

**Tenth Workshop on  
Non-Classical Models of  
Automata and Applications  
(NCMA 2018)**

books@ocg.at  
BAND 332

**Wissenschaftliches Redaktionskomitee**

o.Univ.Prof. Dr. Gerhard Chroust  
Univ.Prof. Dr. Gabriele Kotsis  
Univ.Prof. DDr. Gerald Quirchmayr  
DI Dr. Peter M. Roth  
Univ.Prof. DDr. Erich Schweighofer  
Univ.Prof. Dr. Jörg Zumbach

Rudolf Freund, Michal Hospodár, Galina Jirásková, and Giovanni Pighizzini  
(eds.)

**Tenth Workshop on  
Non-Classical Models of  
Automata and Applications  
(NCMA 2018)**

© Österreichische Computer Gesellschaft  
Komitee für Öffentlichkeitsarbeit  
[www.ocg.at](http://www.ocg.at)

Druck: Druckerei Riegelnik  
1080 Wien, Piaristengasse 19

ISBN 978-3-903035-21-8

# Preface

The *Tenth Workshop on Non-Classical Models of Automata and Applications (NCMA 2018)* was held in Košice, Slovakia, on August 21<sup>st</sup> and 22<sup>nd</sup>, 2018. The NCMA workshop series was established in 2009 as an annual forum for researchers working on different aspects of non-classical and classical models of automata and grammars. The purpose of the NCMA workshop series is to provide an opportunity to exchange and develop novel ideas, and to stimulate research on non-classical and classical models of automata and grammar-like structures. Many models of automata and grammars are studied from different points of view in various areas, both as theoretical concepts and as formal models for applications. The goal of the NCMA workshop series is to motivate a deeper coverage of this particular area and in this way to foster new insights and substantial progress in computer science as a whole.

The previous workshops took place in the following places:

2009 Wrocław, Poland,  
2010 Jena, Germany,  
2011 Milano, Italy,  
2012 Fribourg, Switzerland,  
2013 Umeå, Sweden,  
2014 Kassel, Germany,  
2015 Porto, Portugal,  
2016 Debrecen, Hungary, and  
2017 Praha, Czech Republic.

The Tenth Workshop on Non-Classical Models of Automata and Applications (NCMA 2018) was organized by the Košice branch of the Mathematical Institute of the Slovak Academy of Sciences. Its scientific program consisted of invited lectures, regular contributions, and short presentations.

At NCMA 2018 there were two invited lectures:

- Bruno Guillon (Università degli Studi di Milano, Dipartimento di Informatica, Italy):  
*On Nondeterministic Two-way Transducers*  
and
- José M. Sempere (DSIC, Universitat Politècnica de València, Spain):  
*On the Application of Watson-Crick Finite Automata for the Resolution of Bioinformatic Problems.*

We thank Bruno Guillon and José M. Sempere for accepting our invitation and for presenting their recent results.

For NCMA 2018, we received submissions by a total of 32 authors, from 11 different countries. On the basis of at least three referees' reports each, the Program Committee selected 11 contributions for presentation at NCMA 2018 and for inclusion in the workshop proceedings.

We thank the members of the Program Committee for their excellent work in making this selection:

- Suna Bensch (University of Umeå, Sweden),
- Cezar Câmpeanu (University of Prince Edward Island, Charlottetown, Canada),
- Erzsébet Csuhaj-Varjú (Eötvös Loránd University, Budapest, Hungary),
- Dora Giammarresi (University of Rome Tor Vergata, Rome, Italy),
- Mika Hirvensalo (University of Turku, Finland),
- Szabolcs Iván (University of Szeged, Hungary),
- Galina Jirásková (Slovak Academy of Sciences, Košice, Slovakia), co-chair,
- Ian McQuillan (University of Saskatchewan, Saskatoon, Canada),
- Nelma Moreira (University of Porto, Portugal),
- František Mráz (Charles University in Prague, Czech Republic),
- Alexander Okhotin (St. Petersburg State University, Russia),
- Meenakshi Paramasivan (University of Trier, Germany),
- Dana Pardubská (Comenius University in Bratislava, Slovakia)
- Giovanni Pighizzini (University of Milan, Italy),
- Bianca Truthe (University of Giessen, Germany),
- György Vaszil (University of Debrecen, Hungary),
- Mikhail Volkov (Ural Federal University, Yekaterinburg, Russia),
- Matthias Wendlandt (University of Giessen, Germany).

We also thank the following colleagues for helping in the evaluation process by providing external reviews:

- Jozef Jirásek (University of Saskatchewan, Saskatoon, Canada)
- Luca Prigioniero (University of Milan, Italy).

