorphism h and an h-unambiguous WTA $\mathscr{A}$ over $\mathbb{S}$ as input, it is decidable whether the tree series $h_{[\![\mathscr{A}]\!]}$ is regular.*

5 Conclusion and Future Work

Homomorphic images of regular tree series can be represented using an extension of WTA, the so-called *eq-restricted WTAh* [22]. In this paper, we have shown that regularity is decidable for unambiguous devices of this type over zero-sum free commutative semirings. For this, we reduced this question to the unweighted setting, where regularity is known to be decidable [12]. Moreover, we have phrased

a condition on the input WTA $\mathscr{A}$ and tree homomorphism h that ensures unambiguity of the WTAh representing the image $h_{[\![\mathscr{A}]\!]}$. Thus the *HOM-problem* over zero-sum free semirings which, given $\mathscr{A}$ and h as input, asks whether $h_{[\![\mathscr{A}]\!]}$ is regular, is decidable if the input satisfies our condition.

Notably, the zero-sum freeness of the semiring is only used in Theorem 18 to show that if the tree series recognized by an unambiguous eq-restricted WTAh $\mathscr{A}$ is regular, then its support is also regular. It is plausible that the zero-sum freeness is not needed: Its purpose is to ensure that different accepting runs of $\mathscr{A}$ for the same tree t cannot cancel out, leaving $t \notin \operatorname{supp}[\![\mathscr{A}]\!]$ despite $\mathscr{A}$ having accepting runs for t. This, however, should not be a concern if $\mathscr{A}$ is unambiguous. To discard the zero-sum freeness assumption, it suffices to prove this simple statement: *If $\mathscr{A}$ is an unambiguous eq-restricted WTAh and $[\![\mathscr{A}]\!]$ is regular, then there is an unambiguous WTA equivalent to $\mathscr{A}$*. In fact, the linearization of $\mathscr{A}$, which is unambiguous by Lemma 16, is a promising candidate. Thus we conjecture that Theorem 18 holds for arbitrary commutative semirings, as do then Corollaries 19 – stating that regularity is decidable for unambiguous eq-restricted WTAh – and 26 – stating that the HOM-problem is decidable under our assumptions on $\mathscr{A}$ and h.

Recently, the disambiguation of weighted (string) automata from [24] was lifted to trees [30]. Here, the authors assume a variation of the *twins property* which restricts the behaviour of related states of a WTA. This allows them to construct an equivalent unambiguous WTA. A natural question is whether this proof can be adjusted to provide even an h-unambiguous WTA, say, by refining the twins property with respect to h. That way, we could lift our result to a larger class of input WTA.

References

[1] Symeon Bozapalidis & Antonios Kalampakas (2008): *Graph automata*. *Theoretical Computer Science* 393(1-3), pp. 147–165, doi:10.1016/j.tcs.2007.11.022.

[2] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison & M. Tommasi (2007): *Tree Automata — Techniques and Applications*.

[3] Carles Creus, Adrià Gascón, Guillem Godoy & Lander Ramos (2012): *The HOM problem is EXPTIME-complete*. In: *Proc. 27th Annual IEEE Symp. Logic in Computer Science*, IEEE, pp. 255–264, doi:10.1109/LICS.2012.36.

[4] Leonard E. Dickson (1913): *Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors*. *Amer. J. Math.* 35(4), pp. 413–422, doi:10.2307/2370405.

[5] John Doner (1970): *Tree acceptors and some of their applications*. *J. Comput. System Sci.* 4(5), pp. 406–451, doi:10.1016/S0022-0000(70)80041-1.

[6] Frank Drewes (2006): *Grammatical picture generation*. Springer.

[7] Manfred Droste & Doreen Heusel (2015): *The supports of weighted unranked tree automata*. *Funda. Inform.* 136(1–2), pp. 37–58, doi:10.3233/FI-2015-1143.

[8] Manfred Droste & Dietrich Kuske (2021): *Weighted automata.*

[9] Zoltán Fülöp & Heiko Vogler (2009): *Weighted tree automata and tree transducers*. In: *Handbook of Weighted Automata*, chapter 9, Springer, pp. 313–403, doi:10.1007/978-3-642-01492-5_9.

[10] Ferenc Gécseg & Magnus Steinby (2015): *Tree Automata*. Technical Report 1509.06233, arXiv.

[11] Dora Giammarresi & Antonio Restivo (1992): *Recognizable picture languages*. *International Journal of Pattern Recognition and Artificial Intelligence* 6(02n03), pp. 241–256, doi:10.1142/S021800149200014X.

[12] Guillem Godoy & Omer Giménez (2013): *The HOM problem is decidable*. *J. ACM* 60(4), pp. 1–44, doi:10.1145/2508028.2501600.

[13] Guillem Godoy, Omer Giménez, Lander Ramos & Carme Àlvarez (2010): *The HOM problem is decidable*. In: *Proc. 42nd ACM symp. Theory of Computing*, ACM, pp. 485–494, doi:10.1145/1806689.1806757.

[14] Jonathan S. Golan (1999): *Semirings and their Applications*. Kluwer Academic, Dordrecht, doi:10.1007/978-94-015-9333-5.

[15] Udo Hebisch & Hanns J. Weinert (1998): *Semirings*. World Scientific, doi:10.1142/3903.

[16] Udo Hebisch & Hanns Joachim Weinert (1998): *Semirings: algebraic theory and applications in computer science*. 5, World Scientific, doi:10.1142/9789812815965_bmatter.

[17] Dan Jurafsky & James H. Martin (2008): *Speech and language processing*. Prentice Hall.

[18] Daniel Kirsten (2011): *The support of a recognizable series over a zero-sum free, commutative semiring is recognizable*. *Acta Cybernet.* 20(2), pp. 211–221, doi:10.14232/actacyb.20.2.2011.1.

[19] Tetris Holding; The Tetris Company LLC (1985): *Tetris*. Available at https://tetris.com.

[20] Christof Loding & Wolfgang Thomas (2000): *Alternating Automata and Logics over Infinite Words*. In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses & Takayasu Ito, editors: *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, Springer Berlin Heidelberg, doi:10.1007/3-540-44929-9_36.

[21] Andreas Maletti & Andreea-Teodora Nász (2022): *Weighted tree automata with constraints*. In: *Developments in Language Theory: 26th International Conference, DLT 2022, Tampa, FL, USA, May 9–13, 2022, Proceedings*, Springer, pp. 226–238, doi:10.1007/978-3-031-05578-2_18.

[22] Andreas Maletti & Andreea-Teodora Nász (2023): *Weighted Tree Automata with Constraints*. Technical Report 2302.03434, arXiv. Available at https://arxiv.org/abs/2302.03434.

[23] Andreas Maletti, Andreea-Teodora Nász & Erik Paul (2023): *Weighted HOM-Problem for Nonnegative Integers*. arXiv:2305.04117.

[24] Mehryar Mohri & Michael D. Riley (2017): *A disambiguation algorithm for weighted automata*. *Theoretical Computer Science* 679, pp. 53–68, doi:10.1016/j.tcs.2016.08.019. Available at https://www.sciencedirect.com/science/article/pii/S0304397516304376. Implementation and Application of Automata.

[25] J. Mongy-Steen (1981): *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. Ph.D. thesis, Université de Lille.

[26] Dominique Perrin (1984): *Recent results on automata and infinite words*. In: *Proc. 11th Int. Symp. Mathematical Foundations of Computer Science, LNCS* 176, Springer, pp. 134–148, doi:10.1007/BFb0030294.

[27] Azriel Rosenfeld (2014): *Picture languages: formal models for picture recognition*. Academic Press.

[28] Arto Salomaa & Matti Soittola (1978): *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science, Springer, doi:10.1007/978-1-4612-6264-0.

[29] Marcel Paul Schützenberger (1961): *On the definition of a family of automata*. *Inform. and Control* 4(2–3), pp. 245–270, doi:10.1016/S0019-9958(61)80020-X.

[30] Kevin Stier & Markus Ulbricht (2021): *Disambiguation of Weighted Tree Automata*. In Yo-Sub Han & Sang-Ki Ko, editors: *Descriptional Complexity of Formal Systems*, Springer International Publishing, Cham, pp. 163–175, doi:10.1007/978-3-030-93489-7_14.

[31] James W. Thatcher (1967): *Characterizing Derivation Trees of Context-Free Grammars through a Generalization of Finite Automata Theory*. *J. Comput. Syst. Sci.* 1(4), pp. 317–322, doi:10.1016/S0022-0000(67)80022-9.

[32] James W. Thatcher & Jesse B. Wright (1968): *Generalized finite automata theory with an application to a decision problem of second-order logic*. *Math. Systems Theory* 2(1), pp. 57–81, doi:10.1007/BF01691346.

[33] Reinhard Wilhelm, Helmut Seidl & Sebastian Hack (2013): *Compiler Design - Syntactic and Semantic Analysis*. Springer, doi:10.1007/978-3-642-17540-4.

Once-Marking and Always-Marking 1-Limited Automata

Giovanni Pighizzini
Dipartimento di Informatica
Università degli Studi di Milano, Italy
pighizzini@di.unimi.it

Luca Prigioniero
Department of Computer Science
Loughborough University, UK
l.prigioniero@lboro.ac.uk

Single-tape nondeterministic Turing machines that are allowed to replace the symbol in each tape cell only when it is scanned for the first time are also known as 1-limited automata. These devices characterize, exactly as finite automata, the class of regular languages. However, they can be extremely more succinct. Indeed, in the worst case the size gap from 1-limited automata to one-way deterministic finite automata is double exponential.

Here we introduce two restricted versions of 1-limited automata, *once-marking* 1*-limited automata* and *always-marking* 1*-limited automata*, and study their descriptional complexity. We prove that once-marking 1-limited automata still exhibit a double exponential size gap to one-way deterministic finite automata. However, their deterministic restriction is polynomially related in size to two-way deterministic finite automata, in contrast to deterministic 1-limited automata, whose equivalent two-way deterministic finite automata in the worst case are exponentially larger. For always-marking 1-limited automata, we prove that the size gap to one-way deterministic finite automata is only a single exponential. The gap remains exponential even in the case the given machine is deterministic.

We obtain other size relationships between different variants of these machines and finite automata and we present some problems that deserve investigation.

1 Introduction

In 1967, with the aim of generalizing the concept of determinism for context-free languages, Hibbard introduced *limited automata*, a restricted version of Turing machines [3]. More precisely, for each fixed integer $d \geq 0$, a *d-limited automaton* is a single-tape nondeterministic Turing machine that is allowed to replace the content of each tape cell only in the first d visits.

Hibbard proved that, for each $d \geq 2$, d-limited automata characterize the class of context-free languages. For $d = 0$ these devices cannot modify the input tape, hence they are two-way finite automata, so characterizing regular languages. Furthermore, also 1-limited automata are no more powerful than finite automata. The proof of this fact can be found in [19, Thm. 12.1].

The investigation of these models has been reconsidered in the last decade, mainly from a descriptional point of view. Starting with [8, 9], several works investigating properties of limited automata and their relationships with other computational models appeared in the literature (for a recent survey see [7]).

In this paper we focus on 1-limited automata. We already mentioned that these devices are no more powerful than finite automata, namely they recognize the class of regular languages. However, they can be dramatically more succinct than finite automata. In fact, a double exponential size gap from 1-limited automata to one-way deterministic finite automata has been proved [8]. In other words, every n-state 1-limited automaton can be simulated by a one-way deterministic automaton with a number of states which is double exponential in n. Furthermore, in the worst case, this cost cannot be reduced.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 215–227, doi:10.4204/EPTCS.386.17

As pointed out in [8], this double exponential gap is related to a double role of the nondeterminism in 1-limited automata. When the head of a 1-limited automaton reaches for the first time a tape cell, it replaces the symbol in it according to a nondeterministic choice. Furthermore, the set of nondeterministic choices allowed during the next visits to the same cell depends on the symbol written in the first visit and that cannot be further changed, namely it depends on the nondeterministic choice made during the first visit.

With the aim of better understanding this phenomenon, we started to investigate some restrictions of 1-limited automata. On the one hand, we are interested in finding restrictions that reduce this double exponential gap to a single exponential. We already know that this happens for *deterministic* 1-limited automata [8]. So the problem is finding some restrictions that, still allowing nondeterministic transitions, avoid the double exponential gap. On the other hand, we are also interested in finding some very restricted forms of 1-limited automata for which a double exponential size gap in the conversion to one-way deterministic automata remains necessary in the worst case.

A first attempt could be requiring deterministic rewritings, according to the current configuration of the machine, every time cells are visited for the first time, still keeping nondeterministic the choice of the next state and head movement. Another attempt could be to allow nondeterministic choices for the symbol to rewrite, but not for the next state and the head movement. In both cases the double exponential gap to one-way deterministic finite automata remains possible. Indeed, in both cases, different computation paths can replace the same input prefix on the tape with different strings, as in the original model. Actually, we noticed that the double exponential gap can be achieved already for 1-limited automata that, in each computation, have the possibility to mark just one tape cell leaving the rest of the tape unchanged. This inspired us to investigate machines with such a restriction, which we call *once-marking* 1*-limited automata*. We show that the double exponential size gap to one-way deterministic finite automata remains possible even for once-marking 1-limited automata that are *sweeping* (namely, change the head direction only at the left or right end of the tape) and that are allowed to use nondeterminism only in the first visit to tape cells. Comparing the size of once-marking 1-limited automata with other kinds of finite automata, we prove an exponential gap to two-way nondeterministic automata. The situation changes significantly when nondeterministic transitions are not possible. Indeed, we prove that every deterministic once-marking 1-limited automaton can be converted into an equivalent two-way deterministic finite automaton with only a polynomial size increasing. The costs we obtain concerning once-marking 1-limited automata are summarized in Figure 2.

As mentioned above, the double exponential gap from 1-limited automata to one-way deterministic finite automata is related to the fact that different computation paths can replace the same input prefix on the tape with different strings. This suggested the idea of considering a different restriction, which prevents this possibility, by requiring the replacement of each input symbol a with a symbol that depends only on a. To this aim, here we introduce *always-marking* 1*-limited automata*, that in the first visit replace each symbol with a marked version of it. We show that in this case the gap from these devices, in the nondeterministic version, to one-way deterministic finite automata reduces to a single exponential. The same gap holds when converting always-marking 1-limited automata into one-way nondeterministic finite automata, but even when converting *deterministic* always-marking 1-limited automata into *two-way nondeterministic* finite automata. The bounds we obtain concerning always-marking 1-limited automata are summarized in Figure 3.

The paper is organized as follows. After presenting in Section 2 the preliminary notions used in the paper and, in particular, the definition of 1-limited automata with the fundamental results on their descriptional complexity, in Section 3 we introduce once-marking and always-marking 1-limited automata,

together with some witness languages that will be useful to obtain our results. Sections 4 and 5 are devoted to the investigation of the descriptional complexity of these models. We conclude the paper presenting some final remarks and possible lines for future investigations.

2 Preliminaries

In this section we recall some basic definitions useful in the paper. Given a set S, $\#S$ denotes its cardinality and 2^S the family of all its subsets. Given an alphabet Σ, a string $w \in \Sigma^*$, and a symbol $a \in \Sigma$, $|w|$ denotes the length of w, Σ^k the set of all strings on Σ of length k, $\dot{a}$ the *marked versions* of a, and $\dot{\Sigma} = \{\dot{a} \mid a \in \Sigma\}$ the set of the marked versions of the symbols in Σ.

We assume the reader familiar with notions from formal languages and automata theory, in particular with the fundamental variants of finite automata (1DFAs, 1NFAs, 2DFAs, 2NFAs, for short, where 1/2 mean *one-way/two-way* and D/N mean *deterministic/nondeterministic*, respectively). For any unfamiliar terminology see, e.g., [4].

A 1-*limited automaton* (1-LA, for short) is a tuple $\mathscr{A} = (Q, \Sigma, \Gamma, \delta, q_I, F)$, where Q is a finite *set of states*, Σ is a finite *input alphabet*, Γ is a finite *work alphabet* such that $\Sigma \cup \{\rhd, \lhd\} \subseteq \Gamma$, $\rhd, \lhd \notin \Sigma$ are two special symbols, called the *left* and the *right end-markers*, and $\delta : Q \times \Gamma \to 2^{Q \times (\Gamma \setminus \{\rhd, \lhd\}) \times \{-1, +1\}}$ is the *transition function*. At the beginning of the computation, the input word $w \in \Sigma^*$ is stored onto the tape surrounded by the two end-markers, the left end-marker being in position zero and the right end-marker being in position $|w| + 1$. The head of the automaton is on cell 1 and the state of the finite control is the *initial state* q_I.

In one move, according to δ and the current state, $\mathscr{A}$ reads a symbol from the tape, changes its state, replaces the symbol just read from the tape by a new symbol, and moves its head to one position forward or backward. Furthermore, the head cannot pass the end-markers, except at the end of computation, to accept the input, as explained below. Replacing symbols is allowed to modify the content of each cell only during the first visit, with the exception of the cells containing the end-markers, which are never modified. Hence, after the first visit, a tape cell is "frozen".[1]

The automaton $\mathscr{A}$ accepts an input w if and only if there is a computation path that starts from the initial state q_I with the input tape containing w surrounded by the two end-markers and the head on the first input cell, and that ends in a *final state* $q \in F$ after passing the right end-marker. The device $\mathscr{A}$ is said to be *deterministic* (D-1-LA, for short) whenever $\#\delta(q, \sigma) \leq 1$, for any $q \in Q$ and $\sigma \in \Gamma$.

Two-way finite automata are limited automata in which no rewritings are possible. On the other hand, one-way finite automata can scan the input in a one-way fashion only. A finite automaton is, as usual, a tuple $(Q, \Sigma, \delta, q_I, F)$, where, analogously to 1-LAs, Q is the finite set of states, Σ is the finite input alphabet, δ is the transition function, q_I is the initial state, and F is the set of final states. We point out that for two-way finite automata we assume the same accepting conditions as for 1-LAs.

Two-way machines in which the direction of the head can change only at the end-markers are said to be *sweeping* [18].

In this paper we are interested to compare the size of machines. The *size* of a model is given by the total number of symbols used to write down its description. Therefore, the size of 1-LAs is bounded by

[1] More technical details can be found in [8]. However, a syntactical restriction forcing 1-LAs to replace in the first visit to each tape cell the input symbol in it with another symbol from an alphabet Γ_1 disjoint from Σ, was given. Here we drop this restriction, in order to be able to see once-marking 1-LAs as a restriction of 1-LAs. It is always possible to transform a 1-LA into an equivalent 1-LA satisfying such a syntactical restriction, just extending Γ with a marked copy of Σ and suitably modifying the transition function.

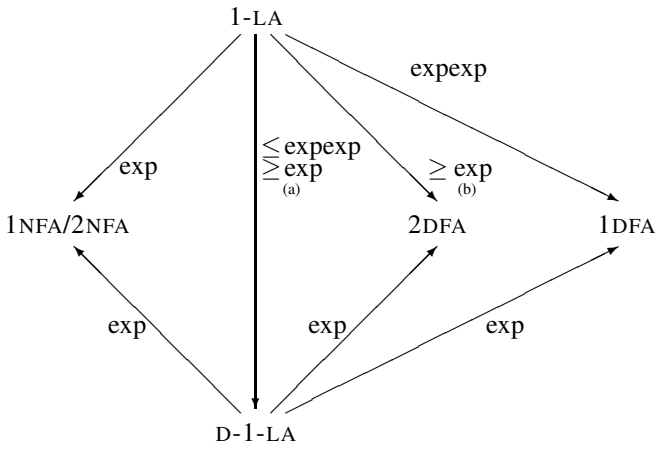


Figure 1: Size costs of conversions of 1-LAs and D-1-LAs into equivalent one-way and two-way deterministic and nondeterministic finite automata. For all the costs upper and matching lower bounds have been proved, with the only exception of (a) and (b), for which the best known lower and upper bounds are, respectively, exponential and double exponential.

a polynomial in the number of states and of work symbols, while, in the case of finite automata, since no writings are allowed, the size is linear in the number of instructions and states, which is bounded by a polynomial in the number of states and in the number of input symbols.

The size costs of the simulations from 1-LAs to finite automata have been studied in [8] and are summarized in Figure 1.

3 Witness Languages and Variants of 1-Limited Automata

As mentioned in the introduction, 1-LAs can be very succinct. In fact, for some languages the size gap to 1DFA is double exponential. We already observed that this gap is related to nondeterminism. Indeed, if nondeterministic choices are not possible, the gap reduces to a single exponential (see Figure 1). However, we want to understand better on the one hand how much we can restrict the model, still keeping this double exponential gap and, on the other hand, if there is a restriction that, still allowing some kind of nondeterminism, reduces the gap to a single exponential.

In our investigations, the following language, which is defined with respect to an integer parameter $n > 0$, will be useful:

$$K_n = \{x_1 \cdots x_k \cdot x \mid k > 0,\ x_1, \ldots, x_k, x \in \{a,b\}^n,\ \exists j \in \{1, \ldots, k\},\ x_j = x\}.$$

We point out that each string in the language is a list of blocks of length n. We ask the membership of the last block to the list of previous ones.

Theorem 1. *The language K_n is accepted by a* 1-LA *with $O(n)$ states that, in each accepting computation, replaces the content only of one cell.*

Proof. A 1-LA $\mathscr{M}$ can scan the tape from left to right, marking a nondeterministically chosen tape cell. In this scan, $\mathscr{M}$ can also verify that the input length is a multiple of n. Furthermore, the marking can be done in the last cell of a block of length n. For this phase $O(n)$ states are enough.

Then the machine has to compare the symbols in the last block with the symbols in the chosen one, namely the block which ends with the marked cell. This can be done by moving the head back and forth from the last block to the chosen block, comparing the symbols in the corresponding positions in the two blocks, and rejecting in case of mismatch. Again, this can be implemented, using a counter modulo n, with $O(n)$ states. □

Using standard distinguishability arguments, it can be proved that to accept K_n, a 1DFA requires a number of states double exponential in n (state lower bounds for K_n are summarized in Theorem 2 below).

Hence, the language K_n is a witness of the double exponential gap from 1-LAs to 1DFAs. From Theorem 1, we can notice that this gap is obtained by using the capabilities of 1-LAs in a very restricted way: during each accepting computation, only the content of one cell is modified. This suggested us to considering the following restricted version of 1-LAs:

Definition 1. *A* 1-LA *is said to be* once marking *if in each computation there is a unique tape cell whose input symbol σ is replaced with its marked version $\dot{\sigma}$, while all the remaining cells are never changed.*

In the following, for brevity, we indicate once-marking 1-LAs and once-marking D-1-LAs as OM-1-LAs and D-OM-1-LAs, respectively.

We shall consider another restriction, in which the 1-LA marks, in the first visit, every cell reached by the head.

Definition 2. *A* 1-LA *is said to be* always marking *if, each time the head visits a tape cell for the first time, it replaces the input symbol σ in it with its marked version $\dot{\sigma}$.*

In the following, for brevity, we indicate always-marking 1-LAs and always-marking D-1-LAs as AM-1-LAs and D-AM-1-LAs, respectively.

We point out that OM-1-LAs and AM-1-LAs use the work alphabet $\Gamma = \Sigma \cup \dot{\Sigma} \cup \{\triangleright, \triangleleft\}$. Hence, the relevant parameter for evaluating the size of these devices is their number of states, differently than 1-LAs, in which the size of the work alphabet is not fixed.

We present another language that will be used in the paper. As K_n, it is defined with respect to a fixed integer $n > 0$:

$$J_n = \{x \cdot x_1 \cdots x_k \mid k > 0,\ x_1, \ldots, x_k, x \in \{a,b\}^n,\ \exists j \in \{1, \ldots, k\},\ x_j = x\}.$$

Even in this case, a string is a list of blocks of length n. Here we ask the membership of the first block to the subsequent list. Notice that J_n is the reversal of K_n.

We have the following lower bounds:

Theorem 2. *Let $n > 0$ be an integer.*

- *To accept J_n,* 1DFA*s and* 1NFA*s need at least 2^n states, while* 2NFA*s need at least $2^{\frac{n-1}{2}}$ states.*
- *To accept K_n,* 1DFA*s need 2^{2^n} states,* 1NFA*s need at least 2^n states, and* 2NFA*s need at least $2^{\frac{n-1}{2}}$ states.*

Proof. (sketch) The lower bounds for one-way machines can be proved using standard distinguishability arguments and the fooling set technique [1] (see [8, 13] for similar proofs with slightly different languages).

Using a standard conversion, from a k-state 2NFA accepting K_n we can obtain an equivalent 1DFA with no more than 2^{k+k^2} states [14, 16]. Since every 1DFA accepting K_n should have at least 2^{2^n} states, we

get that $k+k^2 \geq 2^n$. Hence k grows as an exponential in n. In particular, it can be verified that $k > 2^{\frac{n-1}{2}}$. Since from each 2NFA accepting a language we can easily obtain a 2NFA with a constant amount of extra states accepting the reversal of such a language, we can conclude that the number of states of each 2NFA accepting J_n or K_n must be at least exponential in n. □

4 Once-Marking 1-Limited Automata

During each computation, *once-marking 1-limited automata* are able to mark just one input cell.

From Theorem 1, we already know that the language K_n can be accepted by a OM-1-LA with $O(n)$ states. We now show that such a machine can be turned in a even more restricted form:

Theorem 3. *The language K_n is accepted by a* OM-1-LA *with $O(n)$ states that is sweeping and uses nondeterministic transitions only in the first traversal of the tape.*

Proof. We discuss how to modify the $O(n)$-state OM-1-LA $\mathscr{M}$ described in the proof of Theorem 1 in order obtain a sweeping machine that uses nondeterministic transitions only in the first sweep. $\mathscr{M}$ makes a first scan of the input, exactly as described in the proof of Theorem 1. In this scan the head direction is never changed. When the right end-marker is reached, $\mathscr{M}$ makes n iterations, which in the following description will be counted from 0 to $n-1$.

The purpose of the iteration i, $i = 0,\ldots,n-1$, is to compare the $(n-i)$th symbols of the last block and of the chosen one. To this aim, the iteration starts with the head on the right end-marker, and uses a counter modulo n, initialized to $(i+1) \bmod n$. The counter is decremented while moving to the left. In this way, it contains 0 exactly while visiting the $(n-i)$th cell of each input block. Hence, the automaton can easily locate the $(n-i)$th symbols of the last block and of the chosen one and check if they are equal. Once the left end-marker is reached, $\mathscr{M}$ can cross the tape from left to right, remembering the number i of the iteration. Notice that $\mathscr{M}$ does not need to keep this number while moving from right to left. Indeed the value of i can be recovered from the value of the counter when the left end-marker is reached.

Once the iteration i is completed, if the last check was unsuccessful then $\mathscr{M}$ can stop and reject. Otherwise it can start the next iteration, if $i < n-1$, or accepts.

From the discussion above, it can be easily verified that $\mathscr{M}$ is sweeping, makes nondeterministic choices only in the first sweep, and has $O(n)$ many states. □

We now study the size relationships between OM-1-LAs and finite automata. First, we observe that OM-1-LAs can be simulated by 1NFAs and by 1DFAs at the costs of an exponential and a double exponential increase in the number of states, respectively. These upper bounds derive from the costs of the simulations of 1-LAs by finite automata presented in [8, Thm. 2]. By considering the language K_n, we can conclude that these costs cannot be reduced:

Theorem 4. *Let $\mathscr{M}$ be a n-state* OM-1-LA*s. Then $\mathscr{M}$ can be simulated by a* 1NFA *and by a* 2NFA *with a number of states exponential in n, and by a* 1DFA *with a number of states double exponential in n. In the worst case these costs cannot be reduced.*

Proof. The upper bounds derive from the cost of the simulations of 1-LAs by 1NFAs and 1DFAs given in [8, Thm. 2]. For the lower bounds we consider the language K_n. As proved in Theorem 3, this language can be accepted by a OM-1-LA with $O(n)$ states. Furthermore, according to Theorem 2, it requires a number of state exponential in n to be accepted by 1NFAs or 2NFAs, and a number of states double exponential in n to be accepted by 1DFAs. □

From Theorem 4, it follows that the ability of marking only once can give already a huge descriptional power. Furthermore, from Theorem 3, we can observe that this power is achievable even with a sweeping machine that does not use nondeterminism after the first sweep. From the size costs of the simulation of 1-LAs by finite automata (see Figure 1), we already know that nondeterminism is essential to obtain this huge descriptional power. We now prove that, without nondeterminism, the descriptional power on OM-1-LAs dramatically reduces:

Theorem 5. *For each n-state* D-OM-1-LA *there exists an equivalent* 2DFA *with* $O\left(n^3\right)$ *states.*

Proof. Let $\mathscr{A} = (Q, \Sigma, \Gamma, \delta, q_I, F)$ be a n-state D-OM-1-LA. We give a construction of an equivalent 2DFA $\mathscr{A}'$. Before doing that, let us introduce, from an high-level perspective, how the simulating machine works.

The 2DFA $\mathscr{A}'$ operates in different modes.

In the first part of the computation, before $\mathscr{A}$ marks *one* cell, $\mathscr{A}'$ is in beforeMarking mode, in which it simulates directly each transition of $\mathscr{A}$.

When $\mathscr{A}'$ has to simulate the transition $\delta(s,\sigma) = (\dot{\sigma}, d)$ used by $\mathscr{A}$ for marking a cell, besides changing its state and moving its head according to the transition, $\mathscr{A}'$ switches to afterMarking mode and stores in its finite control the symbol σ that has been marked and the state s in which $\mathscr{A}$ was immediately before the marking.

While in afterMarking mode, every time a cell is visited, $\mathscr{A}'$ has to select which transition of $\mathscr{A}$ to simulate depending on the symbol a scanned by the input head. There are two possibilities: if the scanned symbol is different than the symbol σ that has been marked, then the transition is simulated directly. Otherwise, $\mathscr{A}'$ switches to backwardSimulation mode (described later) to verify whether the current cell is the one that has been marked by $\mathscr{A}$. If this is the case, then $\mathscr{A}'$ simulates the transition of $\mathscr{A}$ on the marked symbol $\dot{\sigma}$, otherwise it simulates the transition on σ. In both cases $\mathscr{A}'$ keeps working in afterMarking mode, so selecting transitions according to the strategy described above, until there are no more moves to simulate. Therefore $\mathscr{A}'$ accepts if the last simulated transition corresponds to a right transition passing the right end-marker while simulating a final state of $\mathscr{A}$.

We now give some details on the backwardSimulation mode, which is the core of the simulation. We remind the reader that $\mathscr{A}'$ switches to this mode when, being in afterMarking mode, the input head is on a cell containing the symbol σ, which has been saved at the end of the beforeMarking mode. Let us indicate by j the current position of the head, namely the position that has to be verified.

The 2DFA $\mathscr{A}'$ has to verify whether j is the cell that has been marked by $\mathscr{A}$. To make this check, $\mathscr{A}'$ can verify whether the computation path of $\mathscr{A}$ on the given input reaches, from the initial configuration, a configuration with state s and the head on the currently scanned cell j (we remind the reader that s and σ have been saved in the control of $\mathscr{A}'$ when switching from beforeMarking to afterMarking mode), whose position, however, cannot be saved in the control.

To be sure that the machine does not "loses track" of the position j while performing this search, we use the following strategy:

- $\mathscr{A}'$ simulates a backward computation from the state s and the current position j.
- If the initial configuration of $\mathscr{A}$ is reached, then the cell from which the check has started is the one where the marking transition has been executed.
- At that point, the position j is recovered by "rolling back" the backward computation. This is done by repeating the (forward) computation of $\mathscr{A}$ from the initial configuration until a marking transition is used. In fact, since $\mathscr{A}$ is deterministic and once marking, this transition is necessarily

the one that, from the state s, marked σ. In other words, the forward computation of $\mathscr{A}$ that is simulated here is the same simulated in beforeMarking mode.

As we shall explain later, even in the case the initial configuration of $\mathscr{A}$ is not reached (namely the verification is not successful), our technique allows to recover the head position j from which the backward simulation started,

It is important to observe two key points for which this approach works. The first one is that OM-1-LAs mark only one cell during their computation. The second observation is that the simulated machine is deterministic. Therefore, along every accepting computation path from the initial configuration, it occurs only once that the symbol σ is scanned while $\mathscr{A}$ is in state s, which is when $\mathscr{A}$ makes a marking transition.

To make such a verification, and in particular the backward search, we use a technique originally introduced by Sipser [17]. This simulation has been then refined by Geffert, Mereghetti, and Pighizzini, which proved that 2DFAs can be made halting with a linear increase of the number of states [2]. In the following, we shall refer to the latter simulation as the *original simulation* and use the notation and terminology contained in [2], to which we address the interested reader for missing details.

The main difference with the original simulation is that there the simulating machine starts from the final configuration of the simulated device, because the goal is to verify the presence of an accepting computation path. In our case, the machine $\mathscr{A}'$ starts the backward simulation from the state s and the cell containing σ that has to be checked.

In the following, a *configuration* is a pair (q,i), where q is the current state and i is the position of the tape head.

Consider the graph whose nodes represent configurations and edges computation steps. Since $\mathscr{A}$ is deterministic, the component of the graph containing (s,j) is a tree rooted at this configuration, with backward paths branching to all possible predecessors of (s,j). In addition, no backward path starting from (s,j) can loop (hence, it is of finite length), because the marking configuration (s,j) cannot be reached by a forward path from a loop (due to the fact that the machine is deterministic).

The simulating machine $\mathscr{A}'$ can perform a depth-first search of this tree in order to detect whether the initial configuration $(q_I,0)$ belongs to the predecessors of (s,j). If this is the case, then the machine returns to the position j, by performing a forward simulation of $\mathscr{A}$ from $(q_I,0)$ until when s is entered while reading the symbol σ. We stress that this approach works because the simulated machine is deterministic. After that, the simulation of $\mathscr{A}$ in afterMarking mode is recovered by performing a move on the symbol $\dot{\sigma}$. On the other hand, if the whole tree has been examined without reaching $(q_I,0)$, then the cell in position j is not the marked one, so the machine simulates a move of $\mathscr{A}'$ on σ from the cell in position j, again switching back to afterMarking mode. Notice that this case occurs when there are no more predecessors of (s,j) to visit. So, in this case, the machine $\mathscr{A}'$ completes the depth-first search on the cell in position j, while looking for further nodes of the graph reachable from the configuration (s,j). Hence, no extra steps are required to retrieve the position j.

In conclusion, $\mathscr{A}'$ has three state components of size $O(n)$: one used in beforeMarking and afterMarking for the direct simulation of the transitions of $\mathscr{A}$, one for storing the state s and the symbol σ, and one used in backwardSimulation mode. So, the total number of states of $\mathscr{A}'$ is $O\left(n^3\right)$. □

In Figure 2 the state costs of the conversions involving OM-1-LAs are summarized. In particular, we proved that the size gap from OM-1-LAs to 2NFAs is exponential and to 1DFAs is double exponential, while D-OM-1-LAs and 2DFAs are polynomially related in size.

Some questions remain open, in particular about the costs of the simulations of OM-1-LAs by D-OM-1-LAs and by 2DFAs. At the moment, from the above mentioned results, we can derive double

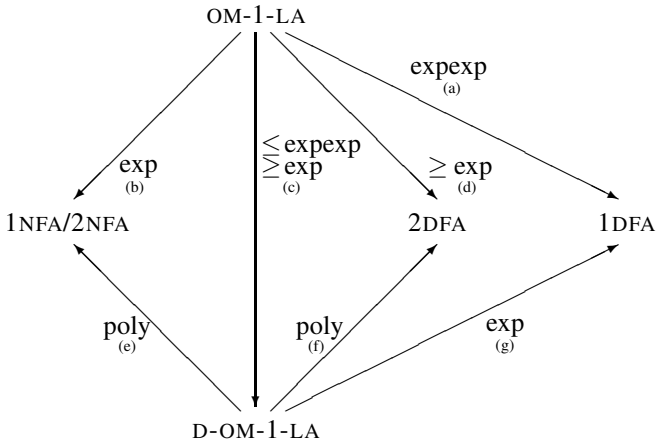


Figure 2: Size costs of conversions involving OM-1-LAs. The gaps (a) and (b) derive from Theorem 4. For (c) and (d) the lower bound derives from the lower bound of the language K_n on 2NFAs (Theorem 2); the best known upper bound derives from (a). The bounds (e) and (f) are from Theorem 5. The upper bound for (g) derives from the conversion from D-1-LAs and the lower bound from the conversion from 2DFAs.

exponential upper bounds and exponential lower bounds. The same questions are open for the simulation of 1-LAs by D-1-LAs and by 2DFAs, namely by dropping the once-marking restriction. We point out that these questions are related to the problem of the cost of the elimination of nondeterminism from two-way finite automata, proposed by Sakoda and Sipser in 1978 [15], which is still open.

5 Always-Marking 1-Limited Automata

Always-marking 1-limited automata replace, when they visit each cell for the first time, the input symbol with its marked version. In this section we study the descriptional complexity of these devices.

First of all, we prove that AM-1-LAs cannot achieve the same succinctness as 1-LAs. In fact, the size gap to 1DFAs reduces from double exponential for 1-LAs to single exponential.

Theorem 6. *Each n-state* AM-1-LA *can be simulated by a* 1NFA *with at most $n \cdot 2^{n^2}$ states and by a complete* 1DFA *with at most $(2^n-1)\cdot 2^{n^2}+1$ states.*

Proof. Let $\mathscr{M} = (Q,\Sigma,\Gamma,\delta,q_0,F)$ be a given n-state AM-1-LA. We adapt the argument used in [8] to convert 1-LAs into 1NFAs and 1DFAs, which is derived from the technique to convert 2DFAs into equivalent 1DFAs, presented in [16], and based on *transitions tables*.

Roughly, transition tables represent the possible behaviors of $\mathscr{M}$ on frozen tape segments. More precisely, given $z \in \Gamma^*$, the *transition table* associated with z is the binary relation $\tau_z \subseteq Q \times Q$, consisting of all pairs (p,q) such that $\mathscr{M}$ has a computation path that starts in the state p on the rightmost symbol of the tape segment containing $\triangleright z$, ends entering the state q by leaving the same tape segment to the right side, i.e., by moving from the rightmost cell of the segment to the right, and does not visit any cell outside the segment.

First, we can apply the conversion presented in [8] from 1-LAs to 1NFAs, in order to obtain from $\mathscr{M}$ an equivalent 1NFA A, whose computations simulate the computations of $\mathscr{M}$ by keeping in the finite state control two components:

- The transition table associated with the part of the tape at the left of the head. This part has been already visited and, hence, it is frozen.
- The state in which the simulated computation of $\mathscr{M}$ reaches the current tape position for the first time.

For details we address the reader to [8, Thm. 2]. Since the number of transition tables is at most 2^{n^2}, the number of states in the resulting 1NFA A is bounded by $n \cdot 2^{n^2}$.

Applying the subset construction, this automaton can be converted into an equivalent deterministic one, with an exponential increase of the number of states, so obtaining a double exponential number of states in n. In the general case, this increasing cannot be reduced. This is due to the fact that different computations of A, after reading the same input, could keep in the control different transitions tables, depending on the fact that $\mathscr{M}$ can replace the same input by different strings.

However, under the restriction we are considering, along different computations, each input string x is always replaced by the same string $\dot{x}$, which is obtained by marking every symbol of x. Hence, at each step of the simulation, the transition table stored by A depends only on the input prefix already inspected. The only part that can change is the state of the simulated computation of $\mathscr{M}$ after reading x.

This allows to obtain from A a 1DFA A', equivalent to $\mathscr{M}$ that, after reading a string x, has in its finite state control the transition table associated with $\dot{x}$, and the *set* of states that the computations of $\mathscr{M}$ can reach after reading x. In other words, the automaton A' is obtained from A by keeping the first component of the control, which is deterministic, and making a subset construction for the second one.

By summarizing, the possible values of the first component are 2^{n^2}, while the values of the second one are 2^n, namely the possible subsets of the state set of $\mathscr{M}$. This gives a $2^n \cdot 2^{n^2}$ upper bound. We can slightly reduce this number, by observing that when the second component contains the empty set, i.e., each computation of $\mathscr{M}$ (or equivalently of A) stops before entering it, then the input is rejected, regardless the first component. Hence, we can replace all the pairs having the empty set as a second component with a unique sink state, so reducing the upper bound to $(2^n - 1) \cdot 2^{n^2} + 1$ □

The asymptotical optimality of the upper bounds in Theorem 6 derives from the optimality of the conversions from 2NFAs to 1NFAs and to 2DFAs [14, 16, 5].

We now show that AM-1-LAs can be more succinct than 2NFAs, even in the deterministic case. In particular we prove the following:

Theorem 7. *The language J_n is accepted by a* D-AM-1-LA *with $O(n)$ states, while it cannot be accepted by any* 2NFA *with less than $2^{\frac{n-1}{2}}$ states.*

Proof. The lower bound for 2NFAs has been given in Theorem 2. The possibility of marking the already-visited cells allows to reduce this cost, even without making use of the nondeterminism, as we now describe. An always marking D-1-LA $\mathscr{M}$ can firstly visit and mark the first n tape cells. Then, it starts to inspect the next block of length n. When the head reaches for the first time a cell, $\mathscr{M}$ remembers the scanned symbol σ in it and moves the head back to the left end-marker and then to the corresponding cell in the first block (this can be implemented with a counter modulo n). If the symbol in this cell is not σ then $\mathscr{M}$ has to skip the remaining symbols in the block under inspection and inspect the next block, if any. This can be done moving the head to the left end-marker and then, starting to count modulo n, moving to the right until finding the first symbol of the next block. This symbol can be located using the value of the counter and the fact that it has not been marked yet. Otherwise, if the symbol in the cell coincides with σ and the block is not completely inspected (see below), $\mathscr{M}$ moves the head to the right to search the next symbol of the block under inspection, namely the first unmarked symbol.

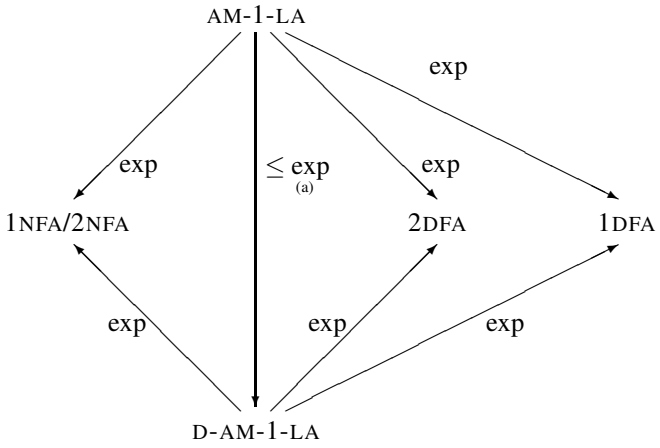


Figure 3: State costs of conversions involving AM-1-LAs. All the exponential upper bounds derive from Theorems 6 and 8, while the lower bounds derive from Theorem 7. For (a) we do not know if in the worst case an exponential size is also necessary.

When locating a symbol, $\mathscr{M}$ can also check and remember if it is in position n. This is useful to detect whether a block has been completely scanned, which also means that the block has been *successfully scanned*, otherwise the machine would have already rejected. Hence, in this case, $\mathscr{M}$ can move the head to the right to finally reach the accepting configuration. However, according to the definition of J_n, before doing that, $\mathscr{M}$ needs to verify that the input has length multiple of n. All these steps can be implemented with a fixed number of variables and a counter modulo n. This allows to conclude that $\mathscr{M}$ can be implemented with $O(n)$ states. □

In Theorem 7 we proved an exponential gap from D-AM-1-LAs to 2NFAs and hence also to one-way finite automata. This allows to conclude that the following upper bounds, that are immediate consequences of the corresponding upper bounds for D-1-LAs [8, Thm. 2], cannot be significantly reduced:

Theorem 8. *Each n-state* D-AM-1-LA *can be simulated by a* 1DFA *and by a* 1NFA *with no more than* $n \cdot (n+1)^n$ *states.*

From the discussion above and Theorem 8, we have the same state gap from D-AM-1-LAs and from D-1-LAs to one-way automata.

The state costs of the conversions involving AM-1-LAs are summarized in Figure 3.

Even in the case of AM-1-LAs, as well as in the cases of 1-LAs and of OM-1-LAs, we do not know how much the elimination of the nondeterminism costs. Here, we have an exponential upper bound for the conversion of AM-1-LAs into D-AM-1-LAs but, at the moment, we do not have a matching lower bound. Considering the conversion of AM-1-LAs into 2DFAs, unlikely the analogous conversions from 1-LAs and OM-1-LAs, here we have matching exponential upper and lower bounds. As already mentioned at the end of Section 4, these questions are related to the open question of Sakoda and Sipser.

6 Conclusion

We study the costs of the simulations of OM-1-LAs and AM-1-LAs by finite automata. Figures 2 and 3 give a summary of the results we obtained. They can be compared with the costs of the simulations

concerning 1-LAs, in Figure 1.

We observed that AM-1-LAs cannot reach the same succinctness as 1-LAs and OM-1-LAs (see Theorems 4 and 6). In particular, in Theorem 3 we have shown that the language K_n can be accepted by a OM-1-LA with $O(n)$ states. Hence, it requires an exponential number of states on AM-1-LAs due to the fact that a double exponential number of states on 1DFAs is necessary (see Theorem 2). It is not difficult to describe a 2NFA accepting K_n with an exponential number of states. We point out that such a machine is also a AM-1-LA. Hence, by summarizing, the language K_n is accepted by a OM-1-LA with $O(n)$ states, by an AM-1-LA with a number of states exponential in n, and by a 1DFA with a number of states double exponential in n. All these costs cannot be reduced.

Since in the nondeterministic case the gaps from OM-1-LAs to finite automata are the same as from 1-LAs, a natural question is to ask if OM-1-LAs are always as succinct as 1-LAs. Intuitively the answer to this question is negative. For instance we do not see how to recognize the language whose strings are concatenations of blocks of length n, in which two blocks are equal, with a OM-1-LA with $O(n)$ states, while it is not hard to accept it using a 1-LA with such a number of states. We leave the study of this question for a future work.

Another candidate for studying this question is the unary language $(a^{2^n})^*$. We proved that this language can be accepted by a D-1-LA with $O(n)$ states and a work alphabet of cardinality $O(n)$, and by a D-1-LA with $O(n^3)$ states and work alphabet of size not dependent on n [10, 12]. As pointed out in [10], each 2NFA accepting it requires at least 2^n states. Hence, by Theorem 5 even each D-OM-1-LA accepting it requires an exponential number of states. We do not see how to reduce this number even by allowing the use of nondeterminism on OM-1-LAs or on AM-1-LAs.

More in general, the comparisons between the sizes of these restricted versions of 1-LAs deserve further investigation, even in the unary case where the cost of several simulations are still unknown [10]. In a recent paper, we investigated *forgetting* 1-LA*s*, another restriction of 1-LAs in which there is a unique symbol X that is used to replace input symbols. Therefore, during the first visit to a cell, its original content is always replaced by X [11].

Finally, we would like to mention once again the problem of the cost of removing nondeterminism from 1-LAs, OM-1-LAs, and AM-1-LAs (see Sections 4 and 5), which is connected to the main question of the cost of the elimination of nondeterminism from two-way finite automata, raised longtime ago by Sakoda and Sipser and still open [15] (for a survey, see [6]).

References

[1] Jean-Camille Birget (1992): *Intersection and Union of Regular Languages and State Complexity*. *Inf. Process. Lett.* 43(4), pp. 185–190, doi:10.1016/0020-0190(92)90198-5.

[2] Viliam Geffert, Carlo Mereghetti & Giovanni Pighizzini (2007): *Complementing two-way finite automata*. *Inf. Comput.* 205(8), pp. 1173–1187, doi:10.1016/j.ic.2007.01.008.

[3] Thomas N. Hibbard (1967): *A Generalization of Context-Free Determinism*. *Inf. Control.* 11(1/2), pp. 196–238, doi:10.1016/S0019-9958(67)90513-X.

[4] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

[5] Christos A. Kapoutsis (2005): *Removing bidirectionality from nondeterministic finite automata*. In: *MFCS 2005*, *Lecture Notes in Computer Science* 3618, Springer, pp. 544–555, doi:10.1007/11549345_47.

[6] Giovanni Pighizzini (2013): *Two-Way Finite Automata: Old and Recent Results*. *Fundam. Inform.* 126(2-3), pp. 225–246, doi:10.3233/FI-2013-879.

[7] Giovanni Pighizzini (2019): *Limited Automata: Properties, Complexity and Variants*. In: *DCFS 2019*, *Lecture Notes in Computer Science* 11612, Springer, pp. 57–73, doi:10.1007/978-3-030-23247-4_4.

[8] Giovanni Pighizzini & Andrea Pisoni (2014): *Limited Automata and Regular Languages*. *Int. J. Found. Comput. Sci.* 25(7), pp. 897–916, doi:10.1142/S0129054114400140.

[9] Giovanni Pighizzini & Andrea Pisoni (2015): *Limited Automata and Context-Free Languages*. *Fundam. Inform.* 136(1-2), pp. 157–176, doi:10.3233/FI-2015-1148.

[10] Giovanni Pighizzini & Luca Prigioniero (2019): *Limited automata and unary languages*. *Inf. Comput.* 266, pp. 60–74, doi:10.1016/j.ic.2019.01.002.

[11] Giovanni Pighizzini & Luca Prigioniero (2023): *Forgetting 1-Limited Automata*. In: *NCMA 2023*, Electronic Proceedings in Theoretical Computer Science. To appear. A preliminary version is available at https://doi.org/10.48550/arXiv.2307.16700.

[12] Giovanni Pighizzini & Luca Prigioniero (2023): *Two-way Machines and de Bruijn Words*. In: *CIAA 2023*, *Lecture Notes in Computer Science* 14151, pp. 254–265, doi:10.1007/978-3-031-40247-0_19.

[13] Giovanni Pighizzini, Luca Prigioniero & Simon Šádovský (2022): *1-Limited Automata: Witness Languages and Techniques*. *J. Autom. Lang. Comb.* 27(1-3), pp. 229–244, doi:10.25596/jalc-2022-229.

[14] Michael O. Rabin & Dana S. Scott (1959): *Finite Automata and Their Decision Problems*. *IBM J. Res. Dev.* 3(2), pp. 114–125, doi:10.1147/rd.32.0114.

[15] William J. Sakoda & Michael Sipser (1978): *Nondeterminism and the Size of Two Way Finite Automata*. In: *STOC 1978*, ACM, pp. 275–286, doi:10.1145/800133.804357.

[16] John C. Shepherdson (1959): *The Reduction of Two-Way Automata to One-Way Automata*. *IBM J. Res. Dev.* 3(2), pp. 198–200, doi:10.1147/rd.32.0198.

[17] Michael Sipser (1980): *Halting Space-Bounded Computations*. *Theor. Comput. Sci.* 10, pp. 335–338, doi:10.1016/0304-3975(80)90053-5.

[18] Michael Sipser (1980): *Lower Bounds on the Size of Sweeping Automata*. *J. Comput. Syst. Sci.* 21(2), pp. 195–202, doi:10.1016/0022-0000(80)90034-3.

[19] Klaus W. Wagner & Gerd Wechsung (1986): *Computational complexity*. D. Reidel Publishing Company, Dordrecht.

A General Approach to Proving Properties of Fibonacci Representations via Automata Theory

Jeffrey Shallit* and Sonja Linghui Shan
School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
shallit@uwaterloo.ca, slshan@uwaterloo.ca

We provide a method, based on automata theory, to mechanically prove the correctness of many numeration systems based on Fibonacci numbers. With it, long case-based and induction-based proofs of correctness can be replaced by simply constructing a regular expression (or finite automaton) specifying the rules for valid representations, followed by a short computation. Examples of the systems that can be handled using our technique include Brown's lazy representation (1965), the far-difference representation developed by Alpert (2009), and three representations proposed by Hajnal (2023). We also provide three additional systems and prove their validity.

1 Introduction

Given an increasing sequence $(s_n)_{n\geq 0}$ of positive integers, a numeration system is a way of expressing natural numbers as a linear combination of the s_n. Many different numeration systems, such as representation in base k, or the more exotic systems based on the Fibonacci numbers, have been proposed. For example, recall that the Fibonacci numbers, sequence A000045 in the *On-Line Encyclopedia of Integer Sequences* (OEIS), are defined by the recurrence $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$ and the initial values $F_0 = 0$, $F_1 = 1$. Consider writing a non-negative integer n as a sum of distinct Fibonacci numbers F_i for $i \geq 2$. Some numbers, such as 12, have only one such representation $(12 = 8+3+1 = F_6 + F_4 + F_2)$, while others have many: $8 = F_6 = F_5 + F_4 = F_5 + F_3 + F_2$.

There are two very desirable characteristics of a numeration system. First, *completeness:* every natural number should have a representation. Second, *unambiguity:* no natural number should have two or more different representations. These two goals are typically achieved by restricting the types of representations that are considered valid within the system. If a system achieves both goals, we say it is *perfect*. For Fibonacci representations, various perfect systems have been proposed.

Among all possible perfect systems based on Fibonacci numbers, one is particularly useful: the *Zeckendorf* or *greedy* representation. This representation can be computed as follows: first, choose the largest index i such that $F_i \leq n$. Then the representation for n is F_i plus the (recursively-computed) representation for $n - F_i$. The representation for 0 is the empty sum of 0 Fibonacci numbers. A simple induction now shows that the greedy algorithm produces a representation for every natural number, which is evidently unique.

This representation was originally noted by Zeckendorf, but was first published by Lekkerkerker [12] and only later by Zeckendorf himself [20]. It was also anticipated, in much more general form, by Ostrowski [15].

An alternative (but equivalent) definition of Zeckendorf representation is to impose a condition that valid representations must obey. For example, we could require that a representation be valid if and only if no two consecutive Fibonacci numbers appear in the sum.

*Research funded by a grant from NSERC, 2018-04118.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 228–242, doi:10.4204/EPTCS.386.18

It is convenient to express arbitrary sums of distinct Fibonacci numbers as strings of digits over a finite alphabet (in analogy with base-k representation). Let $x = a_1 \cdots a_t$ be a string (or word) made up of integer digits. We define its value as a Fibonacci representation as follows:

$$[x]_F := \sum_{1 \le i \le t} a_i F_{t+2-i}. \tag{1}$$

Note that these strings are in "most-significant-digit" first format. For example, $[2101]_F = 2F_5 + F_4 + F_2 = 14$.

It is also useful to define a (partial) inverse to $[x]_F$. By $(n)_F$ we mean the binary string x such that x is the Zeckendorf representation of n; alternatively, such that $[x]_F = n$ and x contains no occurrence of the block 11. In what follows, we adopt this string-based point of view almost exclusively. We can think of the condition "no occurrence of the block 11" as a *rule*, specifying which representations are valid, adopted precisely to guarantee both completeness and unambiguity.

In formal language theory, a language L is a (finite or infinite) collection of strings. A rule is then encoded by the language or set of strings that obey the rule. Completeness then becomes the assertion that for all n there exists a string $x \in L$ such that $[x]_F = n$, while unambiguity becomes the assertion that there do not exist distinct strings $x, y \in L$ such that $[x]_F = [y]_F$. [1]

Let us look at another example involving the Fibonacci numbers, one that is much less well known: the so-called *lazy* representation [3]. In this system, representation as a sum of Fibonacci numbers corresponds (via Eq. (1)) to a binary string having no occurrence of the block 00 (where leading zeros are not even considered). Once again, this rule provides a numeration system that is both complete and unambiguous [3]. Table 1 gives both greedy (Zeckendorf) and lazy representations for the first few natural numbers.

n	0	1	2	3	4	5	6	7	8	9	10	11
greedy	ε	1	10	100	101	1000	1001	1010	10000	10001	10010	10100
lazy	ε	1	10	11	101	110	111	1010	1011	1101	1110	1111

Table 1: Greedy and lazy Fibonacci representations.

The greedy and lazy representations are certainly not the only possible perfect numeration systems based on the Fibonacci numbers. In fact, there are *uncountably many* such systems! These result from making a choice, for all n having at least two different representations as sums of distinct Fibonacci numbers, about which particular representation is chosen to be valid. (By a result of Robbins [17], "most" numbers have more than one representation as a sum of distinct Fibonacci numbers.)