In addition to the invited talks and regular contributions, NCMA 2018 also featured five short presentations to emphasize its workshop character, each of them also having been evaluated by at least two members of the Program Committee.

This volume contains the two invited presentations and the eleven regular contributions. Extended abstracts of the short papers presented at NCMA 2018 appear in a separate booklet.

A special issue of a renowned scientific journal dedicated to NCMA 2018 will also be edited after the workshop, and it will contain extended versions of selected papers, which will undergo the standard refereeing process of the journal.

We are grateful to the Košice branch of the Mathematical Institute of the Slovak Academy of Sciences for the local organization and for the financial support of NCMA 2018.

August 2018

Rudolf Freund, Wien  
Michal Hospodár, Košice  
Galina Jirásková, Košice  
Giovanni Pighizzini, Milano





# Table of Contents

## Invited Papers

ON NONDETERMINISTIC TWO-WAY TRANSDUCERS .....	11
-----------------------------------------------	----

*Bruno Guillon*

ON THE APPLICATION OF WATSON-CRICK FINITE AUTOMATA FOR THE RESOLUTION OF BIOINFORMATIC PROBLEMS (Extended Abstract) ....	29
-----------------------------------------------------------------------------------------------------------------------------	----

*José M. Sempere*

## Regular Contributions

CAUSAL DYNAMICS OF DISCRETE MANIFOLDS .....	31
---------------------------------------------	----

*Pablo Arrighi, Clément Chouteau, Stefano Facchini, and Simon Martiel*

ON REGULAR EXPRESSIONS WITH BACKREFERENCES AND TRANSDUCERS	49
------------------------------------------------------------	----

*Martin Berglund, Frank Drewes, and Brink van der Merwe*

POSTSELECTING PROBABILISTIC FINITE STATE RECOGNIZERS AND VERIFIERS.....	65
----------------------------------------------------------------------------	----

*Maksims Dimitrijevs and Abuzer Yakaryilmaz*

AUTOMATA THAT MAY CHANGE THEIR MIND .....	83
<i>Markus Holzer and Martin Kutrib</i>	
FORBIDDEN PATTERNS FOR ORDERED AUTOMATA .....	99
<i>Ondřej Klíma and Libor Polák</i>	
A JUMPING 5' → 3' WATSON-CRICK FINITE AUTOMATA MODEL .....	117
<i>Radim Kocman, Benedek Nagy, Zbyněk Křivka, and Alexander Meduna</i>	
TWO-SIDED LOCALLY TESTABLE LANGUAGES .....	133
<i>Martin Kutrib and Friedrich Otto</i>	
CHARACTERIZATIONS OF LRR- LANGUAGES BY CORRECTNESS- PRESERVING COMPUTATIONS .....	149
<i>František Mráz, Friedrich Otto, and Martin Plátek</i>	
NETWORKS OF EVOLUTIONARY PROCESSORS WITH RESOURCES RESTRICTED FILTERS .....	165
<i>Bianca Truthe</i>	
JUMPING RESTARTING AUTOMATA .....	181
<i>Qichao Wang and Yongming Li</i>	
ONE-WAY TOPOLOGICAL AUTOMATA AND THE TANTALIZING EFFECTS OF THEIR TOPOLOGICAL FEATURES .....	197
<i>Tomoyuki Yamakami</i>	
<b>Author Index</b> .....	215

# ON NONDETERMINISTIC TWO-WAY TRANSDUCERS

Bruno Guillon

Università degli Studi di Milano, Dipartimento di Informatica, Italy  
guillon.bruno+cs@gmail.com

## **Abstract**

*We study binary relations on words, namely transductions, that are computed by different kinds of transducers, beyond the rational transductions. Our main focus is the class of transductions that are realized by two-way nondeterministic transducers. While determinism (or unambiguity) yields a robust class of word-to-word functions, the situation is more complex in the case of nondeterministic transducers. We discuss two approaches to describe nonfunctional transductions realized by two-way transducers.*

*The first approach is algebraic. We introduce natural operators that mimic the abilities of two-way transducers in a similar way as rational operations mimic abilities of one-way transducers. These operations are sufficient to capture some families of transductions realized by restricted versions of transducers, e.g., sweeping transducers and unary transducers.*

*The second approach is obtained by enriching the semantics of transducers. We consider transductions with origin information. We briefly discuss how this enriched semantics helps in recovering some decidability results, and in describing classes of nonfunctional transductions.*

## 1. Introduction

In the theory of automata two different terms designate more or less indifferently the same object: transductions and binary relations. The former term implicitly distinguishes an input and an output, even if the input does not uniquely determine the output. In certain contexts, it is a synonym for translation where one source and one target are understood. The latter term is meant to suggest pairs of words playing a symmetric role.