If we demand that the set of valid representations forms a regular language—that is, accepted by a finite automaton; see Section 2—there are still infinitely many different systems (although only countably many). For example, consider choosing the t'th largest possible representation for n in lexicographic order (if there are at least t), and otherwise the lexicographically first. It will follow from results below that, for each $t \ge 0$, this choice gives a regular language L_t of valid representations.

Some natural questions then arise: given a language L encoding the "rule" a representation must obey (such as no occurrence of the block 11, or no occurrence of the block 00), how can we determine if the corresponding set of Fibonacci representations is complete and unambiguous? And if it is complete,

[1] We adopt the convention that two strings are considered to be the same if they differ only in the number of leading zeros. Thus, for example, $[100]_F = [0100]_F = 3$ are the same representation.

how can we efficiently find a representation for a given number n? Up to now, each new system proposed required a new proof, often a rather tedious case-based proof by induction. In this paper we provide a *general framework* for answering these questions "automatically", via an algorithm, in the case where the language of valid representations is regular.

These ideas are capable of generalization. For example, we can also consider representations for all integers $\mathbb{Z}$, instead of just the natural numbers $\mathbb{N}$. This can be achieved in two distinct ways:

- By allowing a larger digit set, say, $\{-1, 0, 1\}$;
- By using the so-called negaFibonacci system, based on the Fibonacci numbers of negative index F_{-n} for $n \geq 1$.

Once again, we would like a choice of valid representations that is complete and unambiguous.

In this paper we show how to decide these properties, provided that the set of valid representations forms a regular language (which is indeed the case for all the proposed systems in the literature).

Here is an outline of the paper. In Section 2, we explain the basics of automata theory needed to understand the rest of the paper. In Section 3, we discuss how to test completeness and ambiguity for systems using digits 0 and 1 only. In Section 4 we discuss systems using digits $-1, 0, 1$ only. In Section 5 we discuss representations for all integers, not just the natural numbers. In Section 6 we discuss an entirely new type of Fibonacci representation based on dictionary order. Finally, in Section 7 we describe a few of the new Fibonacci representations we found through exhaustive search of small automata.

2 The decision procedure and `Walnut`

We assume the reader is familiar with the basics of automata theory as discussed, for example, in [10].

The following particular case of a theorem of Büchi [6] (as later corrected by Bruyère et al. [5] is our principal tool in the paper.

Theorem 1. *There is a decision procedure that, given a first-order logical formula F involving natural numbers, comparisons, automata, and addition, and no free variables, will decide the truth or falsity of F. Furthermore, if F has free variables, the procedure constructs a DFA accepting those values of the free variables (in Fibonacci representation) that make F evaluate to* `TRUE`.

For more information about the specific case of the decision procedure for Fibonacci representation, see [14].

We should explain how automata can process pairs, triples, and generally k-tuples of inputs. This is done by replacing the input alphabet Σ with the alphabet $\overbrace{\Sigma \times \Sigma \times \cdots \times \Sigma}^{k \text{ times}}$. In other words, inputs are k-tuples of alphabet symbols. The i'th input then corresponds to the concatenation of the i'th components of all the k-tuples. Of course, this means that all k inputs have to have the same length; this is achieved by padding shorter inputs, if necessary, with leading zeros.

The decision procedure of Theorem 1 has been implemented in free software called `Walnut`, originally created by Hamoon Mousavi [13]; also see the book [19]. We recall some of the basics of `Walnut` syntax:

- `eval` evaluates a formula with no free variables and returns `TRUE` or `FALSE`; `def` defines an automaton for future use; `reg` defines a regular expression.
- In a regular expression, the period is an abbreviation for the entire alphabet.

- `&` is logical `AND`, `|` is logical `OR`, `=>` is logical implication, `<=>` is logical `IFF`, `~` denotes logical `NOT`.
- `A` denotes $\forall$ (for all); `E` denotes $\exists$ (there exists).
- `?msd_fib` tells `Walnut` to evaluate an arithmetic expression using Fibonacci representation.

We use `Walnut` to do the computations needed to verify that a given system is complete and unambiguous. For much more about `Walnut`, including a link to download it, visit

`https://cs.uwaterloo.ca/~shallit/walnut.html` .

3 Representation of natural numbers using digits 0 and 1 only

In this section we consider representations of the natural numbers by Fibonacci numbers using digits 0 and 1 only.

The first step is to find an automaton that can convert from an arbitrary Fibonacci representation to the greedy or Zeckendorf representation. To do this we use the following simple observation:

Proposition 2. *We can convert a binary string x to a Zeckendorf representation y for the same number using the following algorithm: first append a 0 on the front, if necessary. Then scan the string from left to right, replacing each occurrence of "011" successively with "100".*

Proof. Clearly each such replacement does not change the value of $[x]_F$. The algorithm terminates because each replacement lowers the total number of 1's by 1. Finally, the algorithm clearly cannot result in two consecutive 1's, because it introduces two consecutive 0's, only the second of which can later change to a 1. □

We can implement this idea as a DFA C that takes two inputs in parallel, x and y, and accepts if and only if both $[x]_F = [y]_F$ and y is a valid Zeckendorf representation; that is, it contains no two consecutive 1's. It suffices to keep track of $[x']_F - [y']_F$ for the prefix x' of x seen so far, and similarly for the prefix y' of y seen so far. Note that we assume that x and y have the same length, with the shorter of the two prefixed by leading zeros, if necessary. We can think of this as a "converter" or "normalizer" that allows us to turn arbitrary Fibonacci representations into Zeckendorf representations. It is depicted in Figure 1. This automaton was given by Berstel [2] in a slightly different form. Also see [18].

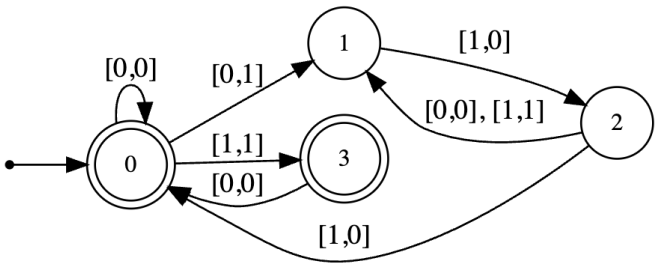


Figure 1: DFA C for conversion to the Zeckendorf representation.

As an example, consider the input $[0,1][1,0][1,0][1,1][0,0]$ to C, whose first components spell out $x = 01110$ and whose second components spell out $y = 10010$. Starting in state 0, the automaton visits, successively, states $1,2,0,3,0$, and hence accepts—as it should, since $[x]_F = [y]_F$.

We now state one of our main results.

Theorem 3. *There is an algorithm that, given rules that specify which representations are valid (in the form of a regular language L of all valid representations), will decide if the corresponding numeration system based on the Fibonacci numbers is complete and unambiguous for* $\mathbb{N}$.

Proof. Using Theorem 1, it suffices to express the properties of completeness and unambiguity as a first-order logic formula F. Once this is done, the decision algorithm can determine if F is true or false.

Completeness says every integer has a representation in L. We can express this as follows:

$$\forall n \; \exists x \; x \in L \; \wedge \; [x]_F = n, \tag{2}$$

Unambiguity says that no integer has two distinct representations in L. We can express this as follows:

$$\neg \exists x, y \in L \; (\neg \text{equal}(x,y)) \; \wedge \; [x]_F = [y]_F. \tag{3}$$

Here equal means that x and y are the same, up to leading zeros. □

Furthermore, if L is a regular language that provides a system that is complete, we can find a representation in L for n efficiently. The first step is to represent n in Fibonacci representation, say using the greedy algorithm. Construct a new automaton from `fcanon` by using two intersections. The first intersection is with an automaton with a first component that belongs to L, while the second component is arbitrary. The second intersection is with an automaton where the first component is arbitrary, and the second is of the form $0^*(n)_F$. This gives a new automaton of $O(\log n)$ states, and it now suffices to find any accepting path (a path from the initial state to the final state). This can be done in linear time in the number of states using depth-first or breadth-first search. This gives us an $O(\log n)$ algorithm to find a representation. Thus we have proved:

Theorem 4. *Suppose L is a regular language. If L is complete, we can find a representation for an integer n in* $O(\log n)$ *time.*

Remark 5. Here we use the convention of the so-called "word RAM" model, where we assume that n fits in a single machine word, or more generally that we can perform basic operations on integers with $O(\log n)$ bits in unit time.

All this can be carried out mechanically with `Walnut`. Here all we have to do is define the language L of valid representations (say, with a regular expression) and type in the `Walnut` commands corresponding to the two logical assertions (2) and (3). We illustrate this with two examples.

The first is the lazy representation mentioned previously, and discussed first by Brown [3]. The first step is to give a regular expression defining a valid representation in Brown's system:

```
reg lazyExclude {0,1} "0*1(0|1)*00(0|1)*":
def lazy "~$lazyExclude(s)":
```

This gives a 4-state automaton testing the lazy criterion that is depicted in Figure 2.

We test the completeness and unambiguity for Brown's system as follows.

```
reg equal {0,1} {0,1} "([0,0]|[1,1])*":
eval brown1 "?msd_fib An Es $fcanon(s,n) & $lazy(s)":
eval brown2 "?msd_fib ~En,s,t $lazy(s) & $lazy(t) & (~$equal(s,t))
    & $fcanon(s,n) & $fcanon(t,n) ":
```

Both return `TRUE`. Given these results, we have now proven that the lazy representation is complete and unambiguous.

For a second example, see the Appendix.

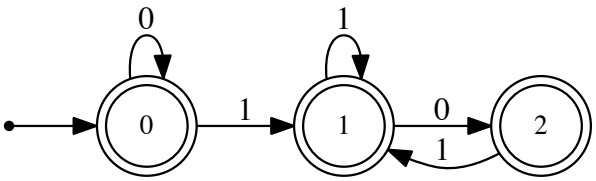


Figure 2: DFA for Brown's lazy representation.

4 Representation using digits -1, 0, and 1

We now turn to representations using digits -1, 0, and 1 in the Fibonacci system.

Recently, Hajnal [9] described three Fibonacci representations using Eq. (1) to associate a string $x = e_t e_{t-1} \cdots e_2 \in \{-1,0,1\}^*$ with a natural number n: *alternating, even, and odd.* Using induction and a case-based argument, he proved that each of these three representations is complete and unambiguous.

Using automata, we can replace his rather long arguments with our general approach. We first describe each of his systems, and show that the set of valid representations for all natural numbers is a regular language.

The alternating representation requires a representation to fulfill four conditions:

1. the most significant nonzero term is positive,
2. two adjacent nonzero terms cannot be of the same sign,
3. two adjacent nonzero terms have at least one zero in between, and
4. if there are two or more nonzero terms, then there has to be at least two zeros between the last and the second last nonzero terms.

We denote a number n in this representation as $[n]_A$. For example, $[9]_A = 10\bar{1}001$, where $\bar{1}$ is used for -1.

For the alternating representation, we can use the following `Walnut` code:

```
reg altInclude1 {-1,0,1} "(0*|0*1.*)":
reg altExclude1 {-1,0,1} ".*(10*1|[-1]0*[-1]).*":
reg altExclude2 {-1,0,1} ".*(1[-1]|[-1]1).*":
reg altInclude2 {-1,0,1} "(0*|0*10*|.*(100+[-1]|[-1]00+1)0*)":
def alt "~$altExclude1(s) & ~$altExclude2(s) & $altInclude1(s) & $altInclude2(s)":
```

The result is an automaton of 12 states that checks whether an input over the alphabet $\{-1,0,1\}$ is alternating, and is illustrated in Figure 3.

The even representation requires three conditions:

1. the most significant nonzero term is positive,
2. only positions indexed with even numbers, such as e_2, can have nonzero terms, and
3. two adjacent nonzero terms cannot both be -1.

We denote a number n in this representation as $[n]_E$. For example, $[14]_E = 10\bar{1}0001$.

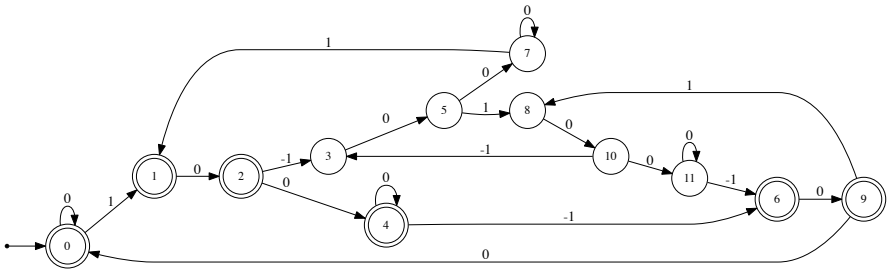


Figure 3: DFA for the alternating condition.

```
reg evenInclude {-1,0,1} "(0*|0*1(0[-1]|01|00)*)":
reg evenExclude {-1,0,1} ".*[-1]0*[-1].*":
def even "$evenInclude(s) & ~$evenExclude(s)":
```

This gives us a 5-state automaton to check the even condition, which is illustrated in Figure 4.

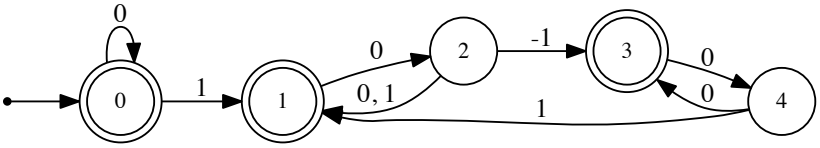


Figure 4: DFA for the even condition.

The odd representation adds an epsilon term to the sum in Eq. (1), therefore associating a string $e_t e_{t-1} \cdots e_2 \varepsilon$, where $\varepsilon \in \{-1, 0\}$, with a number n. The odd representation requires the string to meet three conditions:

1. the most significant nonzero term is positive,
2. only positions indexed with odd numbers (such as e_3) and the epsilon term are allowed to be nonzero, and
3. two adjacent nonzero terms cannot both be -1.

We denote a number n in this representation as $[n]_O$. For example, $[14]_O = 100010\bar{1}$, where $\bar{1}$ is used for $\varepsilon = -1$.

We express the odd representation conditions in `Walnut` as follows. Notice we relax the third condition (required in [9]) slightly by limiting its application to only the string $e_t e_{t-1} \cdots e_2$ without the ε term.

```
reg oddInclude {-1,0,1} "(0*|0*10([-1]0|10|00)*)":
reg oddExclude {-1,0,1} ".*[-1]0*[-1].*":
def odd "$oddInclude(s) & ~$oddExclude(s)":
```

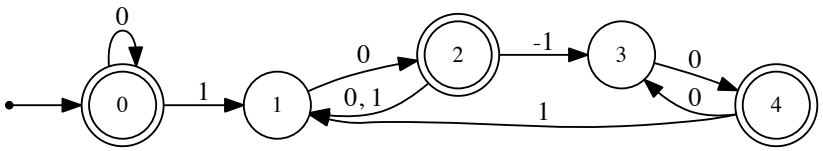


Figure 5: DFA for the odd condition.

This gives us a 5-state automaton to check the odd condition, which is illustrated in Figure 5.

It now remains to use our technique to show that these representations are all complete and unambiguous. In order to do this, we need a "converter" automaton that can compare representations using digits $-1, 0, 1$ to ordinary Zeckendorf representation. We can construct such an automaton based on `fcanon` as follows. The idea is to use one automaton to "select" the positive digits of a representation, another one to "select" the negative digits, and then do an (implicit) subtraction to obtain the value of the representation.

```
reg posdigits {-1,0,1} {0,1} "([1,1]|[-1,0]|[0,0])*":
reg negdigits {-1,0,1} {0,1} "([-1,1]|[1,0]|[0,0])*":
def fcanon2 "?msd_fib Et,u,w,s $negdigits(x,t) & $posdigits(x,u) &
   $fcanon(t,w) & $fcanon(u,s) & z+w=s":
```

This gives a 24-state automaton `fcanon2`, the analogue of `fcanon`, for doing the conversion.

Let us now check that the alternating representation of Hajnal is both complete and unambiguous.

```
reg same {-1,0,1} {-1,0,1} "([-1,-1]|[0,0]|[1,1])*":
eval altRep1 "?msd_fib An Es $fcanon2(s,n) & $alt(s)":
# evaluates to TRUE, 4 ms
eval altRep2 "?msd_fib ~En,s,t $alt(s) & $alt(t) & (~$same(s,t))
    & $fcanon2(s,n) & $fcanon2(t,n)":
# evaluates to TRUE, 31 ms
```

Similarly, we can check the even and odd representations, as follows:

```
eval evenRep1 "?msd_fib An Es $fcanon2(s,n) & $even(s)":
# evaluates to TRUE, 1 ms
eval evenRep2 "?msd_fib ~En,s,t $even(s) & $even(t) & (~$same(s,t))
    & $fcanon2(s,n) & $fcanon2(t,n)":
# evaluates to TRUE, 4 ms
eval oddRep1 "?msd_fib An (Es $fcanon2(s,n) & $odd(s)) |
   (Et $fcanon2(t,n+1) & $odd(t))":
# evaluates to TRUE, 7 ms
eval oddRep2 "~En,s,t $odd(s) & $odd(t) & (~$same(s,t))
    & $fcanon2(s,n) & $fcanon2(t,n)":
# evaluates to TRUE, 4 ms
```

This completes our proof that all three systems of Hajnal are complete and unambiguous.

Remark 6. We noticed, by testing the following, that this representation is also complete if $\varepsilon \in \{1,0\}$ instead of $\varepsilon \in \{-1,0\}$ as required in [9].

```
eval oddRep3 "?msd_fib An
    (Es $fcanon2(s,n) & $odd(s)) | (Et $fcanon2(t,n-1) & $odd(t))":
# evaluates to TRUE, 4 ms
```

5 Representations for all integers

In this section we investigate two different ways to represent *all* integers (not just the natural numbers) using Fibonacci representations.

Alpert [1] described a *far-difference representation* for Fibonacci numbers that writes *every integer* (not just the natural numbers), with a Fibonacci numeration system using the digits $-1,0,1$. In Alpert's system, the far-difference representation requires the string to have

1. at least three zeros between any two nonzero terms of the same sign, and
2. at least two zeros between any two nonzero terms of different signs.

We use $[n]_A$ to denote a natural number in this representation: for example, $[-38]_A = \bar{1}000\bar{1}001$. One nice feature of Alpert's system is that it is very easy to negate an integer: all we have to do is change the sign of each digit.[2]

We express the far-difference representation conditions in `Walnut` as follows.

```
reg exclude1 {-1, 0, 1} ".*([-1][-1]|[-1]0[-1]|[-1]00[-1]|11|101|1001).*":
reg exclude2 {-1, 0, 1} ".*([-1]1|1[-1]|10[-1]|[-1]01).*":
def alpert "~$exclude1(s) & ~$exclude2(s)":
```

This gives a 7-state automaton that checks the Alpert condition, as illustrated in Figure 6.

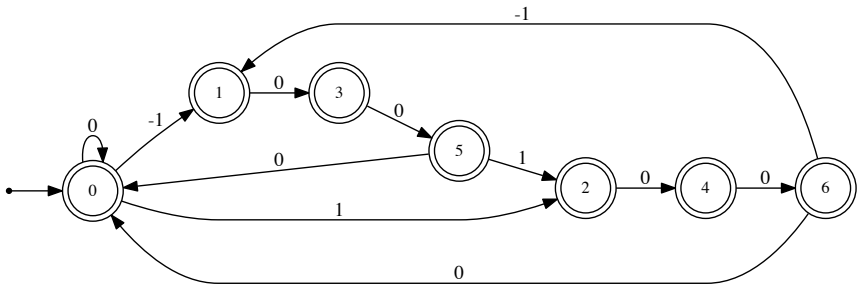


Figure 6: DFA for the Alpert conditions.

To check completeness and ambiguity, we have to check positive and negative integers separately. In addition to `fcanon2`, we need an automaton `fcanon2_neg` that takes a string x over the alphabet $\{-1,0,1\}$ and a natural number $n \geq 0$ as input and accepts if $[x]_F = -n$.

[2]The three systems proposed by Hajnal also exhibit this property. Therefore, if we exclude the condition stating "the most significant nonzero term is positive" from the three systems, they can be perfect representations for all integers.

```
def fcanon2_neg "?msd_fib Et,u,w,s $negdigits(x,t) & $posdigits(x,u) &
   $fcanon(t,w) & $fcanon(u,s) & z+s=w":
```

We can then prove the completeness and unambiguity of this system as follows.

```
eval farDiff1_pos "?msd_fib An Es $fcanon2(s,n) & $alpert(s)":
eval farDiff1_neg "?msd_fib An Es $fcanon2_neg(s,n) & $alpert(s)":
# both evaluate to TRUE, 3 ms
eval farDiff2_pos "?msd_fib ~En,s,t $alpert(s) & $alpert(t)
    & (~$same(s,t)) & $fcanon2(s,n) & $fcanon2(t,n)":
eval farDiff2_neg "?msd_fib ~En,s,t $alpert(s) & $alpert(t)
    & (~$same(s,t)) & $fcanon2_neg(s,n) & $fcanon2_neg(t,n)":
# both evaluate to TRUE, 9 ms
```

Thus we have easily verified the correctness of Alpert's conditions.

Bunder [7] invented a different numeration system for all integers, called the negaFibonacci system. In this system, we write integers as a sum of distinct Fibonacci numbers with negative indices, subject to the condition that no two consecutive Fibonacci numbers can be used. Since $F_{-n} = (-1)^{n+1}F_n$ for $n \geq 1$, this is the same as enforcing the requirement in a Fibonacci representation $a_t F_t + \cdots + a_2 F_2 + a_1 F_1$ with digits $a_i \in \{-1, 0, 1\}$, (a) only the terms with odd indices are allowed to be positive and only the terms with even indices are allowed to be negative and (b) no two consecutive nonzero digits can appear. We can enforce this condition as follows:

```
reg bunder1 {-1,0,1} ".*1.(..)*":
reg bunder2 {-1,0,1} ".*[-1](..)*":
reg bunder3 {-1,0,1} ".*((1[-1])|([-1]1)).*":
def bunder "~$bunder1(x) & ~$bunder2(x) & ~$bunder3(x)":
```

which gives the automaton in Figure 7. We can then check completeness and unambiguity much as

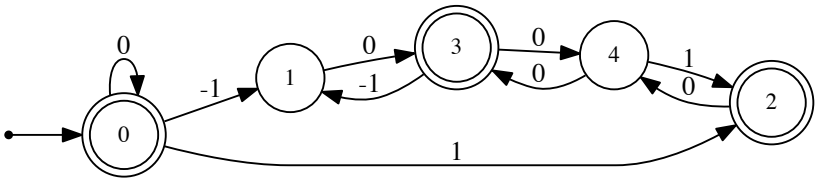


Figure 7: DFA for the Bunder conditions.

we did for Alpert's system, but there is a new wrinkle: representations have an extra digit at the end, corresponding to the term $a_1 F_1$, that must be taken care of. To do this we introduce a "shifter" automaton that shifts a representation to the right, and a "lastbit" that determines if the last bit of a representation is 1 or 0. The shifter is called `rshiftfib` and is displayed in Figure 8.

Then Bunder's representation can be verified to be complete and unambiguous, as follows:

```
reg lastbit {-1,0,1} {0,1} "([0,0]|[1,0]|[-1,0])*([1,1]|[0,0])":
def fcanon3 "?msd_fib Et,u,m $rshiftfib(x,t) &
```

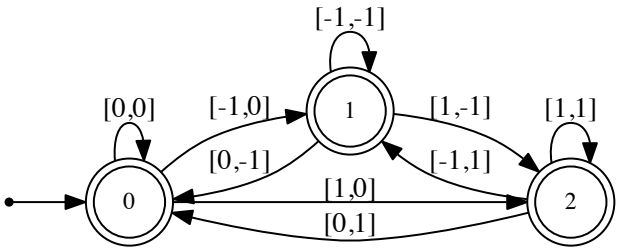


Figure 8: Shifter automaton.

```
   $lastbit(x,u) & $fcanon2(t,m) & z=m+u":
def fcanon3_neg "?msd_fib Et,u,m $rshiftfib(x,t) &
   $lastbit(x,u) & $fcanon2_neg(t,m) & z=m-u":
eval bunder1_pos "?msd_fib An Es $fcanon3(s,n) & $bunder(s)":
eval bunder1_neg "?msd_fib An Es $fcanon3_neg(s,n) & $bunder(s)":
# both evaluate to TRUE, 1 ms
eval bunder2_pos "?msd_fib ~En,s,t $bunder(s) & $bunder(t)
    & (~$same(s,t)) & $fcanon3(s,n) & $fcanon3(t,n)":
eval bunder2_neg "?msd_fib ~En,s,t $bunder(s) & $bunder(t)
    & (~$same(s,t)) & $fcanon3_neg(s,n) & $fcanon3_neg(t,n)":
# both evaluate to TRUE, 12 ms
```

6 Maximum dictionary order representation

In this section we consider an entirely new Fibonacci representation based on dictionary order. We first introduce how strings are compared in dictionary order. Let $s = s_1 s_2 \cdots s_m$ and $t = t_1 t_2 \cdots t_n$ where $m \leq n$ be two strings. Let i such that $1 \leq i \leq m$ be the first position where $s_i \neq t_i$. If $s_i < t_i$, then $s < t$ in dictionary order; otherwise $s > t$. For example, $10\underline{1}11 < 11\underline{0}00$, but $10\underline{1}1 > 10\underline{0}1$. If there is no such position i, then either $s = t$ or s is a proper prefix of t. In this latter case we say $s < t$. For example, $110 = 110$ and $110 < 1100$.

Consider a representation of natural numbers by always choosing the *largest string representation in dictionary order* for every number. Since every number has a Fibonacci-based representation, the representation is complete. Since we choose only one Fibonacci-based representation for each number, the representation is unambiguous. Representations of the first few numbers are given in Table 2.

n	1	2	3	4	5	6	7	8	9	10	11
$(n)_D$	1	10	11	101	110	111	1010	1100	1101	1110	1111

Table 2: Representations for the first few numbers.

We now show that

Theorem 7. *The set of largest Fibonacci representations in dictionary order forms a regular language.*

Proof. The idea is to construct a comparator DFA C_D that can take two representations in parallel and decide if one is greater than the other, in dictionary order.

In order to take two representations in parallel, they would have to be the same length, and therefore the shorter one would have to be padded with leading zeros to make it the same length as the longer one. In this case, it is not hard to see that no automaton can do the needed comparison.

However, in our case, we can take advantage of the following fact: two Fibonacci representations for the same number cannot be of wildly different lengths.

Lemma 8. *The lengths of two Fibonacci-based representation strings for the same natural number differ by one at most (not counting leading zeros).*

Proof. Let s and t be two Fibonacci representations for a natural number m. Without loss of generality, assume that s is longer. Suppose the leading 1 digit of s corresponds to F_i. If s and t differ in length by more than one, then t is a sum of some F_j's where $j \leq i-2$. Now a classic identity on Fibonacci numbers states that $\sum_{0\leq j\leq n} F_j = F_{n+2} - 1$. Using this relation, we conclude that $\sum_{j=2}^{i-2} F_j = F_i - 2 < F_i$. Therefore s and t do not represent the same number. □

Using this fact, it is indeed possible to compare two strings in dictionary order with an automaton. It is shown in Fig. 9 and takes two inputs in parallel, s' and t'. Let s and t be s' and t' without leading

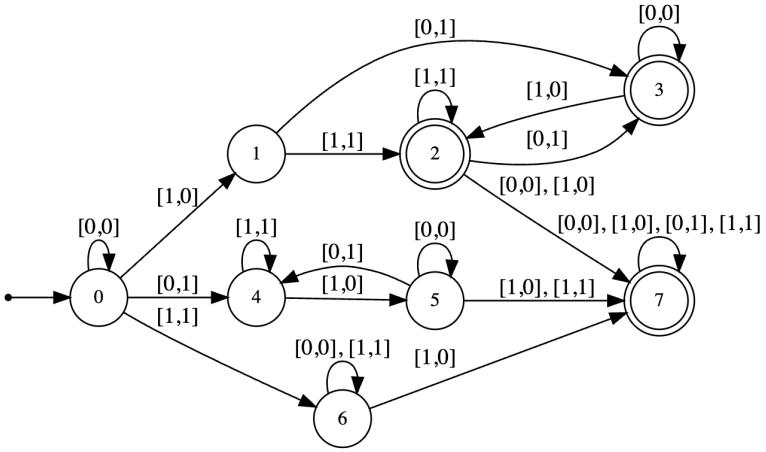


Figure 9: DFA C_D for comparing strings in dictionary order.

zeros. The DFA C_D accepts if and only if s is greater than t in dictionary order. We have three cases to consider: $|s| > |t|$, $|s| < |t|$, and $|s| = |t|$. We now discuss how the 8 states of C_D relate to these 3 cases.

- State 0 is the initial state.
- State 1 is reached if $|s| > |t|$; that is, if s' starts with 1 and t' starts with 01.
- State 2 is reached when $|s| > |t|$, s ends in 1, and based on the inputs so far, t is a proper prefix of s therefore $s > t$.
- State 3 is reached when $|s| > |t|$, s ends in 0, and based on the inputs so far, t is a proper prefix of s therefore $s > t$.
- State 4 is reached when $|s| < |t|$ and t ends in 1, and based on the inputs so far, s is a proper prefix of t therefore $s < t$.

- State 5 is reached when $|s| < |t|$ and t ends in 0, and based on the inputs so far, s is a proper prefix of t therefore $s < t$.
- State 6 is reached when $|s| = |t|$ and, based on the inputs so far, we have $s = t$.
- State 7 is one of the accepting states. It is reached when we can identify a position i such that $s_i > t_i$. Additional symbols read, starting from this state, cannot change the comparison result.

It is now easy to verify that the transitions maintain the invariants corresponding to each state, and we leave this to the reader. □

Using the comparator automaton, we can build a DFA D that finds the maximum dictionary order representation for each natural number. The automaton D takes two inputs in parallel: a number n in Zeckendorf representation and a string $s \in \{0,1\}^*$; and it only accepts if, out of all Fibonacci-based representations of n, the string s is the greatest based on dictionary order. We implement D in `Walnut` as follows.

```
def dictOrder "$fcanon(s,n) & (At $fcanon(t,n) => ($dGreater(s,t)|$equal(s,t)))":
```

Here `dictOrder` implements the automaton D; `fcanon`, the automaton C; and `dGreater`, the automaton C_D. The resulting automaton has 7 states and is depicted in Figure 10.

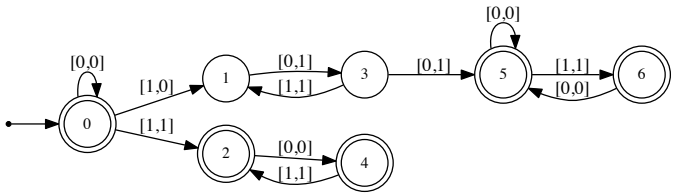


Figure 10: DFA D for converting to dictionary order representation.

7 Finding new perfect systems of small complexity via exhaustive search

We see that a Fibonacci-based representation of natural numbers can be represented by a language over the binary alphabet $\{0,1\}$. If the language is regular, we can express it with a DFA and test its completeness and unambiguity in `Walnut`. For example, the Zeckendorf representation can be expressed as a 3-state DFA and the Brown one, a 4-state DFA. Therefore we were curious about whether there exist other DFAs with a small number of states that can qualify as complete and unambiguous representations. We conducted an exhaustive search to find such automata and found a surprising number of them. If we allow up to 7 states, we found more than 28 new complete and unambiguous representations.[3] We present two interesting examples out of the seven new 6-state representations.

Theorem 9. *Let $L = 0^*(\varepsilon|1|10(\varepsilon|0|1)1^*(01^+)^*(\varepsilon|0))$. Then L is complete and unambiguous.*

Proof. We use the following `Walnut` code:

[3]There could be more as the heuristics we used to trim our search tree can sometimes exclude eligible representations if, for two numbers m,n where $m < n$, the representation of m is longer than that of n.

```
reg one0sq {0,1} "0*(()|1|10(()|0|1)1*(01+)*(()|0))":
eval one0sqTestC "?msd_fib An Ex $one0sq(x) & $fcanon(x,n)":
eval one0sqTestU "?msd_fib ~En,x,y $one0sq(x) & $one0sq(y)
    & (~$equal(x,y)) & $fcanon(x,n) & $fcanon(y,n)":
```

Both returned `TRUE`. Here `one0sq` tests membership in L. □

Notice this representation allows 100 at the very beginning but no other consecutive 0's are allowed. This restriction on 00 blocks is very similar to Brown's. In fact, Brown's can be expressed, in the form of a regular expression, as

$$0^*(\varepsilon|11^*(01^+)^*(\varepsilon|0)) = 0^*(\varepsilon|1|\cancel{10(\varepsilon|0|1)}1^*(01^+)^*(\varepsilon|0)).$$

We can imagine that a new representation could be generated for allowing a block of 00 after the second 1, or the third, or after both the first and third 1, or the first and fourth, etc. This offers another construction of infinitely many perfect representations.

Theorem 10. *Let L be the language accepted by the DFA Z. Then L is complete and unambiguous.*

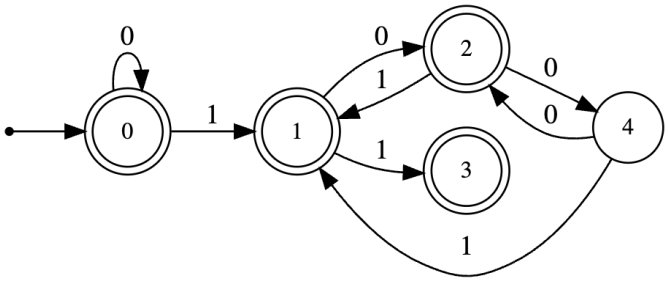


Figure 11: The DFA Z.

Proof. We use the following `Walnut` code:

```
eval azTestC "?msd_fib An Ex $az(x) & $fcanon(x,n)":
eval azTestU "?msd_fib ~En,x,y $az(x) & $az(y) & (~$equal(x,y))
    & $fcanon(x,n) & $fcanon(y,n)":
```

Both returned `TRUE`. Here `az` tests membership in L. □

The strings in L can end with a single 1 or the block 11 or an odd number of 0's, but not an even number of 0's. Additionally, the strings cannot contain the block "11" anywhere but the end. This restriction on "11" is reminiscent of the Zeckendorf representation.

8 Final remarks

The ideas in this paper can be extended in many different ways. For example, we could consider representations in terms of Fibonacci numbers of both positive and negative index with various constraints [16], or representations in terms of sums of the Lucas numbers [4], or other linear recurrences, such as the Pell numbers [11] or Tribonacci numbers [8]. The automaton-based approach can be used in all of these cases.

References

[1] H. Alpert (2009): *Differences of multiple Fibonacci numbers*. *INTEGERS* 9, doi:10.1515/INTEG.2009.061. Paper #A57.

[2] J. Berstel (2001): *An exercise on Fibonacci representations*. *RAIRO Inform. Théor. App.* 35, pp. 491–498, doi:10.1051/ita:2001127.

[3] J. L. Brown, Jr. (1965): *A new characterization of the Fibonacci numbers*. *Fibonacci Quart.* 3(1), pp. 1–8.

[4] J. L. Brown, Jr. (1969): *Unique representation of integers as sums of distinct Lucas numbers*. *Fibonacci Quart.* 7, pp. 243–252.

[5] V. Bruyère, G. Hansel, C. Michaux & R. Villemaire (1994): *Logic and p-recognizable sets of integers*. *Bull. Belgian Math. Soc.* 1, pp. 191–238, doi:10.36045/bbms/1103408547. Corrigendum, *Bull. Belg. Math. Soc.* 1 (1994), p. 577.

[6] J. R. Büchi (1960): *Weak second-order arithmetic and finite automata*. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 6, pp. 66–92, doi:10.1002/malq.19600060105. Reprinted in S. Mac Lane and D. Siefkes, eds., *The Collected Works of J. Richard Büchi*, Springer-Verlag, 1990, pp. 398–424.

[7] M. W. Bunder (1992): *Zeckendorf representations using negative Fibonacci numbers*. *Fibonacci Quart.* 30, pp. 111–115.

[8] L. Carlitz, R. Scoville & V.E. Hoggatt, Jr. (1972): *Fibonacci representations of higher order*. *Fibonacci Quart.* 10, pp. 43–69, 94.

[9] P. Hajnal (2023): *A short note on numeration systems with negative digits allowed*. *Bull. Inst. Combin. Appl.* 97, pp. 54–66.

[10] J. E. Hopcroft & J. D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

[11] A. F. Horadam (1993): *Zeckendorf representations of positive and negative integers by Pell numbers*. In G. E. Bergum, A. N. Philippou & A. F. Horadam, editors: *Applications of Fibonacci Numbers*, 5, Kluwer, pp. 305–316, doi:10.1007/978-94-011-2058-6_29.

[12] C. G. Lekkerkerker (1952): *Voorstelling van natuurlijke getallen door een som van getallen van Fibonacci*. *Simon Stevin* 29, pp. 190–195.

[13] H. Mousavi (2016): *Automatic theorem proving in* `Walnut`. Arxiv preprint arXiv:1603.06017 [cs.FL], available at `http://arxiv.org/abs/1603.06017`.

[14] H. Mousavi, L. Schaeffer & J. Shallit (2016): *Decision Algorithms for Fibonacci-Automatic Words, I: Basic Results*. *RAIRO Inform. Théor. App.* 50, pp. 39–66, doi:10.1051/ita/2016010.

[15] A. Ostrowski (1922): *Bemerkungen zur Theorie der Diophantischen Approximationen*. *Abh. Math. Sem. Hamburg* 1, pp. 77–98,250–251, doi:10.1007/BF02940595. Reprinted in *Collected Mathematical Papers*, Vol. 3, pp. 57–80.

[16] H. Park, B. Cho, D. Cho, Y. D. Cho & J. Park (2020): *Representations of integers as sums of Fibonacci numbers*. *Symmetry* 12(10), doi:10.3390/sym12101625. Paper 1625.

[17] N. Robbins (1996): *Fibonacci partitions*. *Fibonacci Quart.* 34, pp. 306–313.

[18] J. Shallit (2021): *Robbins and Ardila meet Berstel*. *Inform. Process. Lett.* 167, doi:10.1016/j.ipl.2020.106081. Paper 106081.

[19] J. Shallit (2022): *The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with* `Walnut`. *London Math. Soc. Lecture Notes Series* 482, Cambridge University Press, doi:10.1017/9781108775267.

[20] E. Zeckendorf (1972): *Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas*. *Bull. Soc. Roy. Liège* 41, pp. 179–182.

Separating Words from Every Start State with Horner Automata

Nicholas Tran
Santa Clara University
Santa Clara, CA 95053
ntran@scu.edu

We show that a well-known family of deterministic finite automata $H_{b,m}$ can be used to distinguish distinct binary strings of the same length from every start state. Further, we establish a lower bound of $\Omega(\sqrt{n/\log n})$ and an upper bound of $O(\sqrt{n}\log n \log\log n)$ on the number of states of $H_{b,m}$ necessary to achieve this type of separation. Our latter result improves the currently best known $O(n)$ upper bound for arbitrary DFA.

1 Introduction

Given two distinct strings (or words) over some alphabet, we are interested in the minimum size of deterministic finite automata that end in different states after reading the strings *for every (common) start state*. We call this the $\forall$-*separation distance* between strings, and when the alphabet is nonunary, we use $D_\forall(n)$ to denote the largest $\forall$-separation distance between two distinct strings of length n. This variant of the separating words problem was recently introduced and studied in [9], where a lower bound of $\Omega(\log n)$ and an upper bound of $n+1$ on $D_\forall(n)$ were established. The main result of this paper is an improved $O(\sqrt{n}\log n \log\log n)$ upper bound on $D_\forall(n)$.

The original separating words problem was studied in [4, 6, 3, 10, 2]. The standard notion of separation words by deterministic finite automata requires one string to be accepted and the other rejected. It is clearly equivalent to the definition of separation stated initially in [4] that does not involve accepting states: a deterministic finite automaton separates two strings if it ends in different states after reading the strings *for some (common) start state*. We use $D_\exists(n)$ to denote the largest separation distance of this type between two distinct strings of length n for nonunary alphabets. The best known upper bound on $D_\exists(n)$ is $O(n^{1/3}\log^7 n)$ [2], and the best known lower bound on $D_\exists(n)$ is $\Omega(\log n)$ [3]. It is known that the separation distances are tightly bound by $\log n$ for strings of different lengths $m < n$ (in particular, for distinct strings over unary alphabets), and that $D_\exists(n)$ and $D_\forall(n)$ do not depend on the (nonunary) alphabet size. Table 1 lists values of $D_\exists(n)$ and $D_\forall(n)$ for small values of n, obtained via exhaustive search[1].

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$D_\exists(n)$	2	2	2	3	3	3	3	3	3	4	4	4	4	4	4	4	4	5
$D_\forall(n)$	2	2	3	3	3	3	3	4	4	4	4	4	4	4	5	5	5	5

Table 1: Values of $D_\exists(n)$ and $D_\forall(n)$ for $1 \le n \le 18$.

[1]These results were computed using a C++ program running on a Linux workstation with Intel® Core™ i7-4790S CPU @ 3.20GHz and 16 GB RAM.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 243–252, doi:10.4204/EPTCS.386.19

If the separating automaton is required to end in different states for *every pair* of start states, then the largest so-called $\forall^2$-separation distance between two distinct strings of length n is exactly $n+1$ and unbounded between strings of different lengths (i.e., they may not be separable). At the other extreme, if the separating automaton is required to end in different states for *some pair* of start states, then the largest so-called $\exists^2$-separation distance between two distinct strings (regardless of lengths) is 2. These and other results can be found in [9].

The best lower bound of $\Omega(\log n)$ on $D_\exists(n)$ applies trivially to $D_\forall(n)$; it is not known if a stronger lower bound holds for the latter. On the other hand, the best upper bound of $O(n^{1/3}\log^7 n)$ on $D_\exists(n)$ does not readily apply to $D_\forall(n)$. In fact, it is not immediately clear how to establish even a looser upper bound such as $n+1$. In this paper we show how to adapt the technique of counting the number of cyclotomic factors of certain polynomials [10, 2] to obtain an $O(\sqrt{n}\log n\log\log n)$ upper bound on $D_\forall(n)$. Our result improves the currently best known $O(n)$ upper bound obtained in [9].

Specifically, we interpret each nonempty binary string s as the representation of an integer $N_{s,b}$ in some base b and show how to compute $N_{s,b} \bmod m$ for some modulus m using a familiar deterministic finite automaton $H_{b,m}$ with m states and dependent only on b and m. We then show that if s and t are distinct binary strings of length $n \geq 1$, then $N_{b,s} \not\equiv N_{b,t} \pmod{m}$ for some $0 \leq b < m \in O(\sqrt{n}\log n\log\log n)$. Hence $H_{b,m}$ $\exists$-separates s and t, and in fact we show that $H_{b,m}$ $\forall$-separates s and t. On the other hand, we show that for every $n \geq 1$, there are distinct binary strings s and t of length n such that the smallest $H_{b,m}$ that $\forall$-separates s and t have $\Omega(\sqrt{n/\log n})$ states.

The rest of this paper is organized as follows. Section 2 reviews basic definitions about separating words with automata and states some useful number-theoretic facts. The next section contains the main results: Subsection 3.1 presents so-called Horner automata $H_{b,m}$ and shows how to use them to $\forall$-separate strings, Subsection 3.2 proves an $\Omega(\sqrt{n/\log n})$ lower bound on the size of $H_{b,m}$ required to $\forall$-separate two distinct binary strings of length n, and Subsection 3.3 establishes an $O(\sqrt{n}\log n\log\log n)$ corresponding upper bound. Section 4 discusses ideas for future work.

2 Preliminaries

The symbols of a string s of length $n \geq 1$ from left to right are denoted by $s_0, s_1, \ldots, s_{n-1}$. The natural and binary logarithms are denoted by ln and log respectively. We use the following simplified definition of deterministic finite automata that does not specify an initial state or accepting states:

Definition 1. A *deterministic finite automaton* (DFA) is a triple

$$M = (Q, \Sigma, \delta),$$

where Q is a finite set of *states*, Σ is an alphabet, and $\delta : Q \times \Sigma \to Q$ is a *transition* function. We use $|M|$ to denote the number of states of M and refer to it as the *size* of M.

The *extended transition function* $\delta' : Q \times \Sigma^* \to Q$ is defined recursively:

1. $\delta'(q, \varepsilon) = q$, where ε is the empty string, for $q \in Q$;
2. $\delta'(q, xa) = \delta(\delta'(q, x), a)$ for $a \in \Sigma$, $x \in \Sigma^*$ and $q \in Q$.

The first and last states in the sequence of states that M enters when reading a string from left to right are called the *start* and *end* state respectively.

Definition 2. Let x and y be strings over an alphabet Σ. We say a DFA M

- $\exists$-*separates* x and y if $\delta'(s, x) \neq \delta'(s, y)$ for *some* $s \in Q$;

- $\forall$-*separates* x and y if $\delta'(s,x) \neq \delta'(s,y)$ for *every* $s \in Q$.

Example 1. Let $s = 0000\ 0000$ and $t = 1111\ 1100$.

The two-state DFA in Fig. 1 (left) $\exists$-separates s and t because $a = \delta'(a,s) \neq \delta'(a,t) = b$, and it is clearly a smallest such automaton. On the other hand, this DFA does *not* $\forall$-separate s and t because $\delta'(b,s) = \delta'(b,t) = b$.

The four-state DFA in Fig. 1 (right) $\forall$-separates s and t because

$$\begin{aligned} a = \delta'(a,s) &\neq \delta'(a,t) = b, \\ b = \delta'(b,s) &\neq \delta'(b,t) = a, \\ c = \delta'(c,s) &\neq \delta'(c,t) = a, \\ d = \delta'(d,s) &\neq \delta'(d,t) = a. \end{aligned}$$

It is shown to be a smallest such automaton in [9].

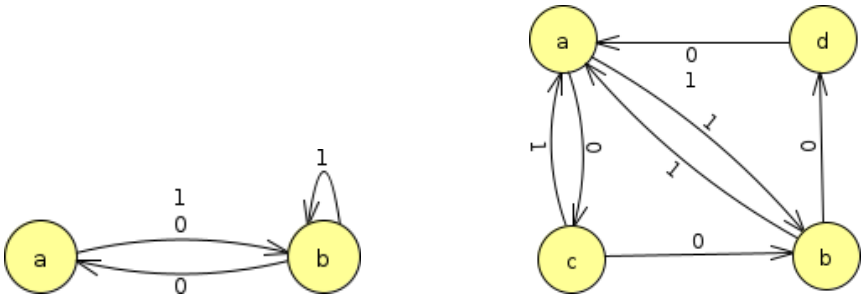


Figure 1: The DFA on the left $\exists$-separates 0000 0000 and 1111 1100 but does not $\forall$-separate them. The DFA on the right $\forall$-separates the strings.

The following facts can be found in standard texts on number theory, e.g., [5]. The **Prime Number Theorem** states that $\pi(x)$, the number of primes less than or equal to x, is approximately $x/\ln x$, i.e.,

$$\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\ln x}} = 1.$$

Bertrand's postulate states that for every $n > 1$, there is a prime p such that $n < p < 2n$. For every integer $n \geq 1$, the n^{th} **cyclotomic polynomial** $\Phi_n(x)$ is defined as the polynomial whose zeros are the *primitive* n^{th} roots of unity:

$$\Phi_n(x) = \prod_{\substack{1\leq k\leq n \\ \gcd(k,n)=1}} (x - e^{2\pi ik/n}).$$

We list relevant properties of cyclotomic polynomials here. The coefficients of $\Phi_n(x)$ are integers, and its degree is $\phi(n)$, the **Euler's totient function**; it is known that $\phi(n) \in \Omega(n/\log\log n)$ [7] and that the function $x/\log\log x$ is increasing when $x \geq 6$ (e.g., Wolfram|Alpha). The cyclotomic polynomials are irreducible and co-prime over $\mathbb{Q}$, i.e., $\gcd(\Phi_n(x), \Phi_m(x)) = 1$ for $n \neq m$. Finally, $\Phi_n(x)$ divides $x^n - 1$ for all $n \geq 1$.

Example 2. The properties listed above can be seen to hold for the first few cyclotomic polynomials:

$$\begin{aligned}
\Phi_1(x) &= x-1,\\
\Phi_2(x) &= x+1,\\
\Phi_3(x) &= x^2+x+1,\\
\Phi_4(x) &= x^2+1,\\
\Phi_5(x) &= x^4+x^3+x^2+x+1,\\
\Phi_6(x) &= x^2-x+1.
\end{aligned}$$

The set of integers modulo m is denoted $\mathbb{Z}_m$, and the set of polynomials in x with integer coefficients is denoted $\mathbb{Z}[X]$. A polynomial $P(x) \in \mathbb{Z}[X]$ is said to **vanish** modulo m if $P(x) \equiv 0 \pmod{m}$ for all $x \in \mathbb{Z}$. **Lagrange's theorem** states that if $P(x) \in \mathbb{Z}[X]$ has degree $n \geq 1$ and p is a prime, then either $P(x)$ has at most n zeros modulo p, or all coefficients of $P(x)$ are divisible by p. By **Fermat's little theorem**, $x^p - x$ vanishes modulo p when p is prime.

We associate with each *binary* string s of length $n \geq 1$ the polynomial

$$s(x) = \sum_{j=0}^{n-1} s_j x^{n-1-j}$$

with coefficients in $\mathbb{Z}_2$. If s and t are distinct binary strings of length $n \geq 1$, $s(x) - t(x)$ is a nonzero polynomial of degree at most $n-1$ with coefficients in $\{-1, 0, 1\}$.

Example 3. Again, let $s = 0000\ 0000$ and $t = 1111\ 1100$. The associated polynomials are $s(x) = 0$ and $t(x) = x^7+x^6+x^5+x^4+x^3+x^2$. Their difference $s(x) - t(x)$ is $-(x^7+x^6+x^5+x^4+x^3+x^2)$.

This difference polynomial vanishes modulo 2 because

$$\begin{aligned}
-(0^7+0^6+0^5+0^4+0^3+0^2) = 0 &\equiv 0 \pmod{2},\\
-(1^7+1^6+1^5+1^4+1^3+1^2) = -6 &\equiv 0 \pmod{2}.
\end{aligned}$$

Similarly, it also vanishes modulo 3 because

$$\begin{aligned}
-(0^7+0^6+0^5+0^4+0^3+0^2) = 0 &\equiv 0 \pmod{3},\\
-(1^7+1^6+1^5+1^4+1^3+1^2) = -6 &\equiv 0 \pmod{3},\\
-(2^7+2^6+2^5+2^4+2^3+2^2) = -252 &\equiv 0 \pmod{3}.
\end{aligned}$$

However, $s(x) - t(x)$ does not vanish modulo 5 because

$$-(1^7+1^6+1^5+1^4+1^3+1^2) = -6 \not\equiv 0 \pmod{5}.$$

We will see in the next section that the above observation can be deduced by noting that

$$x^5 - x = x(x-1)(x+1)(x^2+1)$$

does not divide $s(x) - t(x)$ over the rationals:

$$-(x^7+x^6+x^5+x^4+x^3+x^2) = -x^2(x+1)(x^2-x-1)(x^2+x+1).$$

3 Main Results

In this section we introduce Horner automata and show how to use them to $\forall$-separate strings. We then establish almost matching lower bound and upper bound on the size of the smallest Horner automata that $\forall$-separate two distinct binary strings of length n.

3.1 Horner automata and $\forall$-separation

For $0 \leq b < m$, let $H_{b,m}$ be the deterministic finite automaton with m states 0, 1, ..., $m-1$ and transition function $\delta(i,a) = (ib+a) \bmod m$ for $a \in \{0,1\}$; note that the input alphabet is binary. The special cases $H_{2,m}$ are usually introduced in an introductory course on automata theory to recognize binary strings representing integers divisible by m. They are named **Horner automata** in [8], because on binary input s and start state 0, they compute the value of the associated polynomial $s(b)$ using Horner's rule and end in state $s(b) \bmod m$:

$$\delta'(0,s) = ((\cdots((0b+s_0)b+s_1)b+\cdots+s_{n-2})b+s_{n-1}) \bmod m = s(b) \bmod m.$$

In other words, these automata "compute" $N_{s,b} \bmod m$, where $N_{s,b}$ is the integer represented by *binary* string s in base b (but they are not the smallest ones to do so [1]). We prove a slightly stronger statement of this fact in the following lemma.

Lemma 1. *Let $0 \leq b, i < m$ be integers and s be a binary string of length $n \geq 1$. Starting in state i, the Horner automaton $H_{b,m}$ ends in state $(ib^n + s(b)) \bmod m$ after reading s.*

Proof. By induction on n. When $n = 1$, the string s is just symbol s_0, and the associated polynomial $s(x)$ is the constant polynomial s_0. Starting in state i, $M_{b,m}$ ends in state

$$\delta'(i,s_0) = (ib+s_0) \bmod m = (ib^1 + s(b)) \bmod m,$$

after reading s, so the lemma holds.

Assume that the lemma holds for n and let $s = s_0 s_1 \ldots s_n$ be a binary string of length $n+1$. Starting in state i on input s, the automaton $H_{b,m}$ ends in state

$$\begin{aligned}
\delta'(i,s) &= \delta'(i, s_0 \ldots s_{n-1} s_n) \\
&= \delta(\delta'(i, s_0 \ldots s_{n-1}), s_n) \\
&= \delta((ib^n + \sum_{j=0}^{n-1} s_j b^{n-1-j}) \bmod m, s_n) \\
&= (ib^{n+1} + \sum_{j=0}^{n-1} (s_j b^{n-j}) + s_n) \bmod m \\
&= (ib^{n+1} + \sum_{j=0}^{n} s_j b^{n-j}) \bmod m \\
&= (ib^{n+1} + s(b)) \bmod m.
\end{aligned}$$

□

Example 4. Fig. 2 shows the automaton $H_{2,5}$ on the left. On input $s = 10\ 1111$ and start state 0, it ends in state $s(2) \bmod 5 = 2^5+2^3+2^2+2^1+2^0 = 47 \bmod 5 = 2$. In contrast, on input $t = 11\ 1011$ and start state 0, it ends in state $t(2) \bmod 5 = 2^5+2^4+2^3+2^1+2^0 = 59 \bmod 5 = 4$. Thus, $H_{2,5}$ $\exists$-separates s and t, and in fact, it $\forall$-separates s and t due to Lemma 1. We demonstrate in general this important property of Horner automata below.

Figure 2: Horner automata $H_{2,5}$ (left) and $H_{1,2}$ (right)

It is natural to study the separation distance between two binary strings of the same length by Horner automata. We define this notion formally and study its properties here.

Definition 3.

1. The Horner distance between two distinct binary strings s and t, denoted by $d_H(s,t)$, is the smallest m such that $H_{b,m}$ $\exists$-separates s and t for some $0 \le b < m$.
2. The Horner separation distance, denoted by $D_H(n)$, is the maximum Horner distance over all pairs of distinct binary strings s and t of length n, for $n \ge 1$.

We now show that the Horner distance between two distinct binary strings s and t of the same length is always defined, and furthermore, it is an upper bound of the $\forall$-distance between s and t.

Lemma 2. *Let s and t be binary strings of length n. The following are equivalent:*

1. *s and t are distinct;*
2. *s and t are $\exists$-separated by $H_{b,m}$ for some $0 \le b < m \le 2n$;*
3. *s and t are $\forall$-separated by $H_{b,m}$ for some $0 \le b < m \le 2n$.*

Proof. (1) $\Rightarrow$ (2): Let s and t be two distinct binary strings of length n. When $n = 1$, there is only one pair of distinct strings 0 and 1, and they are $\exists$-separated by $H_{1,2}$ (shown in Fig. 2 on the right) when started in state 0. For $n > 1$, by Bertrand's postulate, there is a prime p such that $n < p < 2n$. The polynomial $d(x) = s(x) - t(x)$ has degree at most $n-1$, coefficients in $\{-1,0,1\}$, and not all coefficients are zero, because s and t are distinct. Thus, by Lagrange's theorem $d(x)$ has at most $n-1$ zeros in $\mathbb{Z}_p$, so there exists $0 \le b < p$ such that $d(b) \not\equiv 0 \pmod{p}$. Since $d(b) = s(b) - t(b)$, we have $s(b) - t(b) \not\equiv 0 \pmod{p}$, or $s(b) \not\equiv t(b) \pmod{p}$. Thus, $H_{b,p}$ $\exists$-separates s and t on start state 0.