Natural models for implementing transductions and relations are transducers and two-tape automata, respectively. Both are finite state automata provided with an additional tape. The concept of multi-tape and thus in particular two-tape automata was introduced by Rabin and Scott [29] and Elgot and Mezei [19], almost sixty years ago. Most closure and structural properties were published in the next couple of years.

On the other hand, transduction is a more suitable term when the intention is that the input preexists the output. Transducers implement this idea. Indeed, they are similar to two-tape

automata, but their computations start with empty content on their second tape (namely the output tape), and proceed on the first tape (namely the input tape) while emitting output symbols that are appended to the output tape content. This difference is almost irrelevant in the case of one-way devices (*i.e.*, when the tape heads proceed rightward only), since there exists an obvious direct translation from one model to the other.<sup>1</sup> It makes however sense to differentiate the models when the two tapes are not processed in the same way.

In this presentation, we focus on two-way transducers which are such models of machine using two tapes. An input tape is read-only, it initially contains the input word, and it is scanned in both directions. An output tape is write-only, initially empty, and it is explored in one direction only.

The dynamics of two-way transducers is complex since such a device may admit computations of unbounded length. The attempts to describe the class of transductions they realize can be grouped in four families:

1. studying some syntactic restrictions, *e.g.*, sweeping transducers;
2. studying some semantic restrictions, *e.g.*, unambiguous, or functional transducers;
3. studying the special case of unary alphabets, for which some specific techniques may apply;
4. enriching of the semantics of the device in order to make precise how the output word is related to the input word. This is the direction followed by Bojańczyk when introducing origin information [8].

This present survey mainly deals with the two last directions. A special attention is paid to the unary cases, while the origin semantics of transducers is briefly discussed at the end of the paper. For the sake of completeness, let us first recall the main known results from the first and the second directions.

**Rational transductions.** One-way transducers are now considered as restrictions of two-way transducers. The transductions they realize have been widely studied and characterized as the *rational transductions*, namely the rational subsets of the direct product of free monoids [19]. We point out that, even in this one-way case, some intractability arises from nondeterminism. Mainly, natural problems such as equivalence or intersection emptiness have been shown to be undecidable by Griffiths [24]. We refer the reader to [7] for further results on one-way transducers; see also [21] and the references therein for connections with logic and algebra.

**“Regular” transductions.** Functional two-way transducers have received a lot of attention in the last decades. This attractiveness comes from the robustness of the class of transductions they define. Following [1], we call it the *class of regular transductions*. The probably most important property witnessing the robustness of the class, is its closure under composition,

---

<sup>1</sup>The difference might be relevant when considering deterministic versions. Deterministic two-tape automata are indeed more expressive than deterministic (or sequential) transducers, since the latter has access only to the next input symbol when determining the next transition to perform, while the former has also access to the next “output” symbol. In particular, transductions realized by deterministic transducers are functions.

obtained by Chytil and Jákł [12].<sup>2</sup> Gurari then proved that the equivalence of two-way deterministic transducers is decidable, thus contrasting with the results of Griffiths. Another key result on the topic, obtained by Engelfriet and Hooġboom, relates two-way transducers with the *monadic second order logic*, *i.e.*, first-order logic extended with quantification over sets of positions [20]. The authors indeed showed that, in the functional case, the model is effectively equivalent to MSO *transductions*, an abstract formalism of graph transformation introduced by Courcelle, that uses monadic second order logic as bottom level of specification, see [13]. A consequence of their proof is that two-way deterministic transducers are as expressive as functional two-way nondeterministic transducers. This again contrasts with the one-way case. Alur and Černý then proposed an alternative equivalent model, called *streaming string transducer*, which processes the input rightward while preparing the output on finitely many registers that can be concatenated in a copyless way [1]. More recently, Dartois *et al.* proved that each two-way transducer admits an equivalent reversible<sup>3</sup> one [16], which allows to improve some of the previously-known results. The problem of deciding given a two-way transducer whether it admits an equivalent one-way transducer has been considered in [22, 5], while uniformization results have been obtained in [31, 16].