(2) $\Rightarrow$ (3): Suppose $H_{b,m}$ separates binary strings s and t of length n on start state i_0 for some $0 \le b, i_0 < m \le 2n$. By Lemma 1, $i_0 b^n + s(b) \not\equiv i_0 b^n + t(b) \pmod{m}$, so $ib^n + s(b) \not\equiv ib^n + t(b) \pmod{m}$ for all $0 \le i < m$. Thus, s and t are $\forall$-separated by $H_{b,m}$.

(3) $\Rightarrow$ (1): Suppose s and t are binary strings of length n that are $\forall$-separated by $H_{b,m}$ for some $0 \le b < m \le 2n$. Then $s(b) \not\equiv t(b) \pmod{m}$, so $s(x) \ne t(x)$, and hence $s \ne t$. $\square$

3.2 Lower bound on the Horner separation distance $D_H(n)$

We use a simple information-theoretic argument to show that the Horner separation distance $D_H(n)$ is at least $\Omega(\sqrt{n/\log n})$.

Theorem 1. *$D_H(n) \in \Omega(\sqrt{n/\log n})$ for $n \geq 1$.*

Proof. Let s and t be distinct binary strings of length n. If they cannot be $\exists$-separated by a Horner automaton of size M or less, the values of their polynomials $s(x)$ and $t(x)$ must be congruent modulo m for all $0 \leq b < m$ and $2 \leq m \leq M$. The congruence classes of these values can be encoded in a binary string (called a *signature*) of length at most

$$\sum_{m=2}^{M}\sum_{b=0}^{m-1}(\log m) \leq \sum_{m=2}^{M}\sum_{b=0}^{m-1}(\log M) \in O(M^2 \log M) \in O(M^2 \log n)$$

due to the upper bound on M in terms of n given by Lemma 2. There are $2^{O(M^2 \log n)}$ such signatures and 2^n binary strings of length n, so to ensure that different s and t have different signatures and $\exists$-separable by a Horner automaton of size $D_H(n)$, we must have $2^n \leq 2^{O(D_H^2(n)\log n)}$, or $n \in O(D_H^2(n)\log n)$, and hence $D_H(n) \in \Omega(\sqrt{n/\log n})$. □

3.3 Upper bound on the Horner separation distance $D_H(n)$

We begin with an observation that for large enough prime p, a polynomial $P(x)$ whose coefficients are in $\{-1,0,1\}$ vanishes modulo p if and only it is divisible by $x^p - x$.

Lemma 3. *Let $P(x) = \sum_{j=0}^{n} p_j x^j$ be a polynomial of degree $n \geq 0$ with coefficients $p_j \in \{-1,0,1\}$, and let $p \geq 1 + \sqrt{n}$ be a prime. The polynomial $P(x)$ vanishes modulo p if and only if $P(x) = (x^p - x)Q(x)$ for some $Q(x) \in \mathbb{Z}[x]$.*

Proof. Suppose $P(x) = (x^p - x)Q(x)$ for some polynomial $Q(x) \in \mathbb{Z}[x]$. By Fermat's little theorem, $x^p - x$ vanishes modulo p, so $P(x)$ also vanishes modulo p.

Conversely, suppose $P(x)$ vanishes modulo p. Since $P(0) = p_0 \equiv 0 \pmod{p}$, and $p_0 \in \{-1,0,-1\}$, we have $p_0 = 0$. The division algorithm for integral polynomials says that

$$P(x) = (x^p - x)Q(x) + R(x)$$

for some $Q(x) = \sum_{j=0}^{n-p} q_j x^j$, $R(x) = \sum_{j=0}^{p-1} r_j x^j \in \mathbb{Z}[x]$, where the degree of $R(x)$ is less than p. Since both $P(x)$ and $x^p - x$ vanish modulo p, so does $R(x)$. Because $R(x)$ has p zeros modulo p but its degree is less than p, by Lagrange's theorem, all coefficients of $R(x)$ must be divisible by p. We now show that the coefficients of $R(x)$ have absolute values at most $p-1$, and hence must all be zeros, i.e., $R(x) = 0$.

Expanding term by term both sides of $P(x) = (x^p - x)Q(x) + R(x)$, we have

$$\begin{aligned}\sum_{j=0}^{n} p_j x^j &= \sum_{j=0}^{n-p} q_j x^{j+p} - \sum_{j=0}^{n-p} q_j x^{j+1} + \sum_{j=0}^{p-1} r_j x^j \\ &= \sum_{j=n-p+2}^{n} q_{j-p} x^j + \sum_{j=p}^{n-p+1} (q_{j-p} - q_{j-1}) x^j + \sum_{j=1}^{p-1} (r_j - q_{j-1}) x^j + r_0.\end{aligned}$$

Comparing coefficients on the left and right sides, we see that the constant coefficient of $R(x)$ is 0, and the absolute values of the leftmost $p-1$ coefficients of $Q(x)$ are bounded by 1, because

$$\begin{aligned} r_0 &= p_0 = 0 \\ |q_{j-p}| &= |p_j| \leq 1, \qquad n-p+2 \leq j \leq n \end{aligned}$$

Similarly, we see that the next leftmost $p-1$ coefficients of $Q(x)$ have absolute values bounded by 2, because their differences with the first leftmost $p-1$ coefficients are bounded in absolute value by 1, as can be seen below after changing the range of j:

$$\begin{aligned} |q_{j-p} - q_{j-1}| &\leq 1, \qquad n-2p+3 \leq j \leq n-p+1 \\ |q_{j-2p+1} - q_{j-p}| &\leq 1, \qquad n-p+2 \leq j \leq n \end{aligned}$$

Repeating this argument, we conclude that the next leftmost $p-1$ coefficients of $Q(x)$ have absolute values bounded by 3, and so on. Since there are $\lceil \frac{n}{p-1} \rceil$ such groups (the constant coefficient is zero), we conclude that the absolute values of coefficients of $R(x)$ are bounded by $\lceil \frac{n}{p-1} \rceil$ and hence by $p-1$ because

$$\begin{aligned} \left\lceil \frac{n}{p-1} \right\rceil &\leq \left\lceil \frac{n}{1+\sqrt{n-1}} \right\rceil \\ &= \lceil \sqrt{n} \rceil \\ &< 1+\sqrt{n} \\ &\leq p. \end{aligned}$$

□

Example 5. Let $s = 0000\ 1010\ 0000\ 0000$ and $t = 0000\ 0000\ 0010\ 1000$. The associated polynomials are $s(x) = x^{11} + x^9$ and $t(x) = x^5 + x^3$. The polynomial $P(x) = s(x) - t(x) = x^{11} + x^9 - x^5 - x^3$ has the following irreducible factors over the rationals:

$$P(x) = x^{11} + x^9 - x^5 - x^3 \quad = \quad x^3(x-1)(x+1)(x^2+1)(x^2-x+1)(x^2+x+1).$$

Primes 5, 7, 11 are all greater than $1+\sqrt{11}$. Since

$$\begin{aligned} x^5 - x = x(x-1)(x+1)(x^2+1) \quad &| \quad P(x), \\ x^7 - x = x(x-1)(x+1)(x^2-x+1)(x^2+x+1) \quad &| \quad P(x), \\ x^{11} - x = x(x-1)(x+1)(x^4-x^3+x^2-x+1)(x^4+x^3+x^2+x+1) \quad &\nmid \quad P(x), \end{aligned}$$

it follows from Lemma 3 that $P(x)$ vanishes modulo 5 and 7, but not modulo 11. Exhaustive search shows that $d_H(s,t) = 9$ while $d_\forall(s,t) = 4$.

We are now ready to prove an $O(\sqrt{n}\log n \log\log n)$ upper bound on $D_H(n)$ using Lemma 3. Let s and t be distinct binary strings of length n, and let $P(x) = s(x) - t(x)$, whose degree is at most $n-1$. If $P(x)$ vanishes modulo p for some $p \geq 1+\sqrt{n-1}$, then $P(x)$ is divisible by $x^p - x$, which is in turn divisible by $\Phi_{p-1}(x)$. Since these cyclotomic polynomials are co-prime, the sum δ of their degrees is bounded above by $n-1$, which implies the stated upper bound on $d_H(s,t)$.

Theorem 2. $D_H(n) \in O(\sqrt{n}\log n \log\log n)$.

Proof. Let s and t be distinct binary strings of length n, and suppose the Horner distance $d_H(s,t)$ is $M\sqrt{n}$, so that $P(x) = s(x) - t(x)$, whose degree is at most $n-1$, is congruent to the zero polynomial mod p for all primes $p < M\sqrt{n}$.

By Lemma 2, $M\sqrt{n} \leq 2n$, so $M \leq 2\sqrt{n}$. By Lemma 3, for each such prime $p > \sqrt{n}$, $P(x)$ is divisible by $x^p - x = x(x^{p-1}-1)$ and hence divisible by the cyclotomic polynomial $\Phi_{p-1}(x)$, whose degree is $\phi(p-1) \in \Omega((p-1)/\log\log(p-1))$. Since these cyclotomic polynomials are co-prime, the sum δ of their degrees is at most $n-1$. There are approximately

$$\frac{M\sqrt{n}}{\ln(M\sqrt{n})} - \frac{\sqrt{n}}{\ln\sqrt{n}} > \frac{M\sqrt{n}}{\ln(2\sqrt{n}\sqrt{n})} - \frac{\sqrt{n}}{\ln\sqrt{n}} \geq \alpha\frac{M\sqrt{n}}{\log n}$$

primes p in the range $[\sqrt{n}, M\sqrt{n}]$ by the Prime Number Theorem, for some constant α. Because the function $x/\log\log x$ is eventually increasing, each $\Phi_{p-1}(x)$ contributes at least $\beta\sqrt{n}/\log\log(\sqrt{n})$, for some constant β, to the degree sum δ, which is at most $n-1$, so

$$\left(\frac{\alpha M\sqrt{n}}{\log n}\right)\left(\frac{\beta\sqrt{n}}{\log\log n}\right) = \left(\frac{\alpha\beta Mn}{\log n\log\log n}\right) \leq \delta < n.$$

It follows that $M \in O(\log n\log\log n)$, and hence $d_H(s,t) \in O(\sqrt{n}\log n\log\log n)$. Since this holds for arbitrary pairs of distinct binary strings of length n, we conclude that $D_H(n) \in O(\sqrt{n}\log n\log\log n)$. □

Because the more restrictive separation distance $D_H(n)$ is an upper bound on $D_\forall(n)$, we obtain our main result.

Theorem 3. $D_\forall(n) \in O(\sqrt{n}\log n\log\log n)$.

4 Conclusion

We show how to use Horner automata $H_{b,m}$ to $\forall$-separate two distinct binary strings of length n and establish almost matching lower and upper bounds on the minimum value of such m. Closing the gap between these two bounds is an interesting open problem, as is the question of whether other families of automata can be designed to achieve better lower and upper bounds on the $\forall$-separation distance.

Acknowledgments

Detailed comments and suggestions by the anonymous referees help improve the presentation of this paper.

References

[1] B. Alexeev (2004): *Minimal DFA for Testing Divisibility*. *Journal of Computer and System Sciences* 69(2), p. 235–243, doi:10.1016/j.jcss.2004.02.001.

[2] Z. Chase (2021): *Separating Words and Trace Reconstruction*. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pp. 21–31, doi:10.1145/3406325.3451118.

[3] E. D. Demaine, S. Eisenstat, J. Shallit & D. A. Wilson (2011): *Remarks on Separating Words*. In M. Holzer, M. Kutrib & G. Pighizzini, editors: *Proceedings of the 13th International Workshop on Descriptional Complexity of Formal Systems (DCFS)*, *Lecture Notes in Computer Science* 6808, Springer, Berlin, Heidelberg, pp. 147–157, doi:10.1007/978-3-642-22600-7.

[4] P. Goralčík & V. Koubek (1986): *On Discerning Words by Automata*. In L. Kott, editor: *Proceedings of the 13th International Colloquium on Automata, Languages and Programming (ICALP)*, *Lecture Notes in Computer Science* 226, Springer, Berlin, Heidelberg, pp. 116–122, doi:10.1007/3-540-16761-7_61.

[5] I. Niven, H. S. Zuckerman & H. L. Montgomery (1991): *An Introduction to the Theory of Numbers*. Wiley.

[6] J. M. Robson (1989): *Separating strings with small automata*. *Information Processing Letters* 30(4), pp. 209–214, doi:10.1016/0020-0190(89)90215-9.

[7] J. B. Rosser & L. Schoenfeld (1962): *Approximate formulas for some functions of prime numbers*. *Illinois Journal of Mathematics* 6(1), pp. 64–94, doi:10.1215/ijm/1255631807.

[8] K. Sutner (2009): *Divisibility and State Complexity*. *The Mathematica Journal* 11(3), pp. 430–445, doi:10.3888/tmj.11.3-8.

[9] N. Tran (2022): *Variations of the Separating Words Problem*. In P. Caron & L. Mignot, editors: *Proceedings of the 26th International Conference on Implementation and Application of Automata (CIAA)*, *Lecture Notes in Computer Science* 13266, Springer, Berlin, Heidelberg, pp. 165–176, doi:10.1007/978-3-031-07469-1_13.

[10] M. N. Vyalyi & R. A. Gimadeev (2014): *Separating words by occurrences of subwords*. *Journal of Applied and Industrial Mathematics* 8(2), pp. 293–299, doi:10.1134/S1990478914020161.

Strictly Locally Testable and Resources Restricted Control Languages in Tree-Controlled Grammars

Bianca Truthe
Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
bianca.truthe@informatik.uni-giessen.de

Tree-controlled grammars are context-free grammars where the derivation process is controlled in such a way that every word on a level of the derivation tree must belong to a certain control language. We investigate the generative capacity of such tree-controlled grammars where the control languages are special regular sets, especially strictly locally testable languages or languages restricted by resources of the generation (number of non-terminal symbols or production rules) or acceptance (number of states). Furthermore, the set theoretic inclusion relations of these subregular language families themselves are studied.

1 Introduction

In the monograph [5] by Jürgen Dassow and Gheorghe Păun, *Seven Circumstances Where Context-Free Grammars Are Not Enough* are presented. A possibility to enlarge the generative power of context-free grammars is to introduce some regulation mechanism which controls the derivation in a context-free grammar. In some cases, regular languages are used for such a regulation. They are rather easy to handle and, used as control, they often lead to context-sensitive or even recursively enumerable languages while the core grammar is only context-free.

One such control mechanism was introduced by Karel Čulik II and Hermann A. Maurer in [16] where the structure of derivation trees of context-free grammars is restricted by the requirement that the words of all levels of the derivation tree must belong to a given regular (control) language. This model is called tree-controlled grammar.

Gheorghe Păun proved that the generative capacity of such grammars coincides with that of context-sensitive grammars (if no erasing rules are used) or arbitrary phrase structure grammars (if erasing rules are used). Thus, the question arose to what extend the restrictions can be weakened in order to obtain 'useful' families of languages which are located somewhere between the classes of context-free and context-sensitive languages.

In [6, 7, 8, 9, 27, 29, 30], many subregular families of languages have been investigated as classes for the control languages. In this paper, we continue this research with further subregular language families, especially strictly locally testable languages or languages restricted by resources of the generation (number of non-terminal symbols or production rules) or acceptance (number of states). Furthermore, the set theoretic inclusion relations of these subregular language families themselves are studied.

2 Preliminaries

Throughout the paper, we assume that the reader is familiar with the basic concepts of the theory of automata and formal languages. For details, we refer to [23]. Here we only recall some notation and the definition of contextual grammars with selection which form the central notion of the paper.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 253–268, doi:10.4204/EPTCS.386.20

2.1 Languages, grammars, automata

Given an alphabet V, we denote by V^* and V^+ the set of all words and the set of all non-empty words over V, respectively. The empty word is denoted by λ. By V^k, and $V^{\leq k}$ for some natural number k, we denote the set of all words of the alphabet V with exactly k letters and the set of all words over V with at most k letters, respectively. For a word w and a letter a, we denote the length of w by $|w|$ and the number of occurrences of the letter a in the word w by $|w|_a$. For a set A, we denote its cardinality by $|A|$.

A right-linear grammar is a quadruple $G = (N,T,P,S)$ where N is a finite set of non-terminal symbols, T is a finite set of terminal symbols, P is a finite set of production rules of the form $A \to wB$ or $A \to w$ with $A,B \in N$ and $w \in T^*$, and $S \in N$ is the start symbol. Such a grammar is called regular, if all the rules are of the form $A \to xB$ or $A \to x$ with $A,B \in N$ and $x \in T$ or $S \to \lambda$. The language generated by a right-linear or regular grammar is the set of all words over the terminal alphabet which are obtained from the start symbol S by a successive replacement of the non-terminal symbols according to the rules in the set P. A non-terminal symbol A is replaced by the right-hand side w of a rule $A \to w \in P$ in order to derive the next sentential form. The language generated consists of all sentential forms without a non-terminal symbol. Every language generated by a right-linear grammar can also be generated by a regular grammar.

A deterministic finite automaton is a quintuple $A = (V,Z,z_0,F,\delta)$ where V is a finite set of input symbols, Z is a finite set of states, $z_0 \in Z$ is the initial state, $F \subseteq Z$ is a set of accepting states, and δ is a transition function $\delta : Z \times V \to Z$. The language accepted by such an automaton is the set of all input words over the alphabet V which lead letterwise by the transition function from the initial state to an accepting state.

A regular expression over an alphabet V is defined inductively as follows:

1. $\emptyset$ is a regular expression;
2. every element $x \in V$ is a regular expression;
3. if R and S are regular expressions, so are the concatenation $R \cdot S$, the union $R \cup S$, and the Kleene closure R^*;
4. for every regular expression, there is a natural number n such that the regular expression is obtained from the atomic elements $\emptyset$ and $x \in V$ by n operations concatenation, union, or star.

The language $L(R)$ which is described by a regular expression R is also inductively defined: $L(\emptyset) = \emptyset$; $L(x) = \{x\}$ for each $x \in V$; and $L(R \cdot S) = L(R) \cdot L(S)$, $L(R \cup S) = L(R) \cup L(S)$, and $L(R^*) = (L(R))^*$ for regular expressions R and S.

The set of all languages generated by some right-linear grammar coincides with the set of all languages accepted by a deterministic finite automaton and with the set of all languages described by a regular expression. All these languages are called regular and form a family denoted by *REG*. Any subfamily of this set is called a subregular language family.

A context-free grammar is a quadruple $G = (N,T,P,S)$ where N, T, and S are as in a right-linear grammar but the production rules in the set P are of the form $A \to w$ with $A \in N$ and $w \in (N \cup T)^*$.

The language generated by a context-free grammar is the set of all words over the terminal alphabet which are obtained from the start symbol S by replacing sequentially the non-terminal symbols according to the rules in the set P. A language is called context-free if it is generated by some context-free grammar. The family of all context-free languages is denoted by *CF*.

With a derivation of a terminal word by a context-free grammar, we associate a derivation tree which has the start symbol in its root and where every node with a non-terminal $A \in N$ has as children nodes with symbols which form, read from left to right, a word w such that $A \to w$ is a rule of the grammar

(if $A \to \lambda$, then the node with A has only one child node and this is labelled with λ). Nodes with terminal symbols or λ have no children. With any derivation tree t of height k and any number $0 \leq j \leq k$, we associate the word of level j and the sentential form of level j which are given by all nodes of depth j read from left to right and all nodes of depth j and all leaves of depth less than j read from left to right, respectively. Obviously, if two words w and v are sentential forms of two successive levels, then $w \Longrightarrow^* v$ holds and this derivation is obtained by a parallel replacement of all non-terminal symbols occurring in the word w.

A context-sensitive grammar is a quadruple $G = (N,T,P,S)$ where N is a finite set of non-terminal symbols, $S \in N$ is the start symbol, T is a finite set of terminal symbols, and P is a finite set of production rules of the form $\alpha \to \beta$ with $\alpha \in (N \cup T)^+ \setminus T^*$, $\beta \in (N \cup T)^*$, and $|\beta| \geq |\alpha|$ with the only exception that $S \to \lambda$ is allowed if the sysmbol S does not occur on any right-hand side of a rule. The language generated by a context-sensitive grammar is the set of all words over the terminal alphabet which are obtained from the start symbol S by replacing sequentially subwords according to the rules in the set P. A language is called context-sensitive if it is generated by some context-sensitive grammar. The family of all context-sensitive languages is denoted by CS. For every context-sensitive language L, there is a context-sensitive grammar $G = (N,T,P,S)$ with $L(G) = L$, where all rules in P are of the form

$$AB \to CD,\ A \to BC,\ A \to B,\ \text{or}\ A \to a$$

with $A,B,C,D \in N$ and $a \in T$, or $S \to \lambda$ if S does not occur on the right-hand side of a rule. Such a grammar is said to be in Kuroda normal form ([17]).

We also mention here four classes of languages without a definition since they are mentioned only in the summary of existing results: By MAT, we denote the family of all languages generated by matrix grammars with appearance checking and without erasing rules; by MAT_{fin}, we denote the family of all such languages where the matrix grammar is of finite index ([5], [23]). By $E0L$ ($ET0L$), we denote the family of all languages generated by extended (tabled) interactionless Lindenmayer systems ([22]).

2.2 Complexity measures and resources restricted languages

Let $G = (N,T,P,S)$ be a right-linear grammar, $A = (V,Z,z_0,F,\delta)$ be a deterministic finite automaton, and L be a regular language. Then, we recall the following complexity measures from [4]:

$$\begin{aligned}
State(A) &= |Z|, Var(G) = |N|, Prod(G) = |P|,\\
State(L) &= \min\{\, State(A) \mid A \text{ is a det. finite automaton accepting } L \,\},\\
Var_{RL}(L) &= \min\{\, Var(G) \mid G \text{ is a right-linear grammar generating } L \,\},\\
Prod_{RL}(L) &= \min\{\, Prod(G) \mid G \text{ is a right-linear grammar generating } L \,\}.
\end{aligned}$$

We now define subregular families by restricting the resources needed for generating or accepting their elements:

$$\begin{aligned}
RL_n^V &= \{\, L \mid L \in REG \text{ with } Var_{RL}(L) \leq n \,\},\\
RL_n^P &= \{\, L \mid L \in REG \text{ with } Prod_{RL}(L) \leq n \,\},\\
REG_n^Z &= \{\, L \mid L \in REG \text{ with } State(L) \leq n \,\}.
\end{aligned}$$

2.3 Subregular language families based on the structure

We consider the following restrictions for regular languages. Let L be a language over an alphabet V.

With respect to the alphabet V, the language L is said to be

- *monoidal* if and only if $L = V^*$,
- *nilpotent* if and only if it is finite or its complement $V^* \setminus L$ is finite,
- *combinational* if and only if it has the form $L = V^*X$ for some subset $X \subseteq V$,
- *definite* if and only if it can be represented in the form $L = A \cup V^*B$ where A and B are finite subsets of V^*,
- *suffix-closed* (or *fully initial* or *multiple-entry* language) if and only if, for any two words $x \in V^*$ and $y \in V^*$, the relation $xy \in L$ implies the relation $y \in L$,
- *ordered* if and only if the language is accepted by some deterministic finite automaton

  $$A = (V, Z, z_0, F, \delta)$$

  with an input alphabet V, a finite set Z of states, a start state $z_0 \in Z$, a set $F \subseteq Z$ of accepting states and a transition mapping δ where $(Z, \preceq)$ is a totally ordered set and, for any input symbol $a \in V$, the relation $z \preceq z'$ implies $\delta(z,a) \preceq \delta(z',a)$,
- *commutative* if and only if it contains with each word also all permutations of this word,
- *circular* if and only if it contains with each word also all circular shifts of this word,
- *non-counting* (or *star-free*) if and only if there is a natural number $k \geq 1$ such that, for every three words $x \in V^*$, $y \in V^*$, and $z \in V^*$, it holds $xy^kz \in L$ if and only if $xy^{k+1}z \in L$,
- *power-separating* if and only if, there is a natural number $m \geq 1$ such that for every word $x \in V^*$, either $J_x^m \cap L = \emptyset$ or $J_x^m \subseteq L$ where $J_x^m = \{ x^n \mid n \geq m \}$,
- *union-free* if and only if L can be described by a regular expression which is only built by product and star,
- *strictly locally k-testable* if and only if there are three subsets B, I, and E of V^k such that any word $a_1a_2 \ldots a_n$ with $n \geq k$ and $a_i \in V$ for $1 \leq i \leq n$ belongs to the language L if and only if

  $$a_1a_2 \ldots a_k \in B,$$
  $$a_{j+1}a_{j+2} \ldots a_{j+k} \in I \text{ for every } j \text{ with } 1 \leq j \leq n-k-1 \text{ and}$$
  $$a_{n-k+1}a_{n-k+2} \ldots a_n \in E,$$

- *strictly locally testable* if and only if it is strictly locally k-testable for some natural number k.

We remark that monoidal, nilpotent, combinational, definite, ordered, union-free, and strictly locally (k-)testable languages are regular, whereas non-regular languages of the other types mentioned above exist. Here, we consider among the commutative, circular, suffix-closed, non-counting, and power-separating languages only those which are also regular.

Some properties of the languages of the classes mentioned above can be found in [24] (monoids), [11] (nilpotent languages), [13] (combinational and commutative languages), [19] (definite languages), [12] and [2] (suffix-closed languages), [25] (ordered languages), [3] (circular languages), [18] (non-counting and strictly locally testable languages), [26] (power-separating languages), [1] (union-free languages).

By *FIN*, *MON*, *NIL*, *COMB*, *DEF*, *SUF*, *ORD*, *COMM*, *CIRC*, *NC*, *PS*, *UF*, SLT_k (for any natural number $k \geq 1$), and *SLT*, we denote the families of all finite, monoidal, nilpotent, combinational, definite, regular suffix-closed, ordered, regular commutative, regular circular, regular non-counting, regular

power-separating, union-free, strictly locally k-testable, and strictly locally testable languages, respectively.

For any natural number $n \geq 1$, let MON_n be the set of all languages that can be represented in the form $A_1^* \cup A_2^* \cup \cdots \cup A_k^*$ with $1 \leq k \leq n$ where all A_i $(1 \leq i \leq k)$ are alphabets. Obviously,

$$MON = MON_1 \subset MON_2 \subset \cdots \subset MON_j \subset \cdots .$$

A strictly locally testable language characterized by three finite sets B, I, and E as above which includes additionally a finite set F of words which are shorter than those of the sets B, I, and E is denoted by $[B,I,E,F]$.

As the set of all families under consideration, we set

$$\begin{aligned}\mathfrak{F} = {} & \{FIN, NIL, COMB, DEF, SUF, ORD, COMM, CIRC, NC, PS, UF\} \\ & \cup \{\, MON_k \mid k \geq 1 \,\} \cup \{SLT\} \cup \{\, SLT_k \mid k \geq 1 \,\} \\ & \cup \{\, RL_n^V \mid n \geq 1 \,\} \cup \{\, RL_n^P \mid n \geq 1 \,\} \cup \{\, REG_n^Z \mid n \geq 1 \,\}.\end{aligned}$$

2.4 Hierarchy of subregular families of languages

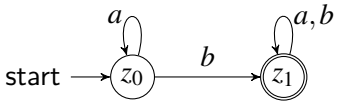
In this section, we present a hierarchy of the families of the aforementioned set $\mathfrak{F}$ with respect to the set theoretic inclusion relation. A summary is depicted in Figure 1.

Before this, we prove some relations of the classes of strictly locally k-testable languages to the subregular language families restricted by resources, which have not been considered in the literature yet.

For this purpose, we first introduce some languages which serve later as witness languages for proper inclusions and incomparabilities.

Lemma 2.1 *The language $L_1 = \{a\}^*\{b\}\{a,b\}^*$ belongs to $REG_2^Z \setminus SLT$.*

Proof. The language L_1 is accepted by the automaton with two states whose transition function is given in the following diagram (double-circled states are accepting):



Suppose, the language L_1 is strictly locally k-testable for some natural number $k \geq 1$. Then, there exist sets $B \subseteq V^k$, $I \subseteq V^k$, $E \subseteq V^k$, and $F \subseteq V^{\leq k-1}$ such that $L_1 = [B,I,E,F]$. Since the word $a^{2k}ba^{2k}$ belongs to the language L_1, we know that $a^k \in B \cap I \cap E$. But then, also the word a^{2k} belongs to the language which is a contradiction. □

Lemma 2.2 *The language $L_2 = [\{a,b\},\{b,c\},\{a,c\},\emptyset]$ belongs to $SLT_1 \setminus REG_4^Z$.*

Proof. By definition, $L_2 \in SLT_1$.