**Outline.** In this work we consider different approaches to describe nonfunctional transductions realized by two-way transducers. In Section 2 are gathered basic definitions and typical examples. Then, in Section 3, a particular attention is paid to some syntactical restrictions of two-way nondeterministic transducers, namely to rotating and sweeping transducers, for which an algebraic characterization can be obtained [27, 26]. We prove in Section 4 that this characterization is also sufficient to capture the expressiveness of unary transducers, *i.e.*, the special case of single-letter input and output alphabets [10]. This however fails when one of the alphabets contains two letters [25]. Finally, we briefly discuss origin semantics of transducers in Section 5, and its contribution in describing nonfunctional transductions.

## 2. Preliminaries

We assume the reader is familiar with language and automata theory. For the sake of completeness, we briefly recall some notions and fix notations.

An *alphabet*  $\Sigma$  is a finite set of *symbols*. The *free monoid* it generates is denoted  $\Sigma^*$ , and its elements are *words* over  $\Sigma$  including the *empty word*  $\epsilon$ . The *length* of a word  $u$  is  $|u|$ , and for  $i = 1, \dots, |u|$ , the  $i$ -th symbol of  $u$  is  $u_i$ . The *reverse* of a word  $u = u_1 \cdots u_n$  is the word  $\bar{u} = u_n \cdots u_1$ . The *concatenation* of two words  $u$  and  $v$  is denoted  $uv$ , and the  $n$ -th *power* of  $u$  is denoted  $u^n$ . A *language* is a set of words, *i.e.*, a subset of  $\Sigma^*$ .

In this presentation, we are interested in transductions, *i.e.*, binary relations on words. Throughout the paper, we fix an *input alphabet*  $\Sigma$  and an *output alphabet*  $\Delta$ . A *transduction* over  $\Sigma, \Delta$  is

---

<sup>2</sup>In the sense of functional composition: a second transducer takes as input the output of a first one.

<sup>3</sup>Namely, whose underlying automaton is deterministic and co-deterministic.

a subset of  $\Sigma^* \times \Delta^*$ . It is *functional* if it is a (partial) function, namely, if for each word  $u \in \Sigma^*$  there exists at most one word  $v \in \Delta^*$  such that  $(u, v)$  belongs to the transduction.

Given a monoid  $M$ , the family of its *rational subsets*, denoted  $\text{RAT}(M)$ , is the least family  $\mathcal{F}$  of subsets containing finite sets and closed under

- *set union*:  $X, Y \in \mathcal{F} \implies X \cup Y \in \mathcal{F}$ ;
- *set product*:  $X, Y \in \mathcal{F} \implies XY \in \mathcal{F}$ ;
- *Kleene star*:  $X \in \mathcal{F} \implies X^* \in \mathcal{F}$ .

Recall that  $XY = \{xy \mid x \in X \text{ and } y \in Y\}$  and  $X^* = \{1\} \cup X \cup \dots \cup X^i \cup \dots$  where 1 is the unit of the monoid. Here, we are interested in the case where  $M$  is a free monoid or the direct product of free monoids, namely in *rational languages* and *rational transductions*, respectively.

**Automata.** A *two-way automaton* is a finite state machine which has read-only access to the input tape on which is written the *input word* surrounded by the *left* ( $\triangleright$ ) and the *right* ( $\triangleleft$ ) *endmarkers* not belonging to  $\Sigma$ . Starting in the initial state from the leftmost tape cell (*i.e.*, on  $\triangleright$ ) the automaton *accepts* the input word if it eventually enters a final state while scanning the rightmost tape cell (*i.e.*, on  $\triangleleft$ ) The set of input words accepted by the automaton is the *language accepted*. Two automata are *equivalent* if they accept the same language. There are different kinds of automata, depending on the constrained they satisfy.

*One-way automata* can read the input tape only once and only from left to right. *Rotating automata* can read the input tape several times, but only from left to right. Clearly, they can be presented as a restriction of two-way automata. *Sweeping automata* can read the whole input tape several times, from left-to-right and from right-to-left. Their input head may indeed change direction only when scanning an endmarker. An automaton is *deterministic* if at any time, the next transition to be performed can be decided uniquely from the information of its current state, and the symbol currently read by the input head. Otherwise, it is *nondeterministic*. *Unambiguous automata* are particular nondeterministic automata which admit at most one accepting computation on every input.

**Transducers.** *Transducers* extend automata by providing a way to produce an *output word* from the input word, thus defining transductions. A natural way to define transducers is to associate an automaton with a *production function* that maps each transition of the automaton to some kind of output. This definition allows in particular to consider different kinds of transducers according to the different kinds of underlying automata, *e.g.*, one-way or two-way transducers. Usually, the output associated with a transition is a word or a letter. A more general definition maps transitions into rational languages over the output alphabet  $\Delta$ . This corresponds to see transducers as a particular case of weighted automata, namely as *weighted automata* over the semiring of rational languages over  $\Delta$ , see, *e.g.*, [30, 27]. These different possible definitions do not alter the expressiveness of the model in general as long as stationary moves and empty productions are allowed. According to its underlying automaton, a transducer may be *one-way*, *rotating*, *sweeping*, *deterministic* or *unambiguous*.

**Examples.** A typical example of transduction realized by a one-way transducer, is the *identity transduction*, assuming  $\Sigma = \Delta$ , which is functional and rational:

$$\text{IDENTITY} = \{(u, u) \mid u \in \Sigma^*\} = \{(\sigma, \sigma) \mid \sigma \in \Sigma\}^*.$$

Another functional rational transduction, which is here instrumental, is the ERASE function that maps every word to the empty word:

$$\text{ERASE} = \{(u, \epsilon) \mid u \in \Sigma^*\} = \{(\sigma, \epsilon) \mid \sigma \in \Sigma\}^*.$$

A nonfunctional rational example is the relation *subword*, which, to any input word, associates its (not necessarily connected) subwords:

$$\text{SUBWORD} = \{(u, v) \mid v \text{ is a subword of } u \in \Sigma^*\} = (\text{IDENTITY} \cup \text{ERASE})^*.$$

Rotating transducers are more expressive than one-way transducers, even in the case of functions. A functional transduction witnessing this separation is the *squaring* function, which maps an input word to its square:

$$\text{SQUARE} = \{(u, uu) \mid u \in \Sigma^*\}.$$

This transduction is not rational whence no one-way transducer realize it. Another interesting transduction realized by a rotating transducer is the power transduction, which associate the powers of a word to itself:

$$\text{POWER} = \{(u, v) \mid u \in \Sigma^*, v = u^n \text{ for some integer } n\}.$$

Lastly, we can show that sweeping transducers are more expressive than rotating ones. Indeed, the *mirror* functional transduction that maps every input word  $u$  to its reverse  $\bar{u}$ , is realized by a sweeping transducer, but by no rotating transducer:

$$\text{MIRROR} = \{(u, \bar{u}) \mid u \in \Sigma^*\}.$$

**Rational transductions.** *Rational operations* (namely union, componentwise concatenation, and Kleene star) are well-suited for one-way transducers. In particular, given two such transducers, one can easily obtain a one-way transducer for their union, their componentwise concatenation, or the Kleene star of one of them. Elgot and Mezei proved that these operations are sufficient to characterize the transductions realized by one-way transducers [19].

**Theorem 2.1** *A transduction is realized by a one-way transducer if and only if it is rational.*

This is not any longer the case for two-way transducers, for which no such characterization is known, *cf.* Table 1.<sup>4</sup> The next sections are devoted to attempts to describe the corresponding class of transductions.

---

<sup>4</sup>For regular transductions, namely functional transductions realized by two-way transducers, a characterization of similar flavour has been obtained, by introducing new operations (some of them being considered in the next session), and by semantically restricting componentwise concatenation and Kleene star to unambiguous concatenations only [3, 17, 6].

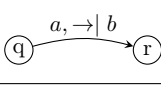
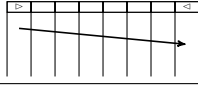
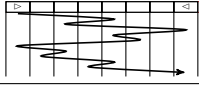
transducer	one-way	two-way
		
general	RAT	?

Table 1:: Expressiveness of one-way transducers

### 3. Hadamard Transductions

It can be shown that the class of transductions realized by two-way transducers is not closed under componentwise concatenation. For instance, when  $\Sigma = \Delta$  contains at least two symbols, no transducer realizes  $\text{SQUARE} \cdot \text{ERASE}$ , the transduction which associates  $u_1u_1$  to  $u_1u_2$  for any  $u_1, u_2 \in \Sigma^*$ . Intuitively, the factorization  $u_1u_2$  of the input word has to be guessed by the device, but cannot be stored in its finite control. Hence, when scanning for the second time the prefix  $u_1$  (which should happen since  $\text{SQUARE}$  is not a rational transduction), the automaton cannot recover the position of the last symbol of  $u_1$ .

However, other operations are well-suited for two-way transducers. Given two transductions  $R$  and  $S$ , we define the *Hadamard product* of  $R$  by  $S$ , denoted  $R \odot S$ , to be the transduction:

$$R \odot S = \{(u, v_1v_2) \mid (u, v_1) \in R \text{ and } (u, v_2) \in S\}.$$

So defined, we have  $\text{SQUARE} = \text{IDENTITY} \odot \text{IDENTITY}$ . The class of transductions defined by two-way (rotating or sweeping) transducers is closed under Hadamard product [27, 26].