We now prove that L_2 is not accepted by an deterministic finite automaton with less than five states. Let $L = L_2$ and let R_L be the Myhill-Nerode equivalence relation (see [15]): two words x and y are in this relation if and only if, for all words z, either both words xz and yz belong to the language L or none of them. The words λ, a, b, c, and aa are pairwise not in this relation, as one can check.

Therefore, the index of the language L is at least five. Hence, at least five states are necessary for accepting the language L. □

Lemma 2.3 *For each natural number $n \geq 2$, let $V_n = \{a_1, a_2, \ldots, a_{n-1}\}$ be an alphabet with $n-1$ pairwise different letters and let $L_{3,n} = \{a_1 a_2 \ldots a_{n-1}\}$. Then, every language $L_{3,n}$ for $n \geq 2$ belongs to the set $SLT_2 \setminus REG_n^Z$.*

Proof. The statement $L_{3,n} \in SLT_2$ for $n \geq 2$ can be seen as follows. If $n = 2$, then $L_{3,n} = [\emptyset, \emptyset, \emptyset, \{a_1\}]$, otherwise $L_{3,n} = [\{a_1 a_2\}, \{\ a_p a_{p+1} \mid 2 \leq p \leq n-3\ \}, \{a_{n-2} a_{n-1}\}, \emptyset]$.

For accepting any language $L_{3,n}$ for $n \geq 2$, at least $n+1$ states are necessary (follows from the fact that the n partial words $a_1 \ldots a_i$ for $0 \leq i \leq n-1$ and $a_1 a_1$ are pairwise not in the Myhill-Nerode relation). □

Lemma 2.4 *For each natural number $n \geq 1$, let $L_{4,n} = \{a^n\}$. Then $L_{4,n}$ belongs to the set $RL_1^P \setminus SLT_n$.*

Proof. The single word a^n can be generated with one rule, hence, $L_{4,n} \in RL_1^P$.

Assume that such a language is strictly locally n-testable. Then, it is $L_{4,n} = [B, I, E, F]$ for suitable sets B, I, E, and F. From $L_{4,n} = \{a^n\}$, it follows that $B = E = \{a^n\}$. But then, also the word a^{n+1} belongs to the language $L_{4,n}$ which is a contradiction. □

Lemma 2.5 *For each natural number $n \geq 1$, let $V_n = \{a_1, a_2, \ldots, a_n\}$ be an alphabet with n pairwise different letters and let $L_{5,n} = V_n^*$. Then, for $n \geq 1$, the language L_n belongs to the set $SLT_1 \setminus RL_n^P$.*

Proof. The language $L_{5,n}$ can be represented as $L_{5,n} = [V, V, V, \{\lambda\}]$. Hence, $L_{5,n} \in SLT_1$ for $n \geq 1$.

For generating a language $L_{5,n}$ for some number $n \geq 1$, at least a non-terminating rule is necessary for every letter a_i ($1 \leq i \leq n$) and additionally a terminating rule. Hence, $L_{5,n} \notin RL_n^P$. □

Lemma 2.6 *The language $L_6 = \{a\}$ belongs to $RL_1^V \setminus SLT_1$.*

Proof. The language L_6 can be generated with a single rule and, hence, with one non-terminal only.

Assume that L_6 is strictly locally 1-testable and can be represented as $[B, I, E, F]$. Then $B = E = \{a\}$. But then, also the word aa belongs to the language which is a contradiction. □

Lemma 2.7 *The language $L_7 = \{a\}\{b\}^*\{a\} \cup \{a\}$ belongs to $SLT_1 \setminus RL_1^V$.*

Proof. The language L_7 is strictly locally 1-testable and can be represented as $[\{a\}, \{b\}, \{a\}, \emptyset]$.

Assume that the language L_7 is generated by a right-linear grammar with one non-terminal symbol only. Let m be the maximal length of the right-hand side of a rule: $m = \max(\{\ w \mid S \to w \in P\ \})$. Then, the word $ab^m a$ cannot be derived in one step. Hence, there is a derivation $S \Longrightarrow ab^p S \Longrightarrow^* ab^m a$ for some number p with $0 \leq p \leq m-2$. But then, also the derivation $S \Longrightarrow ab^p S \Longrightarrow ab^p ab^p S \Longrightarrow^* ab^p ab^m a$ is possible which yields a word which does not belong to the language L_7. Due to this contradiction, we obtain that $L_7 \notin RL_1^V$. □

Lemma 2.8 *The language $L_8 = \{\ a^{3m} \mid m \geq 1\ \}$ belongs to $RL_1^V \setminus SLT$.*

Proof. The language L_8 is generated by the right-linear grammar $G = (\{S\}, \{a\}, \{S \to a^3 S,\ S \to a^3\}, S)$. Hence, $L_8 \in RL_1^V$.

Assume that the language L_8 is generated by a strictly locally k-testable grammar for some number $k \geq 1$. Then, L_8 has a representation as $[B, I, E, F]$ with $B \cup I \cup E \subseteq \{a\}^k$ and $F \subseteq \{a\}^{\leq k-1}$. Since the word a^{3k} belongs to the language L_8, we obtain that B, I, and E contain the word a^k. But then, also the word a^{3k+1} belongs to the language L_8 which is a contradiction. □

Lemma 2.9 *For each natural number $n \geq 1$, let $V_n = \{a_1, a_2, \ldots, a_{n+1}\}$ be an alphabet with $n+1$ pairwise different letters and let $L_{9,n} = \{a_1\}^+\{a_2\}^+ \cdots \{a_{n+1}\}^+$. Then, for $n \geq 1$, the language $L_{9,n}$ belongs to the set $SLT_2 \setminus RL_n^V$.*

Proof. The language $L_{9,n}$ can be represented as

$$L_{9,n} = [\{a_1a_1, a_1a_2\}, \{\, a_pa_p \mid 1 \leq p \leq n+1 \,\} \cup \{\, a_pa_{p+1} \mid 1 \leq p \leq n \,\}, \{a_na_{n+1}, a_{n+1}a_{n+1}\}, \emptyset].$$

Hence, $L_{9,n} \in SLT_2$ for $n \geq 1$.

For generating a language $L_{9,n}$ for some number $n \geq 1$, at least a non-terminal symbol is necessary for every letter a_i ($1 \leq i \leq n+1$). Hence, $L_{9,n} \notin RL_n^V$. □

We now prove inclusion relations and incomparabilities.

Lemma 2.10 *The class SLT_1 is properly included in the class REG_5^Z.*

Proof. We first prove the inclusion $SLT_1 \subseteq REG_5^Z$.

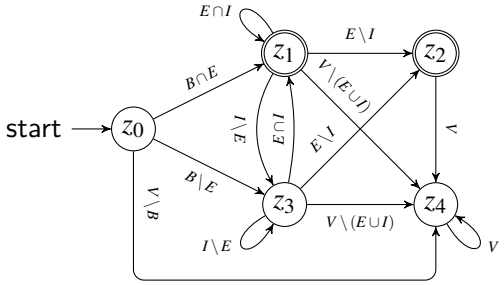
Let L be a strictly locally 1-testable language. Then $L = [B,I,E,F]$ with $B \subseteq V$, $I \subseteq V$, $E \subseteq V$, and $F \subseteq \{\lambda\}$. We construct the following deterministic finite automaton:

$$A = (V, \{z_0, z_1, \ldots, z_4\}, z_0, Z_\mathrm{f}, \delta)$$

where

$$Z_\mathrm{f} = \{z_1, z_2\} \cup \begin{cases} \{z_0\}, & \text{if } \lambda \in F, \\ \emptyset, & \text{otherwise,} \end{cases}$$

and the transition function δ is given by the following diagram (z_0 is an accepting state if and only if $\lambda \in F$):



Due to space reasons, we leave the proof that $L(A) = L$ to the reader. From the construction follows the inclusion $SLT_1 \subseteq REG_5^Z$.

A witness language for the properness of this inclusion is the language $L_1 = \{a\}^*\{b\}\{a,b\}^*$ from Lemma 2.1. □

Lemma 2.11 *The class SLT_1 is incomparable to the classes REG_i^Z for $i \in \{2,3,4\}$.*

Proof. Due to the inclusion relations, it suffices to show that there is a language in the set $REG_2^Z \setminus SLT_1$ and a language in the set $SLT_1 \setminus REG_4^Z$. A language for the first case is $L_1 = \{a\}^*\{b\}\{a,b\}^*$ as shown in Lemma 2.1. A language for the second case is $L_2 = [\{a,b\}, \{b,c\}, \{a,c\}, \emptyset]$ as shown in Lemma 2.2. □

Lemma 2.12 *The classes SLT_k for $k \geq 2$ and SLT are incomparable to the classes REG_n^Z for $n \geq 2$.*

Proof. Due to the inclusion relations, it suffices to show that there is a language in the set $REG_2^Z \setminus SLT$ and a language in each set $SLT_2 \setminus REG_n^Z$ for $n \geq 2$. A language for the first case is $L_1 = \{a\}^*\{b\}\{a,b\}^*$ as shown in Lemma 2.1. Languages for the second case are $L_{3,n} = \{a_1 a_2 \dots a_{n-1}\}$ as shown in Lemma 2.3. □

Lemma 2.13 *The classes SLT_k for $k \geq 1$ are incomparable to the classes RL_n^P for $n \geq 1$.*

Proof. Due to the inclusion relations, it suffices to show that there is a language in the set $RL_1^P \setminus SLT_k$ for every $k \geq 1$ and a language in each set $SLT_1 \setminus RL_n^P$ for $n \geq 1$. Languages for the first case are $L_{4,k} = \{a^k\}$ for $k \geq 1$ as shown in Lemma 2.4. Languages for the second case are $L_{5,n} = \{a_1, a_2, \dots, a_n\}^*$ as shown in Lemma 2.5. □

Lemma 2.14 *The class SLT_1 is properly included in the class RL_2^V.*

Proof. Let $L = [B, I, E, F]$ be a strictly locally 1-testable language over an alphabet T. We construct a right-linear grammar $G = (\{S, S'\}, T, P, S)$ with the rules

- $S \to w$ for every word $w \in F \cup (B \cap E)$,
- $S \to wS'$ for every word $w \in B$,
- $S' \to wS'$ for every word $w \in I$, and
- $S' \to w$ for every word $w \in E$.

The language $L(G)$ generated is $F \cup (B \cap E) \cup (BI^*E)$ which is L. Hence, $L \in RL_2^V$ and $SLT_1 \subseteq RL_2^V$. A witness language for the properness of the inclusion is $L_6 = \{a\}$ for which was proved in Lemma 2.6 that it belongs to the set RL_1^V and therefore also to RL_2^V but not to SLT_1. □

Lemma 2.15 *The class SLT_1 is incomparable to the class RL_1^V.*

Proof. There is a language in the set $RL_1^V \setminus SLT_1$, namely $L_6 = \{a\}$ as shown in Lemma 2.6, and a language in the set $SLT_1 \setminus RL_1^V$, namely $L_7 = \{a\}\{b\}^*\{a\} \cup \{a\}$ as shown in Lemma 2.7. □

Lemma 2.16 *The classes SLT_k for $k \geq 2$ and SLT are incomparable to the classes RL_n^V for $n \geq 1$.*

Proof. Due to the inclusion relations, it suffices to show that there is a language in the set $RL_1^V \setminus SLT$ and a language in the set $SLT_2 \setminus RL_n^V$ for every number $n \geq 1$. A language for the first case is $L_8 = \{\, a^{3m} \mid m \geq 1 \,\}$ as shown in Lemma 2.8. A language for the second case is $L_{9,n} = \{a_1\}^+\{a_2\}^+ \cdots \{a_{n+1}\}^+$ as shown in Lemma 2.9. □

A summary of the inclusion relations is given in Figure 1. An edge label in this figure refers to the paper or lemma above where the respective inclusion is proved.

Theorem 2.17 *The inclusion relations presented in Figure 1 hold. An arrow from an entry X to an entry Y depicts the proper inclusion $X \subset Y$; if two families are not connected by a directed path, then they are incomparable.*

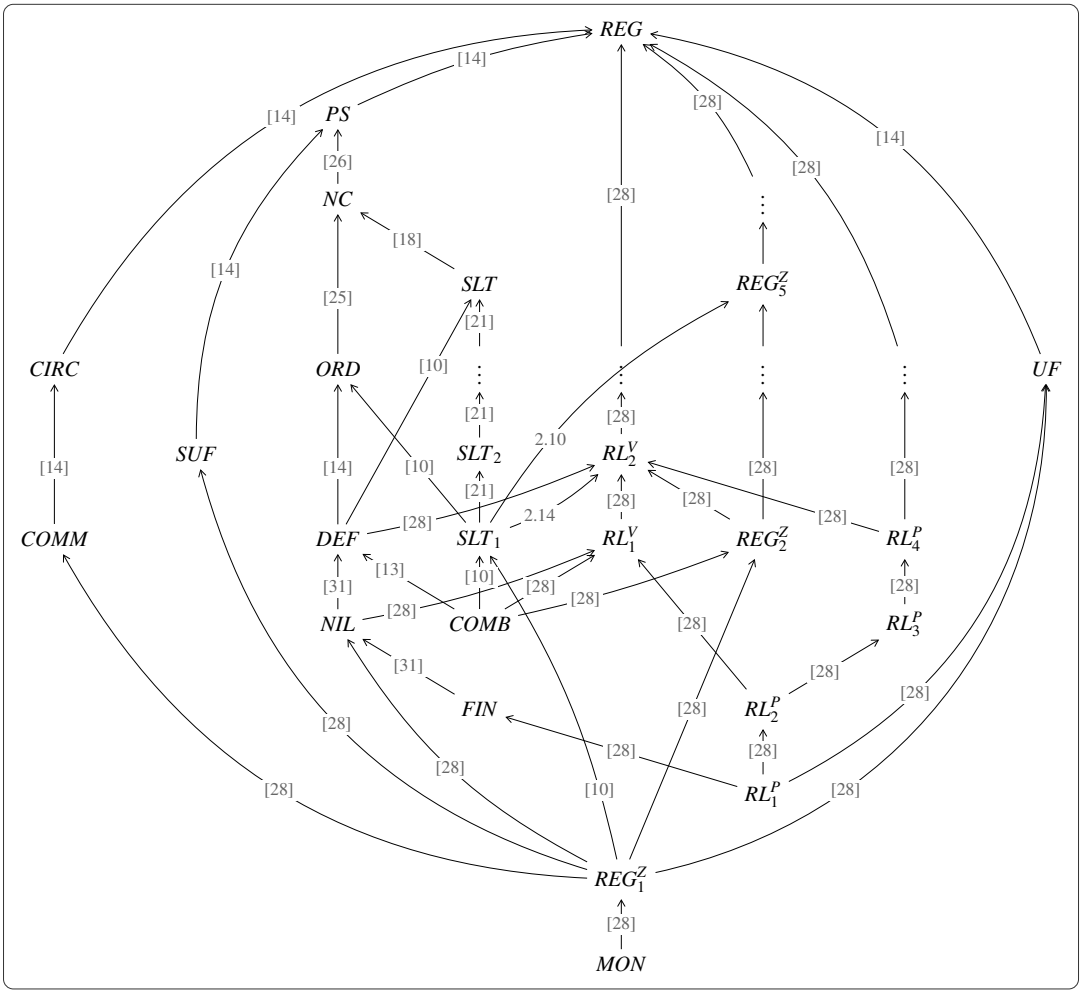


Figure 1: Hierarchy of subregular language families

2.5 Tree-controlled grammars

A tree-controlled grammar is a quintuple $G = (N, T, P, S, R)$ where

- (N, T, P, S) is a context-free grammar with a set N of non-terminal symbols, a set T of terminal symbols, a set P of context-free non-erasing rules (with the only exception that the rule $S \to \lambda$ is allowed if S does not occur on a right-hand side of a rule), and an axiom S,
- R is a regular set over $N \cup T$.

The language $L(G)$ generated by a tree-controlled grammar $G = (N, T, P, S, R)$ consists of all such words $z \in T^*$ which have a derivation tree t where z is the word obtained by reading the leaves from left to right and the words of all levels of t – besides the last one – belong to the regular control language R.

Let $\mathscr{F}$ be a subfamily of *REG*. Then, we denote the family of languages generated by tree-controlled grammars $G = (N, T, P, S, R)$ with $R \in \mathscr{F}$ by $\mathscr{TC}(\mathscr{F})$.

Example 2.18 *As an example, we consider the tree-controlled grammar*

$$G_1 = (\{S\}, \{a\}, \{S \to SS, S \to a\}, S, \{S\}^*).$$

Since the terminal symbol a is not allowed to appear before the last level, on all levels before, any occurrence of S is replaced by SS. Finally, any letter S is replaced by a. Therefore, the levels of an allowed derivation tree consist of the words S, SS, SSSS, …, S^{2^n}, a^{2^n} for some $n \geq 0$. Thus, $L(G_1) = \{ a^{2^n} \mid n \geq 0 \}$. Due to the structure of the control language which is monoidal and can be generated by a grammar with one non-terminal symbol and two rules, we further obtain

$$L(G_1) \in \mathscr{TC}(MON) \cap \mathscr{TC}(RL_1^V) \cap \mathscr{TC}(RL_2^P).$$

Example 2.19 *We now consider the tree-controlled grammar*

$$G_2 = (\{S,A,B,C\}, \{a,b,c\}, P, S, \{S, aAbBcC\})$$

with

$$P = \{S \to aAbBcC, A \to aA, B \to bB, C \to cC, A \to a, B \to b, C \to c\}.$$

By the definition of the control language, any derivation in G_2 has the form

$$S \Longrightarrow aAbBcC \Longrightarrow aaAbbBccC \Longrightarrow \ldots \Longrightarrow a^{n-1}Ab^{n-1}Bc^{n-1}C \Longrightarrow a^n b^n c^n$$

with $n \geq 2$. Thus, the tree-controlled grammar G_2 generates the non-context-free language

$$L(G_2) = \{a^n b^n c^n\} n \geq 2.$$

Due to the structure of the control language which is finite and can be generated by a grammar with one non-terminal symbol and two rules, we further obtain

$$L(G_2) \in \mathscr{TC}(FIN) \cap \mathscr{TC}(RL_1^V) \cap \mathscr{TC}(RL_2^P).$$

In [20] (see also [5]), it has been shown that a language L is generated by a tree-controlled grammar if and only if it is generated by a context-sensitive grammar.

Theorem 2.20 ([20], [5]) *It holds $\mathscr{TC}(REG) = CS$.*

In subsequent papers, tree-controlled grammars have been investigated where the control language belongs to some subfamily of the class *REG* ([6, 7, 8, 9, 27, 29, 30]). In this paper, we continue this research with further subregular language families.

From the definition follows that the subset relation is preserved under the use of tree-controlled grammars: if we allow more, we do not obtain less.

Lemma 2.21 *For any two language classes X and Y with $X \subseteq Y$, we have the inclusion*

$$\mathscr{TC}(X) \subseteq \mathscr{TC}(Y).$$

A summary of the inclusion relations known so far is given in Figure 2. An arrow from an entry X to an entry Y depicts the inclusion $X \subseteq Y$; a solid arrow means proper inclusion; a dashed arrow indicates that it is not known whether the inclusion is proper. If two families are not connected by a directed path, then they are not necessarily incomparable. An edge label in this figure refers to the paper where the respective inclusion is proved.

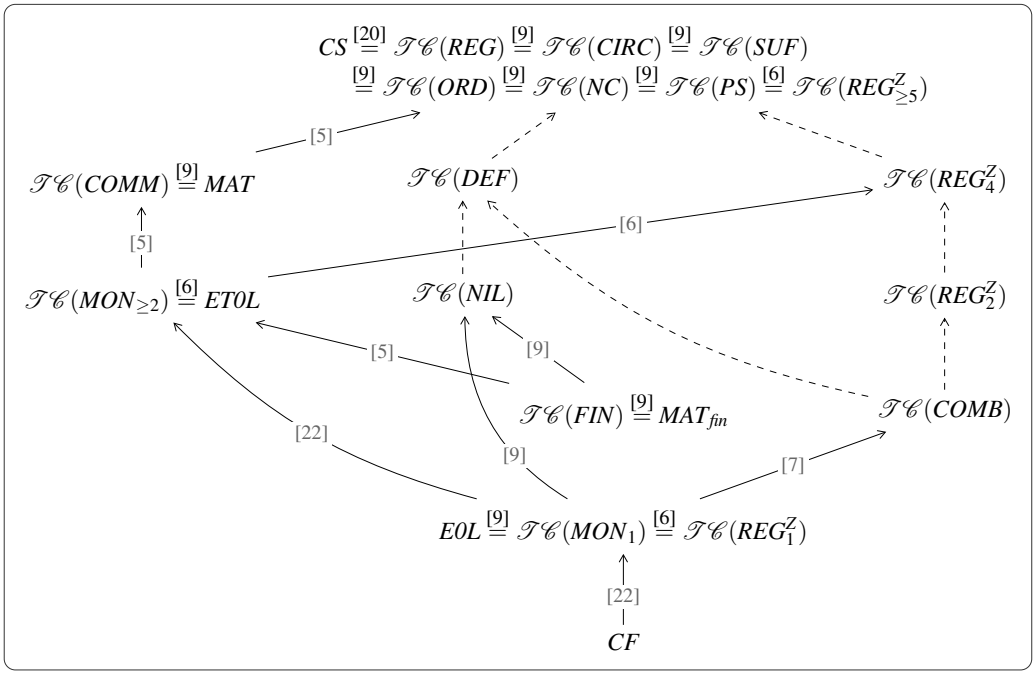


Figure 2: Hierarchy of subregularly tree-controlled language families

3 Results

We insert the classes $\mathscr{TC}(SLT_k)$ for $k \geq 1$, $\mathscr{TC}(SLT)$, $\mathscr{TC}(RL^V_n)$ for $n \geq 1$, and $\mathscr{TC}(RL^P_n)$ for $n \geq 1$ into the existing hierachy (see Figure 2).

The inclusions follow from the inclusion relations of the respective families of the control languages (see Figure 1 and Lemma 2.21).

In most cases, we obtain that any context-sensitive language can be generated by a tree-controlled grammar where the control language is taken from that family.

Theorem 3.1 *We have $\mathscr{TC}(SLT_k) = CS$ for $k \geq 2$ and $\mathscr{TC}(SLT) = CS$.*

Proof. Let L be a context-sensitive language. Then, there is a context-sensitive grammar $G = (N,T,P,S)$ with $L(G) = L$ which is in Kuroda normal form, where the rule set P can be divided into two sets P_1 and P_2 such that all rules of P_1 are of the form $A \to BC$ or $A \to B$ or $A \to a$ with $A,B,C,D \in N$ and $a \in T$ and all rules of P_2 are of the form $AB \to CD$ with $A,B,C,D \in N$.

We will construct a tree-controlled grammar G_{tc} which simulates the grammar G. Since G_{tc} has only context-free rules, the non-context-free rules of G have to be substituted by context-free rules and some control such that the parts of a non-context-free rule which are independent from the view of the core grammar of G_{tc} remain connected.

We label the non-context-free rules and associate the non-terminal symbols of their left-hand sides with new non-terminal symbols which are marked with the rule label and the position (first or second letter). The context-free rules can be freely applied also in the tree-controlled grammar. A non-context-free rule $p : AB \to CD$ will be simulated by context-free rules

$$A \to A_{p,1},\ B \to B_{p,2},\ A_{p,1} \to C, \text{ and } B_{p,2} \to D.$$

The control language ensures that the rules which belong together (here $A \to A_{p,1}$ and $B \to B_{p,2}$) are applied together (at the same time and next to each other). If a terminal symbol is produced in a sentential form of the grammar G, then it remains there until the whole terminal word is produced. In the tree-controlled grammar G_{tc}, one has to keep track of terminal symbols because they 'disappear' (once produced, they are not present in the next level anymore) and then two non-terminal symbols appear next to each other, although they are not neighbours in the sentential form. So, the tree-controlled grammar should produce placeholders for terminal symbols and replace them by the actual terminal symbols only in the very end. In a tree-controlled grammar, from one level to the next, all non-terminal symbols are replaced. This can be seen as some kind of shortcut where production rules which are independent from each other are applied in parallel.

We construct such a tree-controlled grammar $G_{\mathrm{tc}} = (N_{\mathrm{tc}}, T, P_{\mathrm{tc}}, S, R_{\mathrm{tc}})$. The terminal alphabet and start symbol are the same as in the grammar G. We now give the rules; the non-terminal symbols will be collected later from the rules. At the end, we will give the control language R_{tc}.

In order to simulate the context-free rules directly, we take all non-terminating rules of them from G as they are:

$$P_{\mathrm{cf}} = P \cap (\{ A \to BC \mid A,B,C \in N \} \cup \{ A \to B \mid A,B \in N \}).$$

Instead of the terminating rules, we take rules with a placeholder (for each terminal symbol a, we introduce a unique non-terminal symbol $\hat{a}$), but finally, those placeholders have to be terminated:

$$P_{\mathrm{t}} = \{ A \to \hat{a} \mid A \in N, a \in T, A \to a \in P \} \cup \{ \hat{a} \to a \mid a \in T \}.$$

We give also rules which can delay the derivation such that not everything needs to be replaced in parallel:

$$P_{\mathrm{d}} = \{ A \to A \mid A \in N \} \cup \{ \hat{a} \to \hat{a} \mid a \in T \}.$$

For simulating the non-context-free rules, first rules are applied which mark the position of the intended application such that the control language has the chance to check whether the plan is alright (if it is not, then the derivation will block). In the next step, the markers will be replaced by their actual target non-terminal symbols:

$$P_{\mathrm{cs}} = \bigcup_{p:AB \to CD \in P} \{A \to A_{p,1},\ B \to B_{p,2},\ A_{p,1} \to C,\ B_{p,2} \to D\}.$$

Other rules are not needed, hence,

$$P_{\mathrm{tc}} = P_{\mathrm{cf}} \cup P_{\mathrm{t}} \cup P_{\mathrm{d}} \cup P_{\mathrm{cs}}.$$

The set N_{tc} of non-terminal symbols results as follows:

$$N_{\mathrm{cf}} = N \cup \{ \hat{a} \mid a \in T \},\ N_1 = \{ A_{p,1} \mid p : AB \to CD \in P \},\ N_2 = \{ B_{p,2} \mid p : AB \to CD \in P \},$$
$$N_{12} = \{ A_{p,1}B_{p,2} \mid p : AB \to CD \in P \},\ N_{\mathrm{tc}} = N_{\mathrm{cf}} \cup N_1 \cup N_2.$$

A derivation can go wrong only if the simulation of a non-context-free rule is not properly planned. Hence, as control language, we take

$$R_{\mathrm{tc}} = (N_{\mathrm{cf}} \cup N_{12})^*.$$

Since the context-free rules of the grammar G can be applied independently from each other and do not have to be applied at a certain time (thanks to the rules from the subset P_{d}) and the correct simulation

of the non-context-free rules is ensured by the control language R_{tc}, it is not hard to see that the generated languages $L(G)$ and $L(G_{\mathrm{tc}})$ coincide.

The control language R_{tc} is strictly locally 2-testable as can be seen from the following representation: Let

$$B = N_{\mathrm{cf}}^2 \cup N_{\mathrm{cf}} N_1 \cup N_{12}, \qquad I = N_{\mathrm{cf}}^2 \cup N_{\mathrm{cf}} N_1 \cup N_{12} \cup N_2 N_{\mathrm{cf}} \cup N_2 N_1,$$
$$E = N_{\mathrm{cf}}^2 \cup N_{12} \cup N_2 N_{\mathrm{cf}}, \qquad F = N_{\mathrm{cf}} \cup \{\lambda\}.$$

Then $R_{\mathrm{tc}} = [B, I, E, F]$.

Altogether, we obtain $CS \subseteq \mathscr{TC}(SLT_2) \subseteq \mathscr{TC}(SLT_k) \subseteq \mathscr{TC}(SLT) \subseteq CS$ for $k \geq 3$. Thus, it holds $\mathscr{TC}(SLT_k) = CS$ for $k \geq 2$ and $\mathscr{TC}(SLT) = CS$. □

Theorem 3.2 *We have $\mathscr{TC}(RL_n^V) = CS$ for $n \geq 1$.*

Proof. The control language $R_{\mathrm{tc}} = (N_{\mathrm{cf}} \cup N_{12})^*$ from the tree-controlled grammar G_{tc} in the proof of Theorem 3.1 can be generated by a right-linear grammar $G' = (\{S'\}, N_{\mathrm{tc}}, P', S')$ where

$$P' = \{\, S' \to xS' \mid x \in N_{\mathrm{cf}} \cup N_{12} \,\} \cup \{\, S' \to x \mid x \in N_{\mathrm{cf}} \cup N_{12} \,\}.$$

Hence, $CS \subseteq \mathscr{TC}(RL_1^V) \subseteq \mathscr{TC}(RL_n^V) \subseteq CS$ for $n \geq 2$. Thus, we conclude $\mathscr{TC}(RL_n^V) = CS$ for $n \geq 1$. □

From the proof of Theorem 3.1, we conclude also the following statement.

Theorem 3.3 *We have $\mathscr{TC}(UF) = CS$.*

Proof. Let $L = \{w_1, w_2, \ldots, w_n\}$ be a finite language. Then $L^* = (\{w_1\}^*\{w_2\}^* \cdots \{w_n\}^*)^*$ and is therefore union-free.

The control language $R_{\mathrm{tc}} = (N_{\mathrm{cf}} \cup N_{12})^*$ from the tree-controlled grammar G_{tc} in the proof of Theorem 3.1 is the Kleene closure of a finite language and, hence, it is union-free. □

Regarding the classes $\mathscr{TC}(RL_n^P)$ for $n \geq 1$, the situation is different since the number of rules depends on the size of the alphabet (which is not necessarily the case for the number of non-terminal symbols or the number of states).

If the control language is generated with one rule only, then either the control language is the empty set (if the right-hand side of the rule contains a non-terminal symbol) or it contains exactly one terminal word. Since the start symbol of the tree-controlled grammar always forms the first level of the derivation tree, it must be contained in the control language (otherwise, the derivation would be blocked right from the beginning). Therefore, we obtain the following result.

Lemma 3.4 *Let $G = (N, T, P, S, R)$ a tree-controlled grammar with $R \in RL_1^P$. Then, the generated language is*

$$L(G) = \begin{cases} \{\, w \mid w \in T^* \text{ and } S \to w \in P \,\}, & \text{if } R = \{S\}, \\ \emptyset, & \text{otherwise}. \end{cases}$$

Proof. If $R = \{S\}$, then every level but the last one of the derivation tree is S and the last level is a terminal word which is produced by S. On the other hand, all terminal words derived from S belong to the generated language.

If $R \neq \{S\}$, then $S \notin R$ since R contains at most one word because $R \in RL_1^P$. Since S is the word of the first level of the derivation tree, there is no derivation possible. Hence, $L(G)$ is empty. □

From this result, the next one immediately follows.

Theorem 3.5 *We have* $\mathscr{TC}(RL_1^P) = FIN$.

Proof. The inclusion $\mathscr{TC}(RL_1^P) \subseteq FIN$ follows from Lemma 3.4. The inclusion $FIN \subseteq \mathscr{TC}(RL_1^P)$ can also be seen from Lemma 3.4: Let L be a finite language over an alphabet T. Then, construct a tree-controlled grammar $G = (\{S\}, T, \{ S \to w \mid w \in L \}, S, \{S\})$. It holds $L(G) = L$ and $L(G) \in \mathscr{TC}(RL_1^P)$. □

If the control language is taken from the family $\mathscr{TC}(RL_2^P)$, then already context-sensitive languages can be generated as the Examples 2.18 and 2.19 show.

Theorem 3.6 *We have* $\mathscr{TC}(RL_1^P) \subset \mathscr{TC}(RL_2^P)$.

Proof. The inclusion follows from Theorem 2.17 and Lemma 2.21. According to Theorem 3.5, the family $\mathscr{TC}(RL_1^P)$ contains finite languages only. As shown in the Examples 2.18 and 2.19, the family $\mathscr{TC}(RL_2^P)$ contains non-context-free languages. □

A summary of all the inclusion relations is given in Figure 3. An arrow from an entry X to an entry Y depicts the inclusion $X \subseteq Y$; a solid arrow means proper inclusion; a dashed arrow indicates that it is not known whether the inclusion is proper. If two families are not connected by a directed path, then they are not necessarily incomparable. An edge label in this figure refers to the paper or theorem above where the respective inclusion is proved.

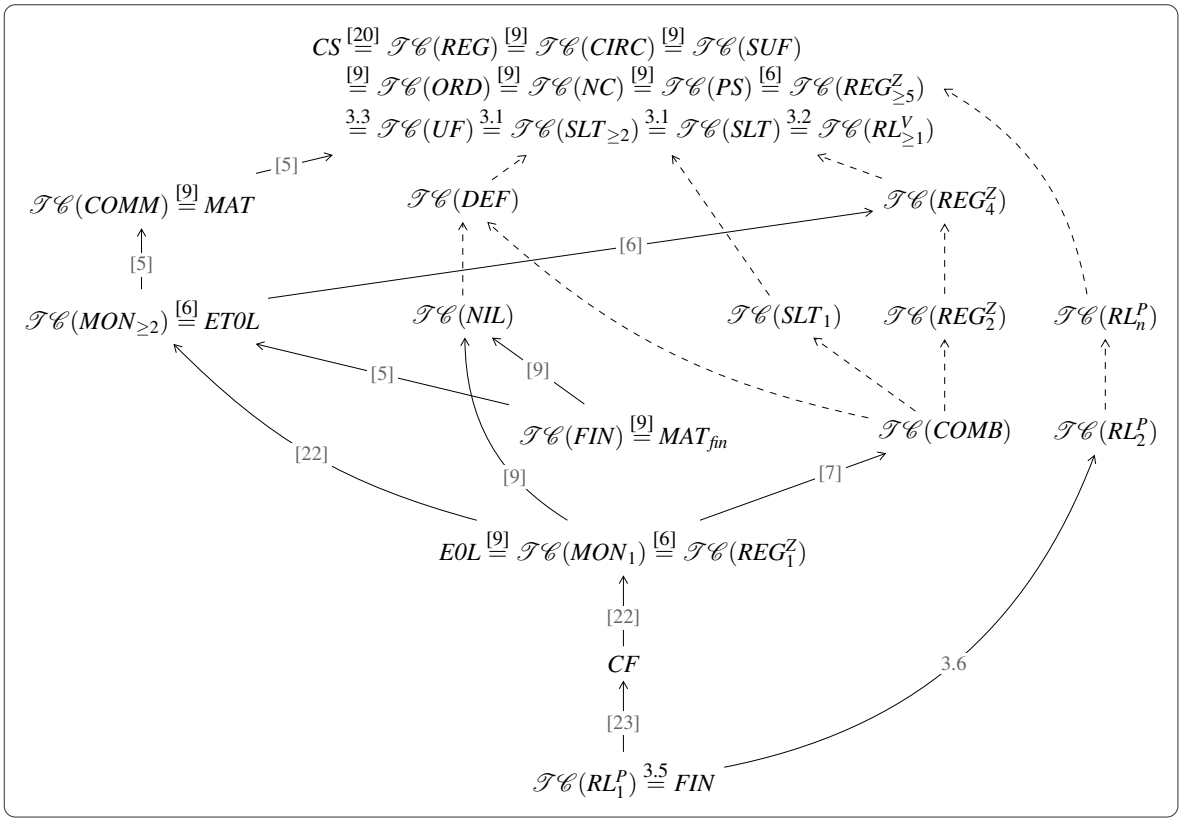


Figure 3: New Hierarchy of subregularly tree-controlled language families

4 Conclusion

There are several families of languages generated by tree-controlled grammars where we do not have a characterization by some other language class. The strictness of some inclusions and the incomparability of some families remain as open problems.

In the present paper, we have only considered tree-controlled grammars without erasing rules. For tree-controlled grammars where erasing rules are allowed, several results have been published already (see, e. g., [7, 29, 30]). Also in this situation, there are some open problems.

Another direction for future research is to consider other subregular language families or to relate the families of languages generated by tree-controlled grammars to language families obtained by other grammars/systems with regulated rewriting.

References

[1] Janusz A. Brzozowski (1962): *Regular Expression Techniques for Sequential Circuits*. Ph.D. thesis, Princeton University, Princeton, NJ, USA.

[2] Janusz A. Brzozowski, Galina Jirásková & C. Zou (2014): *Quotient complexity of closed languages*. *Theory of Computing Systems* 54, pp. 277–292, doi:10.1007/s00224-013-9515-7.

[3] Jürgen Dassow (1979): *On the circular closure of languages*. *Elektronische Informationsverarbeitung und Kybernetik/Journal of Information Processing and Cybernetics* 15(1–2), pp. 87–94.

[4] Jürgen Dassow, Florin Manea & Bianca Truthe (2011): *On contextual grammars with subregular selection languages*. In Markus Holzer, Martin Kutrib & Giovanni Pighizzini, editors: *Descriptional Complexity of Formal Systems – 13th International Workshop, DCFS 2011, Gießen/Limburg, Germany, July 25 – 27, 2011. Proceedings*, *LNCS* 6808, Springer-Verlag, pp. 135–146, doi:10.1007/978-3-642-22600-7_11.

[5] Jürgen Dassow & Gheorghe Păun (1989): *Regulated Rewriting in Formal Language Theory*. *EATCS Monographs in Theoretical Computer Science* 18, Springer-Verlag, doi:10.1007/978-3-642-74932-2.

[6] Jürgen Dassow, Ralf Stiebe & Bianca Truthe (2009): *Two collapsing hierarchies of subregularly tree controlled languages*. *Theoretical Computer Science* 410(35), pp. 3261–3271, doi:10.1016/j.tcs.2009.03.005.

[7] Jürgen Dassow, Ralf Stiebe & Bianca Truthe (2010): *Generative capacity of subregularly tree controlled grammars*. *International Journal of Foundations of Computer Science* 21(5), pp. 723–740, doi:10.1142/S0129054110007520.

[8] Jürgen Dassow & Bianca Truthe (2008): *On two hierarchies of subregularly tree controlled languages*. In Cezar Câmpeanu & Giovanni Pighizzini, editors: *Descriptional Complexity of Formal Systems, 10th International Workshop, Charlottetown, Prince Edward Island, Canada, July 16–18, 2008, Proceedings*, University of Prince Edward Island, pp. 145–156.

[9] Jürgen Dassow & Bianca Truthe (2008): *Subregularly tree controlled grammars and languages*. In Erzsébet Csuhaj-Varjú & Zoltán Ésik, editors: *Automata and Formal Languages, 12th International Conference, AFL 2008, Balatonfüred, Hungary, May 27–30, 2008, Proceedings*, Computer and Automation Research Institute, Hungarian Academy of Sciences, pp. 158–169.

[10] Jürgen Dassow & Bianca Truthe (2022): *On the generative capacity of contextual grammars with strictly locally testable selection languages*. In Henning Bordihn, Géza Horváth & György Vaszil, editors: *12th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2022), Debrecen, Hungary, August 26 – 27, 2022. Proceedings*, *EPTCS* 367, Open Publishing Association, pp. 65–80, doi:10.4204/EPTCS.367.5.

[11] Ferenc Gécseg & István Peák (1972): *Algebraic Theory of Automata*. Academiai Kiado, Budapest.

[12] Arthur Gill & Lawrence T. Kou (1974): *Multiple-entry finite automata*. *Journal of Computer and System Sciences* 9(1), pp. 1–19, doi:10.1016/S0022-0000(74)80034-6.

[13] Ivan M. Havel (1969): *The theory of regular events II*. *Kybernetika* 5(6), pp. 520–544.

[14] Markus Holzer & Bianca Truthe (2015): *On relations between some subregular language families*. In Rudolf Freund, Markus Holzer, Nelma Moreira & Rogério Reis, editors: *Seventh Workshop on Non-Classical Models of Automata and Applications (NCMA), Porto, Portugal, August 31 – September 1, 2015, Proceedings*, *books@ocg.at* 318, Österreichische Computer Gesellschaft, pp. 109–124.

[15] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading.

[16] Karel Čulik II & Hermann A. Maurer (1977): *Tree controlled grammars*. *Computing* 19(2), pp. 129–139, doi:10.1007/BF02252350.

[17] Sige Yuki Kuroda (1964): *Classes of languages and linear bounded automata*. *Information and Control* 7(2), pp. 207–223, doi:10.1016/S0019-9958(64)90120-2.

[18] Robert McNaughton & Seymour Papert (1971): *Counter-Free Automata*. MIT Press, Cambridge, USA.

[19] Micha A. Perles, Michael O. Rabin & Eliahu Shamir (1963): *The theory of definite automata*. *IEEE Trans. Electronic Computers* 12, pp. 233–243, doi:10.1109/PGEC.1963.263534.

[20] Gheorghe Păun (1979): *On the generative capacity of tree controlled grammars*. *Computing* 21, pp. 213–220, doi:10.1007/BF02253054.

[21] Stefano Crespi Reghizzi & Pierluigi San Pietro (2011): *From regular to strictly locally testable languages*. In Petr Ambrož, Štěpán Holub & Zuzana Masáková, editors: *8th International Conference WORDS 2011*, *EPTCS* 63, pp. 103–111, doi:10.4204/EPTCS.63.14.

[22] Grzegorz Rozenberg & Arto Salomaa (1980): *The Mathematical Theory of L Systems*. Academic Press.

[23] Grzegorz Rozenberg & Arto Salomaa, editors (1997): *Handbook of Formal Languages*. Springer-Verlag, Berlin.

[24] Huei-Jan Shyr (1991): *Free Monoids and Languages*. Hon Min Book Co., Taichung, Taiwan.

[25] Huei-Jan Shyr & Gabriel Thierrin (1974): *Ordered automata and associated languages*. *Tankang Journal of Mathematics* 5(1), pp. 9–20.

[26] Huei-Jan Shyr & Gabriel Thierrin (1974): *Power-separating regular languages*. *Mathematical Systems Theory* 8(1), pp. 90–95, doi:10.1007/BF01761710.

[27] Ralf Stiebe (2008): *On the complexity of the control language in tree controlled grammars*. In Jürgen Dassow & Bianca Truthe, editors: *Colloquium on the Occasion of the 50th Birthday of Victor Mitrana, Otto-von-Guericke-Universitäat Magdeburg, Germany, June 27, 2008, Proceedings*, Otto-von-Guericke-Universitäat Magdeburg, Germany, pp. 29–36.

[28] Bianca Truthe (2018): *Hierarchy of Subregular Language Families*. Technical Report, Justus-Liebig-Universität Giessen, Institut für Informatik, IFIG Research Report 1801.

[29] Sherzod Turaev, Jürgen Dassow, Florin Manea & Mohd Hasan Selamat (2012): *Language classes generated by tree controlled grammars with bounded nonterminal complexity*. *Theoretical Computer Science* 449, pp. 134–144, doi:10.1016/j.tcs.2012.04.013.

[30] György Vaszil (2012): *On the nonterminal complexity of tree controlled grammars*. In Henning Bordihn, Martin Kutrib & Bianca Truthe, editors: *Languages Alive – Essays Dedicated to Jürgen Dassow on the Occasion of His 65th Birthday*, *LNCS* 7300, Springer, pp. 265–272, doi:10.1007/978-3-642-31644-9_18.

[31] Barbara Wiedemann (1978): *Vergleich der Leistungsfähigkeit endlicher determinierter Automaten*. Diplomarbeit, Universität Rostock.

GAPs for Shallow Implementation of Quantum Finite Automata

Mansur Ziiatdinov
Kazan Federal University, Kazan 420008, Russia
gltronred@gmail.com

Aliya Khadieva
Faculty of Computing, University of Latvia, Rīga, Latvia
Kazan Federal University, Kazan 420008, Russia
aliya.khadi@gmail.com

Abuzer Yakaryılmaz
Faculty of Computing, University of Latvia, Rīga, Latvia
abuzer.yakaryilmaz@lu.lv

Quantum fingerprinting is a technique that maps classical input word to a quantum state. The obtained quantum state is much shorter than the original word, and its processing uses less resources, making it useful in quantum algorithms, communication, and cryptography. One of the examples of quantum fingerprinting is quantum automata algorithms for $MOD_p = \{a^{i \cdot p} \mid i \geq 0\}$ languages, where p is a prime number.

However, implementing such an automaton on the current quantum hardware is not efficient. Quantum fingerprinting maps a word $x \in \{0,1\}^n$ of length n to a state $|\psi(x)\rangle$ of $O(\log n)$ qubits, and uses $O(n)$ unitary operations. Computing quantum fingerprint using all available qubits of the current quantum computers is infeasible due to a large number of quantum operations.

To make quantum fingerprinting practical, we should optimize the circuit for depth instead of width in contrast to the previous works. We propose explicit methods of quantum fingerprinting based on tools from additive combinatorics, such as generalized arithmetic progressions (GAPs), and prove that these methods provide circuit depth comparable to a probabilistic method. We also compare our method to prior work on explicit quantum fingerprinting methods.

1 Introduction

A quantum finite state automaton (QFA) is a generalization of classical finite automaton [16, 4]. Here we use the known simplest QFA model [12]. Formally, a QFA is 5-tuple $M = (Q, A \cup \{¢, \$\}, |\psi_0\rangle, \mathscr{U}, \mathscr{H}_{acc})$, where $Q = \{q_1, \ldots, q_D\}$ is a finite set of states, A is the finite input alphabet, $¢, \$$ are the left and right end-markers, respectively. The state of M is represented as a vector $|\psi\rangle \in \mathscr{H}$, where $\mathscr{H}$ is the D-dimensional Hilbert space spanned by $\{|q_1\rangle, \ldots, |q_D\rangle\}$ (here $|q_j\rangle$ is a zero column vector except its j-th entry that is 1). The automaton M starts in the initial state $|\psi_0\rangle \in \mathscr{H}$, and makes transitions according to the operators $\mathscr{U} = \{U_a \mid a \in A\}$ of unitary matrices. After reading the whole input word, the final state is observed with respect to the accepting subspace $\mathscr{H}_{acc} \subseteq \mathscr{H}$.

Quantum fingerprinting provides a method of constructing automata for certain problems. It maps an input word $w \in \{0,1\}^n$ to much shorter quantum state, its fingerprint $|\psi(w)\rangle = U_w |0^m\rangle$, where U_w is the single transition matrix representing the multiplication of all transition matrices while reading w and $|0^m\rangle = \underbrace{|0\rangle \otimes \cdots \otimes |0\rangle}_{m \text{ times}}$. Quantum fingerprint captures essential properties of the input word that can be useful for computation.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 269–280, doi:10.4204/EPTCS.386.21

One example of quantum fingerprinting applications is the QFA algorithms for MOD_p language [3]. For a given prime number p, the language MOD_p is defined as $MOD_p = \{a^i \mid i \text{ is divisible by } p\}$. Let us briefly describe the construction of the QFA algorithms for MOD_p.

We start with a 2-state QFA M_k, where $k \in \{1, \ldots, p-1\}$. The automaton M_k has two base states $Q = \{q_0, q_1\}$, it starts in the state $|\psi_0\rangle = |q_0\rangle$, and it has the accepting subspace spanned by $|q_0\rangle$. At each step (for each letter) we perform the rotation

$$U_a = \begin{pmatrix} \cos\frac{2\pi k}{p} & \sin\frac{2\pi k}{p} \\ -\sin\frac{2\pi k}{p} & \cos\frac{2\pi k}{p} \end{pmatrix}.$$

It is easy to see that this automaton gives the correct answer with probability 1 if $w \in MOD_p$. However, if $w \notin MOD_p$, the probability of correct answer can be close to 0 rather than 1 (i.e., bounded below by $1 - \cos^2(\pi/p)$). To boost the success probability we use d copies of this automaton, namely M_{k_1}, …, M_{k_d}, as described below.

The QFA M for MOD_p has $2d$ states: $Q = \{q_{1,0}, q_{1,1}, \ldots, q_{d,0}, d_{d,1}\}$, and it starts in the state $|\psi_0\rangle = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} |q_{i,0}\rangle$. In each step, it applies the transformation defined as:

$$|q_{i,0}\rangle \mapsto \cos\frac{2\pi k_i}{p} |q_{i,0}\rangle + \sin\frac{2\pi k_i}{p} |q_{i,1}\rangle \tag{1}$$

$$|q_{i,1}\rangle \mapsto -\sin\frac{2\pi k_i}{p} |q_{i,0}\rangle + \cos\frac{2\pi k_i}{p} |q_{i,1}\rangle \tag{2}$$

Indeed, M enters into equal superposition of d sub-QFAs, and each sub-QFA applies its rotation. Thus, quantum fingerprinting technique associates the input word $w = a^j$ with its fingerprint

$$|\psi\rangle = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} \cos\frac{2\pi k_i j}{p} |q_{i,0}\rangle + \sin\frac{2\pi k_i j}{p} |q_{i,1}\rangle .$$

Ambainis and Nahimovs [3] proved that this QFA accepts the language MOD_p with error probability that depends on the choice of the coefficients k_i's. They also showed that for $d = 2\log(2p)/\varepsilon$ there is at least one choice of coefficients k_i's such that error probability is less than ε. The proof uses a probabilistic method, so these coefficients are not explicit. They also suggest two explicit sequences of coefficients: cyclic sequence $k_i = g^i \pmod p$ for primitive root g modulo p and more complex AIKPS sequences based on the results of Ajtai et al. [2].

While quantum fingerprinting is versatile and has different applications [6, 1], it is not practical for the currently available real quantum computers. The main obstacle is that quantum fingerprinting uses an exponential (in the number m of qubits) circuit depth (e.g., see [11, 5, 15] for some implementations of the aforementioned automaton M). Therefore, the required quantum volume[1] V_Q is roughly $2^{|w|\cdot 2^m}$. For example, IBM reports [8] that its Falcon r5 quantum computer has 27 qubits with a quantum volume of 128. It means that we can use only 7 of 27 qubits for the fingerprint technique.

In this paper, we investigate how to obtain better circuit depth by optimizing the coefficients used by M: $k_1, \ldots, k_d$. We use generalized arithmetic progressions for generating a set of coefficients and show that such sets have a circuit depth comparable to the set obtained by the probabilistic method.

[1] Quantum volume is an exponent of the maximal square circuit size that can be implemented on the quantum computer [7, 18].

Table 1: Comparison of different methods.

Method	Width	Depth	Source	Note
Cyclic	$p^{c/\log\log p}$	$p^{c/\log\log p}$	[3]	for some constant $c > 0$
AIKPS	$\log^{2+3\varepsilon} p$	$(1+2\varepsilon)\log^{1+\varepsilon} p \log\log p$	[14]	
Probabilistic	$4\log(2p)/\varepsilon$	$2\log(2p)/\varepsilon$	[3]	
GAPs	p/ε^2	$\lceil \log p - 2\log\varepsilon \rceil + 2$	this paper	

We summarize the previous and our results in Table 1. Note that p is exponential in the number of qubits m. The depth of the circuits is discussed in Section 3.

The rest of the paper is organized as follows. In Section 2 we give the necessary definitions and results on quantum computation and additive combinatorics to follow the rest of the paper. Section 3 contains the construction of the shallow fingerprinting function and the proof of its correctness. Then, we present certain numerical simulations in Section 4. We conclude the paper with Section 5 by presenting some open questions and discussions for further research.

2 Preliminaries

Let us denote by $\mathscr{H}^2$ two-dimensional Hilbert space, and by $(\mathscr{H}^2)^{\otimes m}$ 2^m-dimensional Hilbert space (i.e., the space of m qubits). We use bra- and ket-notations for vectors in Hilbert space. For any natural number N, we use $\mathbb{Z}_N$ to denote the cyclic group of order N.

Let us describe in detail how the automaton M works. As we outlined in the introduction, the automaton M has $2d$ states: $Q = \{q_{1,0}, q_{1,1}, \ldots, q_{d,0}, d_{d,1}\}$, and it starts in the state $|\psi_0\rangle = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} |q_{i,0}\rangle$. After reading a symbol a, it applies the transformation U_a defined by (1), (2):

$$|q_{i,0}\rangle \mapsto \cos\frac{2\pi k_i}{p} |q_{i,0}\rangle + \sin\frac{2\pi k_i}{p} |q_{i,1}\rangle$$
$$|q_{i,1}\rangle \mapsto -\sin\frac{2\pi k_i}{p} |q_{i,0}\rangle + \cos\frac{2\pi k_i}{p} |q_{i,1}\rangle$$

After reading the right endmarker \$, it applies the transformation $U_\$$ defined in such way that $U_\$ |\psi_0\rangle = |q_{1,0}\rangle$. The automaton measures the final state and accepts the word if the result is $q_{1,0}$.

So, the quantum state after reading the input word $w = a^j$ is

$$|\psi\rangle = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} \cos\frac{2\pi k_i j}{p} |q_{i,0}\rangle + \sin\frac{2\pi k_i j}{p} |q_{i,1}\rangle .$$

If $j \equiv 0 \pmod{p}$, then $|\psi\rangle = |\psi_0\rangle$, and $U_\$$ transforms it into accepting state $|q_{1,0}\rangle$, therefore, in this case, the automaton always accepts. If the input word $w \notin MOD_p$, then the quantum state after reading the right endmarker \$ is

$$|\psi'\rangle = \frac{1}{d}\Big(\sum_{i=1}^{d} \cos\frac{2\pi k_i j}{p}\Big) |q_{1,0}\rangle + \ldots,$$

and the error probability is

$$P_e = \frac{1}{d^2}\Big(\sum_{i=1}^{d} \cos\frac{2\pi k_i x}{p}\Big)^2 .$$

In the rest of the paper, we denote by m the number of qubits in the quantum fingerprint, by $d = 2^m$ the number of parameters in the set K, by p the size of domain of the quantum fingerprinting function, and by $U_a(K)$ the transformation defined above, which depends on the set K.

Let us also define a function $\varepsilon : \mathbb{Z}_p^d \to \mathbb{R}$ as follows:

$$\varepsilon(K) = \max_{x \in \mathbb{Z}_p} \left(\frac{1}{d^2} \Big| \sum_{j=1}^{d} \exp \frac{2\pi i k_j x}{p} \Big|^2 \right).$$

Note that $P_e \leq \varepsilon(K)$.

We also use some tools from additive combinatorics. We refer the reader to the textbook by Tao and Vu [17] for a deeper introduction to additive combinatorics.

An additive set $A \subseteq Z$ is a finite non-empty subset of Z, an abelian group with group operation $+$. We refer Z as the ambient group.

If A, B are additive sets in Z, we define the sum set $A+B = \{a+b \mid a \in A, b \in B\}$. We define additive energy $E(A,B)$ between A, B to be

$$E(A,B) = \Big| \{(a,b,a',b') \in A \times B \times A \times B \mid a+b = a'+b'\} \Big|.$$

Let us denote by $e(\theta) = e^{2\pi i \theta}$, and by $\xi \cdot x = \xi x/p$ bilinear form from $\mathbb{Z}_p \times \mathbb{Z}_p$ into $\mathbb{R}/\mathbb{Z}$. Fourier transform of $f : \mathbb{Z}_p \to \mathbb{Z}_p$ is $\hat{f}(\xi) = \mathbf{E}_{x \in Z} f(x) \overline{e(\xi \cdot x)}$.

We also denote the characteristic function of the set A as 1_A, and we define $\mathbf{P}_Z(A) = \widehat{1_A}(0) = |A|/|Z|$.

Definition 1 ([17]). *Let Z be a finite additive group. If $A \subseteq Z$, we define Fourier bias $\|A\|_{\mathscr{U}}$ of the set A to be*

$$\|A\|_{\mathscr{U}} = \sup_{\xi \in Z \setminus \{0\}} |\widehat{1_A}(\xi)|$$

There is a connection between the Fourier bias and the additive energy.