**Proposition 3.1** *Let  $\mathcal{T}$  and  $\mathcal{T}'$  be two two-way transducers respectively realizing the transductions  $R$  and  $S$ . Then, there exists a two-way transducer  $\mathcal{T} \odot \mathcal{T}'$  which realizes  $R \odot S$ . Furthermore, the construction preserves determinism, unambiguity, and the properties of being rotating and sweeping.*

*Proof.* The construction is illustrated in Figure 1a. □

Another operation is the *Hadamard star* (or *Hadamard iteration*) of a transduction  $R$ , denoted  $R^{\otimes}$  and defined as follows:

$$R^{\otimes} = \{(u, v_1v_2 \cdots v_n) \mid n \in \mathbb{N} \text{ and } (u, v_i) \in R \text{ for each } i\}.$$

Hence,  $\text{POWER} = \text{IDENTITY}^{\otimes}$ . Observe that the operation necessarily introduces nondeterminism, since the number of iterations is not fixed. As for the Hadamard product, two-way transducers enjoy closure properties with respect to Hadamard star [27, 26].

**Proposition 3.2** *Let  $\mathcal{T}$  be a two-way transducer realizing  $R$ . Then, there exists a two-way transducer  $\mathcal{T}^{\otimes}$  which realizes  $R^{\otimes}$ . Furthermore, the construction preserves the properties of being rotating and sweeping.*

*Proof.* The construction is illustrated in Figure 1b. □



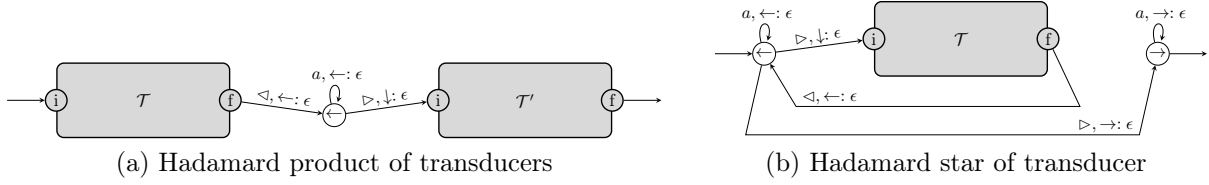


Figure 1:: Closure of two-way transducers under Hadamard product and Hadamard star.

Here,  $\leftarrow, \downarrow, \rightarrow$  respectively stand for left, stationary and right head moves, and  $a$  denotes any symbol from  $\Sigma$ .

**Definition 3.3** *The class of Hadamard transductions, denoted HAD, is the least class of transductions containing rational transductions which is closed under Hadamard operations, namely union, Hadamard product and Hadamard star.*

Because rational transductions correspond to left-to-right processing of the input by transducer, and since Hadamard product and Hadamard star capture the idea of rescanning the input word, it is not surprising to obtain the following characterization of transductions realized by rotating transducers [27, 14, 26].

**Theorem 3.4** *A transduction is realized by rotating transducer if and only if it is in HAD.*

The difference between rotating and sweeping transducers is the ability of the latter to scan the whole input word from right to left, *e.g.*, to compute MIRROR. This ability is captured by the *mirror* operation, defined as follows for a transduction  $R$ :

$$\overline{R} = \{(\overline{u}, v) \mid (u, v) \in R\}.$$

So defined,  $\text{MIRROR} = \overline{\text{IDENTITY}}$ .<sup>5</sup> We extend Definition 3.3 to a superclass of transductions.

**Definition 3.5** *The class of Mirror-Hadamard transductions, denoted MHAD, is the least class containing rational transductions which is closed under mirror and Hadamard operations.*

As expected, we obtain the following result [27, 25].

**Theorem 3.6** *A transduction is realized by sweeping transducer if and only if it is in MHAD.*

We have seen characterizations for three families of transducers, namely one-way, rotating, and sweeping transducers (see Table 2). However, two-way transducers are more expressive than sweeping transducers. A witness of this separation is the transduction PREFIX2PALINDROMIC over  $\Sigma = \Delta$  of cardinality at least 2, defined by:

$$\text{PREFIX2PALINDROMIC} = \{(u_1 u_2, u_1 \overline{u_1}) \mid u_1, u_2 \in \Sigma^*\} = (\text{IDENTITY} \odot \text{MIRROR}) \cdot \text{ERASE}.$$

<sup>5</sup>The transduction MIRROR is close to be realized by a one-way transducer, since such a device requires only one pass over the input word, but from right to left. Indeed, it is the mirror of a rational transduction, namely IDENTIFY. Such transductions have been studied in [11].

transducer	one-way	rotating	sweeping	two-way
general	RAT	HAD		MHAD
input unary				?
output unary				?
both unary				?