Theorem 1 ([17]). *Let A be an additive set in a finite additive group Z. Then*

$$\|A\|_{\mathscr{U}}^4 \leq \frac{1}{|Z|^3} E(A,A) - \mathbf{P}_Z(A)^4 \leq \|A\|_{\mathscr{U}}^2 \mathbf{P}_Z(A)$$

Definition 2 ([17]). *Generalized arithmetic progression (GAP) of dimension d is a set*

$$A = \{x_0 + n_1 x_1 + \ldots + n_d x_d \mid 0 \leq n_1 \leq N_1, \cdots, 0 \leq n_d \leq N_d\},$$

where $x_0, x_1, \ldots, x_d, N_1, \ldots, N_d \in Z$. The size of GAP is a product $N_1 \cdots N_d$. If the size of set A, $|A|$, equals to $N_1 \cdots N_d$, we say that GAP is proper.

3 Shallow Fingerprinting

Quantum fingerprint can be computed by the quantum circuit given in Figure 1. The last qubit is rotated by a different angle $2\pi k_j x/q$ in different subspaces enumerated by $|j\rangle$. Therefore, the circuit depth is $|K| = t = 2^m$. As the set K is random, it is unlikely that the depth can be less than $|K|$.

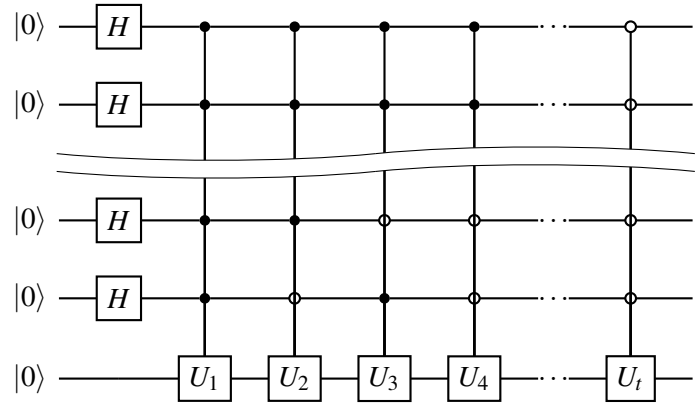


Figure 1: Deep fingerprinting circuit example. Gate U_j is a rotation $R_y(4\pi k_j x/p)$. Controls in controlled gates run over all binary strings of length s

Let us note that fingerprinting is similar to quantum Fourier transform. Quantum Fourier transform computes the following transformation:

$$|x\rangle \mapsto \frac{1}{N}\sum_{k=0}^{N-1} \omega_N^{xk} |k\rangle, \tag{3}$$

where $\omega_N = e(1/N)$. Here is the quantum fingerprinting transform:

$$|x\rangle \mapsto \frac{1}{t}\sum_{j=1}^{t} \omega_N^{k_j x} |k\rangle.$$

The depth of the circuit that computes quantum Fourier transform is $O((\log N)^2)$, and it heavily relies on the fact that in Eq. (3) the sum runs over all $k = 0,\ldots,N-1$. Therefore, to construct a shallow fingerprinting circuit we desire to find a set K with special structure.

Suppose that we construct a coefficient set $K \subset \mathbb{Z}_p$ in the following way. We start with a set $T = \{t_1,\ldots,t_m\}$ and construct the set of coefficients as a set of sums of all possible subsets:

$$K = \Big\{ \sum_{t\in S} t \mid S \subseteq T \Big\},$$

where we sum modulo p.

The quantum fingerprinting function with these coefficients can be computed by a circuit of depth $O(m)$ [9] (see Figure 2).

Finally, let us prove why the construction of the set $K \subset \mathbb{Z}_p$ works.

Theorem 2. *Let $\varepsilon > 0$, let $m = \lceil \log p - 2\log\varepsilon \rceil$ and $d = 2^m$.*

Suppose that the number $t_0 \in \mathbb{Z}_p$ and the set $T = \{t_1,\ldots,t_m\} \subset \mathbb{Z}_p$ are such that

$$B = \{2t_0 + n_1 t_1 + \cdots + n_m t_m \mid 0 \le n_1 < 3, \ldots, 0 \le n_m < 3\}$$

is a proper GAP.

Then the set A defined as

$$A = \left\{ t_0 + \sum_{t\in S} t \mid S \subseteq T \right\}$$

has $\varepsilon(A) \le \varepsilon$.

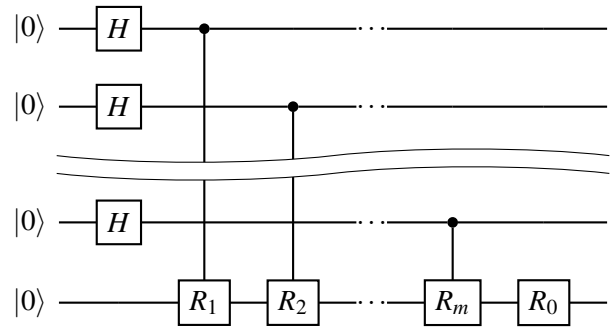


Figure 2: Shallow fingerprinting circuit example. Gate R_j is a rotation $R_y(4\pi t_j x/p)$

Let us outline the proof of this theorem. Firstly, we estimate the number of solutions to $a+b=n$. Secondly, we use it to bound the additive energy $E(A,A)$ of the set A. Thirdly, we bound the Fourier bias $\|A\|_{\mathscr{U}}$. Finally, we get a bound on $\varepsilon(A)$ in terms of p and m.

Proof. Let us denote a set $R_n(A)$ of solutions to $a+b=n$, where $a,b\in A$ and $n\in\mathbb{Z}_p$:

$$R_n(A)=\{(a,b)\mid a+b=n;\ a,b\in A\}.$$

Note that we have $E(A,A)=\sum_{n\in Z}R_n(A)^2$.

Suppose that n is represented as $n=2t_0+\sum_{i=1}^{m}\gamma_i t_i$, $\gamma_i\in\{0,1,2\}$. If such representation exists, it is unique, because B is a proper GAP. Let us denote $c_0:=\{i\mid\gamma_i=0\}$, $c_1:=\{i\mid\gamma_i=1\}$, $c_2:=\{i\mid\gamma_i=2\}$. It is clear that $c_0\uplus c_1\uplus c_2=[m]$.

Now suppose that $n=a+b$ for some $a,b\in A$. But $a=t_0+\sum_i\alpha_i t_i$ and $b=t_0+\sum_i\beta_i t_i$, $\alpha_i,\beta_i\in\{0,1\}$. We get that if $i\in c_0$ or $i\in c_2$ then the corresponding coefficients α_i and β_i are uniquely determined. Consider $i\in c_1$. Then we have two choices: either $\alpha_i=1;\beta_i=0$, or $\alpha_i=0;\beta_i=1$. Therefore, we have $R_n(A)=2^{|c_1(n,A)|}$.

We have that

$$E(A,A)=\sum_{n\in Z}R_n(A)^2=\sum_{n\in Z}2^{2|c_1(n,A)|}.$$

Using the fact that $|c_0(n,A)|+|c_1(n,A)|+|c_2(n,A)|=m$, we see that

$$E(A,A)=\sum_{n\in Z}2^{2|c_1(n,A)|}=\sum_{j=0}^{m}\binom{m}{j}2^{m-j}2^{2j}=\sum_{j=0}^{m}\binom{m}{j}2^{m+j}\le 2^{3m}$$

We can bound the Fourier bias by Theorem 1:

$$\|A\|_{\mathscr{U}}^4\le\frac{1}{|Z|^3}E(A,A)-\mathbf{P}_Z(A)^4\le\|A\|_{\mathscr{U}}^2\,\mathbf{P}_Z(A)$$

$$\|A\|_{\mathscr{U}}^4\le\frac{2^{3m}}{2^{3\cdot 2^m}}-\frac{2^{4m}}{2^{4\cdot 2^m}}=\frac{d^3}{2^{3d}}-\frac{d^4}{2^{4d}}$$

$$\|A\|_{\mathscr{U}}\le\frac{d^{3/4}}{p^{3/4}}$$

Finally, we have

$$\varepsilon(A)=\left(\frac{p}{d}\|A\|_{\mathscr{U}}\right)^2\le\frac{p^{1/2}}{d^{1/2}}.$$

By substituting the definitions of d and m, we prove the theorem. □

Corollary 1. *The depth of the circuit that computes $U_a(A)$ is $\lceil \log p - 2\log\varepsilon \rceil$.*

Theorem 3 (Circuit depth for AIKPS sequences)**.** *For given $\varepsilon > 0$, let*

$$R = \{r \mid r \text{ is prime}, (\log p)^{1+\varepsilon}/2 < r < (\log p)^{1+\varepsilon}\},$$
$$S = \{1, 2, \ldots, (\log p)^{1+2\varepsilon}\},$$
$$T = \{s \cdot r^{-1} \mid r \in R, s \in S\},$$

where r^{-1} is the inverse of r modulo p.

Then the depth of the circuit that computes $U_a(T)$ is less than $(1+2\varepsilon)\log^{1+\varepsilon} p \, \log\log p$.

Proof. Let us denote the elements of R by $r_1, r_2, \ldots$. Let $S \cdot \{r^{-1}\}$ be a set $\{s \cdot r^{-1} \mid s \in S\}$.

Consider the following circuit $\mathscr{C}_j$ (see Figure 3) with $w = \lceil (1+2\varepsilon)\log\log p \rceil + 1$ wires.

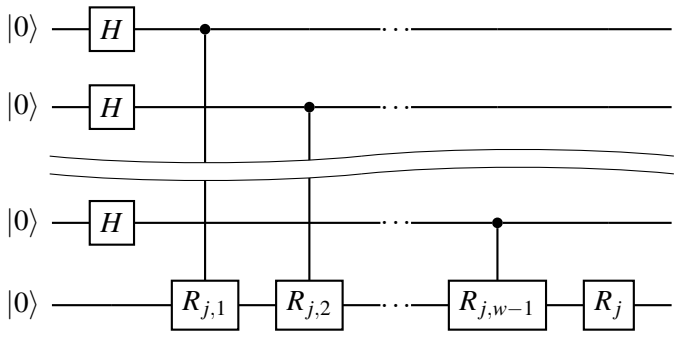


Figure 3: Circuit $\mathscr{C}_j$ for AIKPS subsequence. Gate R_j is a rotation $R_y(4\pi(r_j^{-1})/p)$. Gate $R_{j,k}$ is a rotation $R_y(2^{k-1} \cdot 4\pi(r_j^{-1})/p)$

The circuit $\mathscr{C}_j$ has depth $\lceil (1+2\varepsilon)\log\log p \rceil + 1$ and computes the transformation $U_a(S \cdot \{r_j^{-1}\})$. By repeating the same circuit for all $r_j \in R$ we get the required circuit for $U_a(T)$ (see Figure 4).

Since $|R| < (\log p)^{1+\varepsilon}$, we obtain that the depth of the circuit $U_a(T)$ is less than

$$(1+2\varepsilon)\log^{1+\varepsilon} p \, \log\log p. \qquad \square$$

4 Numerical Experiments

We conduct the following numerical experiments. We compute sets of coefficients K for the automaton for the language MOD_p with minimal computational error.

Finding an optimal set of coefficients is an optimization problem with many parameters, and the running time of a brute force algorithm is large, especially with an increasing number m of control qubits and large values of parameter p. Then, the original automaton has $2d$ states, where $d = 2^m$. We observe circuits for several m values and use a heuristic method for finding the optimal sets K with respect to an error minimization. For this purpose, the coordinate descend method [19] is used.

We find an optimal sets of coefficients for different values of p and m and compare computational errors of original and shallow fingerprinting algorithms for the automaton (see Figure 5). Namely, we set $m = 3, 4, 5$ and find sets using the coordinate descend method for each case. Even heuristic computing, for $s > 5$, takes exponentially more computational time and it is hard to implement on our devices.

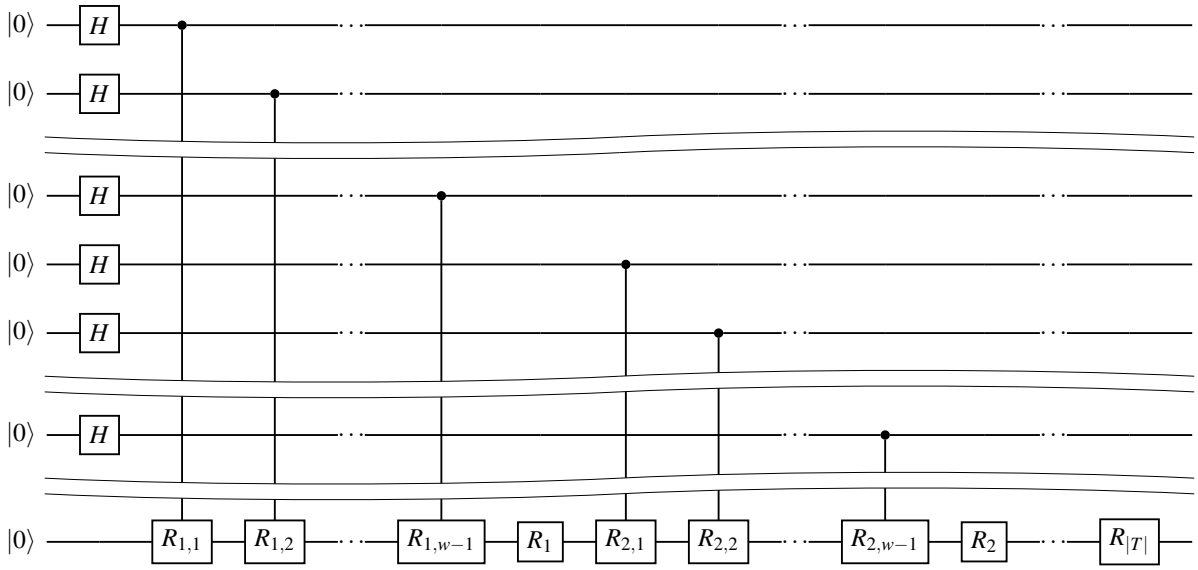


Figure 4: Circuit for $U_a(T)$. Gate R_j is a rotation $R_y(4\pi(r_j^{-1})/p)$. Gate $R_{j,k}$ is a rotation $R_y(2^{k-1} \cdot 4\pi(r_j^{-1})/p)$

One can note that difference between errors becomes bigger with increasing m, especially for big values p. The program code and numerical data are presented in a git repo [10].

The graphics in Figure 6 show a proportion of the errors of the original automaton over the errors of the shallow automaton for $m = 3,4,5$ and the prime numbers until 1013.

As we see, for a number of control qubits $m = 3$, the difference between the original and shallow automata errors is approximately constant. The ratio of values fluctuates between 1 and 1.2. In the case $m = 4$, this ratio is approximately 1.5 for almost all observed values p. The ratio of errors is nearly between 1.5 and 3, for $m = 5$.

According to the results of our experiments, the circuit depth $m+1$ is enough for valid computations, while the original circuit uses $O(2^m)$ gates. Since the shallow circuit is much simpler than the original one, its implementation on real quantum machines is much easier. For instance, in such machines as IBMQ Manila or Baidu quantum computer, a "quantum computer" is represented by a linearly related sequence of qubits. CX-gates can be applied only to the neighbor qubits. For such a linear structure of qubits, the shallow circuit can be implemented using $3m+3$ CX-gates. Whereas a nearest-neighbor decomposition [13] of the original circuit requires $O(d \log d) = O(m2^m)$ CX-gates.

5 Conclusions

We show that generalized arithmetic progressions generate some sets of coefficients k_i for the quantum fingerprinting technique with provable characteristics. These sets have large sizes, however, their depth is small and comparable to the depth of sets obtained by the probabilistic method. These sets can be used in the implementations of quantum finite automata suitable for running on the current quantum hardware.

We run numerical simulations. They show that the actual performance of the coefficients found by our method for quantum finite automata is not much worse than the performance of the other methods.

Optimizing quantum finite automata implementation for depth also poses an open question. The

Figure 5: Computational errors for $m = 3, 4, 5$ of original and shallow automata

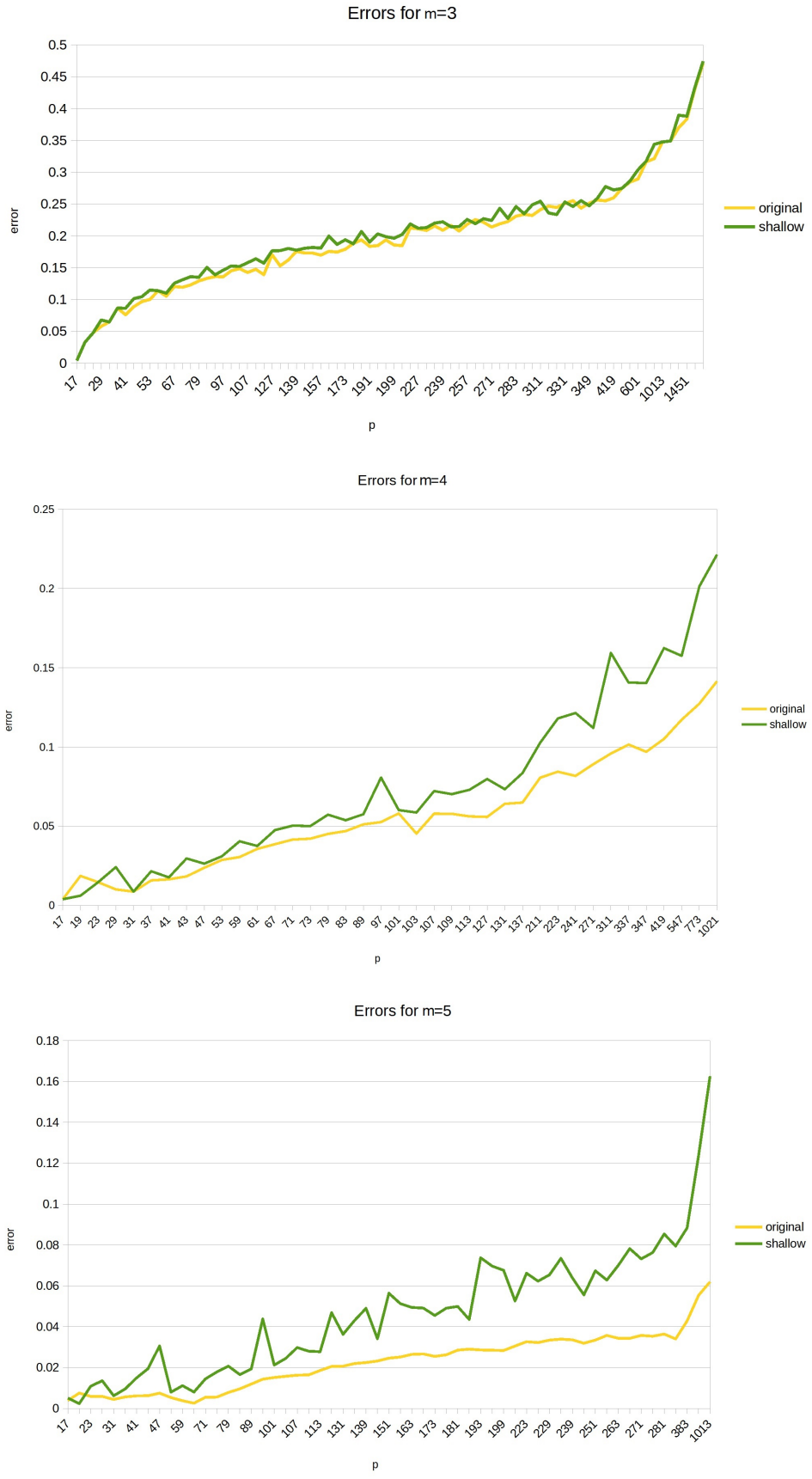
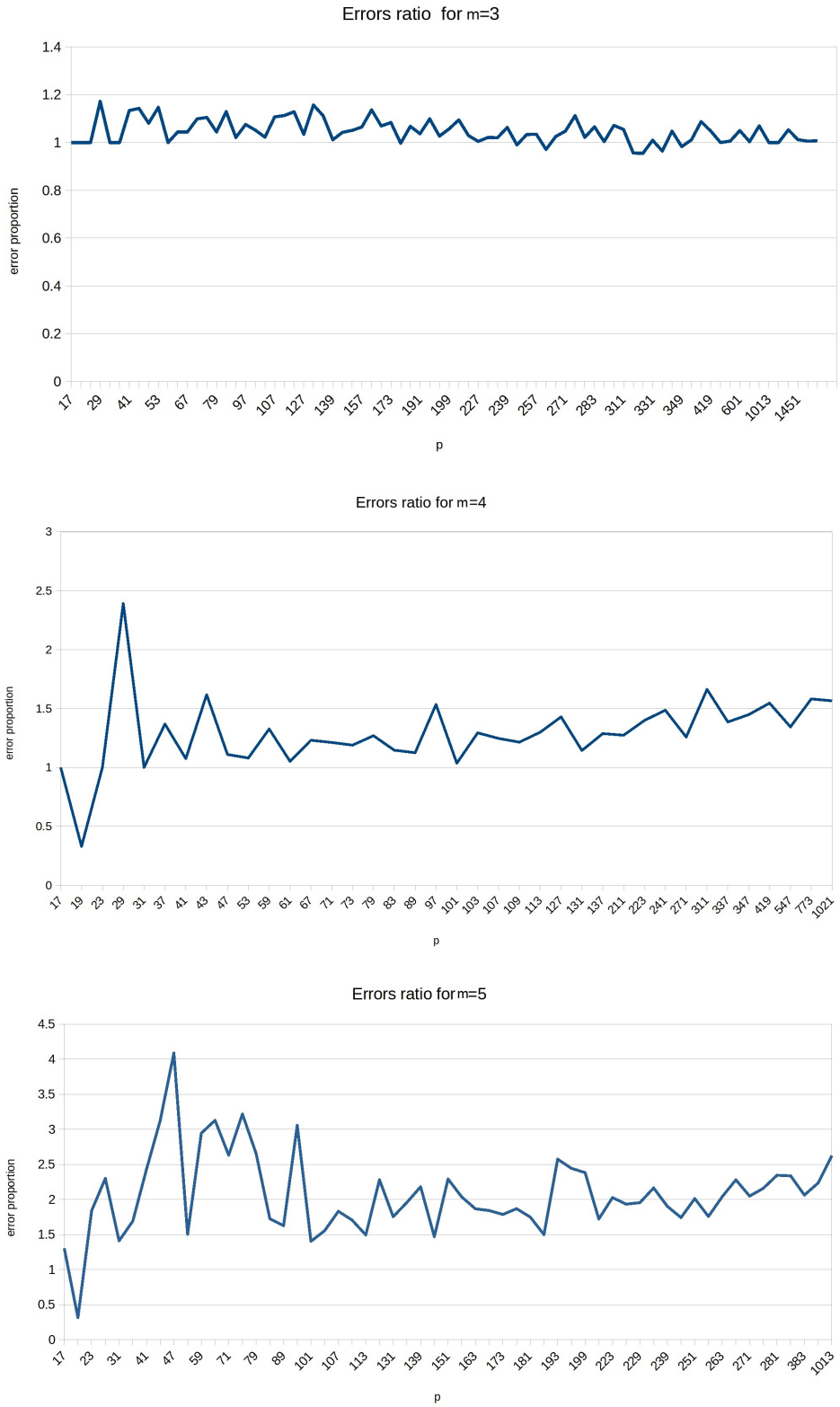


Figure 6: Proportions of the shallow automaton errors over the original automaton errors for $m = 3,4,5$ and different values of p



lower bound for the size of K in terms of p and ε is known [1]. Therefore, for given p and ε, quantum finite automata cannot have less than $O(\log p/\varepsilon)$ states. But, to our knowledge, a lower bound for the circuit depth of the transition function implementation is not known. So, we pose an open question: is it possible to implement a transition function with depth less than $O(\log p)$? What is the lower bound for it?

6 Acknowledgments

Yakaryılmaz was partially supported by the ERDF project Nr. 1.1.1.5/19/A/005 "Quantum computers with constant memory" and the project "Quantum algorithms: from complexity theory to experiment" funded under ERDF programme 1.1.1.5.

This paper has been supported by the Kazan Federal University Strategic Academic Leadership Program ("PRIORITY-2030"). Research in Section 4 were supported by the subsidy allocated to Kazan Federal University for the state assignment in the sphere of scientific activities, project No. 0671-2020-0065.

References

[1] Farid Ablayev, Marat Ablayev, Alexander Vasiliev & Mansur Ziatdinov (2016): *Quantum Fingerprinting and Quantum Hashing. Computational and Cryptographical Aspects*. Baltic Journal of Modern Computing 4(4), pp. 860–875, doi:10.22364/bjmc.2016.4.4.17.

[2] Miklós Ajtai, Henryk Iwaniec, János Komlós, János Pintz & Endre Szemerédi (1990): *Construction of a thin set with small Fourier coefficients*. Bulletin of the London Mathematical Society 22(6), pp. 583–590, doi:10.1112/blms/22.6.583.

[3] Andris Ambainis & Nikolajs Nahimovs (2009): *Improved constructions of quantum automata*. Theoretical Computer Science 410(20), pp. 1916–1922, doi:10.1016/j.tcs.2009.01.027.

[4] Andris Ambainis & Abuzer Yakaryılmaz (2021): *Automata and quantum computing*. In Jean Éric Pin, editor: Handbook of Automata Theory, chapter 39, 2, European Mathematical Society Publishing House, pp. 1457–1493, doi:10.4171/Automata-2/17.

[5] Utku Birkan, Özlem Salehi, Viktor Olejar, Cem Nurlu & Abuzer Yakaryılmaz (2021): *Implementing Quantum Finite Automata Algorithms on Noisy Devices*. In: International Conference on Computational Science, Springer, pp. 3–16, doi:10.1007/978-3-030-77980-1_1.

[6] Harry Buhrman, Richard Cleve, John Watrous & Ronald de Wolf (2001): *Quantum Fingerprinting*. Physical Review Letters 87(16), p. 167902, doi:10.1103/PhysRevLett.87.167902. arXiv:0102001.

[7] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation & Jay M. Gambetta (2019): *Validating quantum computers using randomized model circuits*. Physical Review A 100, p. 032328, doi:10.1103/PhysRevA.100.032328.

[8] IBM (2022): *Eagle's quantum performance progress*. Available at `https://research.ibm.com/blog/eagle-quantum-processor-performance`.

[9] Martin Kālis (2018): *Kvantu Algoritmu Realizācija Fiziskā Kvantu Datorā (Quantum Algorithm Implementation on a Physical Quantum Computer)*. Master's thesis, University of Latvia.

[10] Aliya Khadieva: *Optimal Parameters Computing Code*. Available at `https://github.com/aliyakhadi/Parameters_counting`.

[11] Aliya Khadieva & Mansur Ziatdinov (2023): *Deterministic Construction of QFAs Based on the Quantum Fingerprinting Technique*. Lobachevskii Journal of Mathematics 44(2), pp. 713–723, doi:10.1134/S199508022302021X.

[12] Cristopher Moore & James P Crutchfield (2000): *Quantum automata and quantum grammars*. *Theoretical Computer Science* 237(1-2), pp. 275–306, doi:10.1016/S0304-3975(98)00191-1.

[13] Mikka Möttönen & Juha J Vartiainen (2006): *Decompositions of general quantum gates*. *Trends in Quantum Computing Research*, doi:10.48550/ARXIV.QUANT-PH/0504100.

[14] Alexander Razborov, Endre Szemerédi & Avi Wigderson (1993): *Constructing small sets that are uniform in arithmetic progressions*. *Combinatorics, Probability and Computing* 2(4), pp. 513–518, doi:10.1017/S0963548300000870.

[15] Özlem Salehi & Abuzer Yakaryılmaz (2021): *Cost-efficient QFA Algorithm for Quantum Computers*, doi:10.48550/arXiv.2107.02262.

[16] A. C. Cem Say & Abuzer Yakaryılmaz (2014): *Quantum finite automata: A modern introduction*. In: *Computing with New Resources*, Springer, pp. 208–222, doi:10.1007/978-3-319-13350-8_16.

[17] Terence Tao & Van Vu (2006): *Additive combinatorics*. *Cambridge Studies in Advanced Mathematics* 105, Cambridge University Press, doi:10.1017/CBO9780511755149.

[18] Andrew Wack, Hanhee Paik, Ali Javadi-Abhari, Petar Jurcevic, Ismael Faro, Jay M. Gambetta & Blake R. Johnson (2021): *Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers*, doi:10.48550/ARXIV.2110.14108.

[19] Stephen J Wright (2015): *Coordinate descent algorithms*. *Mathematical programming* 151(1), pp. 3–34, doi:10.1007/s10107-015-0892-3.