Table 2:: Expressiveness of different types of transducers according to the alphabet sizes

## 4. Unary Cases

We consider here the cases in which either the input, or the output, or both alphabets are *unary*, namely contain only one symbol. The situation is depicted in Table 2.

First of all, it is routine to prove that the mirror operation is irrelevant as far as one of the alphabets is unary, *e.g.*, [25].

**Proposition 4.1** *If  $\Sigma$  or  $\Delta$  is unary, then  $\text{MHAD} = \text{HAD}$ .*

Hence, by Theorems 3.4 and 3.6, rotating are as expressive as sweeping transducers in these special cases. They are however more expressive than one-way transducers, even if both alphabets are unary. A witness is the restriction of  $\text{POWER}$  to unary alphabets, denoted  $\text{UPOWER}$ :

$$\text{UPOWER} = \{(a^n, a^{kn}) \mid k, n \in \mathbb{N}\}.$$

The transduction is not rational. Indeed, identifying  $a^*$  with the additive monoid of integers  $\mathbb{N}$ , it defines the relation “being multiple of”. However, rational subsets of  $\mathbb{N}$  are first-order definable in Presburger arithmetics, *i.e.*, arithmetics with addition only.

### 4.1. Commutative Outputs

As highlighted by the  $\text{UPOWER}$  example, the further expressiveness of rotating transducers over unary output alphabet with respect to one-way transducers comes from the presence of loops in successful computations, that can be repeated an arbitrary number of times thus producing some power of a factor of the output. This is confirmed by a result of Anselmo on two-way weighted automata over commutative semirings, from which we can deduce that whenever the output alphabet is unary, *loop-free transducers*, namely two-way transducers that do not allow loops in successful computations, recognize rational transductions only [4]. This result applies to many interesting subcases, *e.g.*, functional transductions,<sup>6</sup> transducers

<sup>6</sup>Functional transducers are not always loop-free, but their accessible and co-accessible loops necessarily produce empty outputs. They can therefore be cut off when considering successful computations.

with deterministic underlying automaton. Another consequence of Anselmo's construction is the uniformization of output-unary two-way transducers by one-way transducer.<sup>7</sup> Indeed, by applying the construction to transducers with loops, we obtain a one-way transducer which realizes a sub-transduction of same domain (only outputs generated through loop-free successful computations are recovered). This one-way transducer can in turn be uniformized [18, 2, Prop. IX 8.2].

Hadamard transductions admit also a simpler form [10, 26], which mainly comes from the facts that, with commutative outputs, the Hadamard product is commutative, and rational transductions are closed under Hadamard product, *e.g.*, [30].

**Proposition 4.2** *Let  $T$  be a transduction over  $\Sigma, \Delta$  with  $\Delta$  unary. Then  $T$  is Hadamard if and only if there exists a finite set  $I$ , and two finite families  $(R_i)_{i \in I}$  and  $(S_i)_{i \in I}$  of rational transductions such that  $T = \bigcup_{i \in I} R_i \odot S_i^{\otimes}$ .*

## 4.2. Connecting Hits

Let  $\mathcal{T}$  be a two-way transducer and  $T$  be the transduction it realizes. A *hit* of  $\mathcal{T}$  over some input word  $u$  is a partial computation of  $\mathcal{T}$  over  $u$  between two successive visits of endmarkers. Hits can be grouped in four types according to their starting and ending sides. Moreover, they can be parametrized by their initial and final states. We thus speak of  $(q, s)$ -to- $(q', s')$  hits, for  $q$  a state and  $s$  an endmarker in  $\{\triangleright, \triangleleft\}$ , with the obvious meaning. Such a pair  $(q, s)$  is called a *border point*. Each parametrized family of hits define the transduction  $T_{q,s,q',s'}$ , defined by:

$$T_{q,s,q',s'} = \{(u, v) \mid v \text{ is produced during a } (q, s)\text{-to-}(q', s') \text{ hit over } u\}.$$

Notice that a simple modification of  $\mathcal{T}$  realizes  $T_{q,s,q',s'}$ . Given some descriptions of each  $T_{q,s,q',s'}$ , we can describe the transduction  $T$  as a finite expression connecting the  $T_{q,s,q',s'}$ 's with Hadamard operations, since successful computations are compositions of hits. (This was actually used in the proof of Theorems 3.4 and 3.6.) We thus obtain the following.

**Lemma 4.3** *If each  $T_{q,s,q',s'}$  is mirror-Hadamard or Hadamard then so is the transduction  $T$ .*

This can already apply to some particular output-unary transducers.

**Corollary 4.4** *Over a unary output alphabet, if  $\mathcal{T}$  is outer-nondeterministic, namely if non-deterministic choices may occur only while reading an endmarker [23], then the transduction realized is Hadamard. In particular, there exists an equivalent rotating transducer.*

*Proof.* Since the transducer is outer-nondeterministic, every hit occurring in a successful computation is loop-free. Hence, because the output alphabet is unary, each  $T_{q,s,q',s'}$  is rational by Anselmo's theorem [4]. Finally, since rational transductions are in particular Hadamard, we obtain the results by Lemmata 4.3 and 3.4.  $\square$

<sup>7</sup>A *uniformization* of a relation  $R$ , is a partial function  $F$  included in  $R$  which has same domain as  $R$ .

### 4.3. Unary Transducers

We consider now the case in which both the input and the output alphabets are unary. In this special case, we are able to characterize the class of transductions realized by two-way transducers: they are exactly the Hadamard transductions [10].

This result is obtained by combining three ingredients:

1. a one-way simulation of loop-free hits with commutative outputs, obtained by adapting the classical notion of *crossing sequences* as in [4];
2. the study of unary outputs that can be produced during a loop in a hit;
3. the connection of hits described in Lemma 4.3.

Only Ingredient 2, which is the key point of the proof, uses the assumption of having a unary input alphabet. Intuitively, since the input is unary, the production associated with a *central loop* (i.e., a loop that does not visit the endmarkers) does not really depend on the starting position of the loop, but can be shifted along the input tape, providing the starting position is sufficiently far away from the endmarkers. Given two elements  $\ell$  and  $r$  in  $\mathbb{N} \cup \{\infty\}$ , and a central loop  $\mathbf{r} = (q_0, p_0) \cdots (q_n, p_n)$  represented as a sequence of configurations over some fixed input word ( $q_0 = q_n$  and  $p_0 = p_n$ ), we say that  $\mathbf{r}$  is  $(\ell, r)$ -limited if for all  $0 \leq i \leq n$ , the head position component  $p_i$  is greater than or equal to  $p_0 - \ell$ , and less than or equal to  $p_0 + r$ , i.e.,  $p_0 - \ell \leq p_i \leq p_0 + r$ . Observe that if  $\ell' \geq \ell$  and  $r' \geq r$ , every  $(\ell, r)$ -limited central loop is  $(\ell', r')$ -limited. Also, any central loop is  $(\infty, \infty)$ -limited. We denote by  $\Lambda_{\ell, r}(q)$  the set of outputs generated by  $(\ell, r)$ -limited central loops around state  $q$ , namely:

$$\Lambda_{\ell, r}(q) = \{\Phi(\mathbf{r}) \mid \mathbf{r} \text{ is a } (\ell, r)\text{-limited central loop around } q\}.$$

As observed previously, we have  $\Lambda_{\ell, r}(q) \subseteq \Lambda_{\ell', r'}(q) \subseteq \Lambda_{\infty, \infty}(q)$  for any  $\ell' \geq \ell$  and  $r' \geq r$ . Furthermore, each  $\Lambda_{\ell, r}(q)$  contains  $\epsilon$ , as being the output of the trivial central loops around  $q$ . The set  $\Lambda_{\infty, \infty}(q)$  is the set of all outputs of central loops around  $q$ . The following shows that each of these languages is rational and that there are finitely many different such languages [10].

**Lemma 4.5** *For each state  $q$  and each  $\ell, r \in \mathbb{N} \cup \{\infty\}$ , the language  $\Lambda_{\ell, r}(q)$  is rational. Moreover, there exists a constant  $N \in \mathbb{N}$  depending on  $\mathcal{T}$  only, such that for each state  $q$  and each  $\ell, r \geq N$ , it holds:  $\Lambda_{\ell, r}(q) = \Lambda_{N, N}(q) = \Lambda_{\infty, \infty}(q)$ .*

*Proof.* We fix  $q$  and  $\ell, r \in \mathbb{N} \cup \{\infty\}$ . Since concatenating two  $(\ell, r)$ -limited loops yields a  $(\ell, r)$ -limited loop, we obtain the closure of  $\Lambda_{\ell, r}(q)$  under Kleene star. Because the language is furthermore unary, we obtain that it is rational, by identifying it with a sub-monoid of the additive monoid of integers which is therefore finitely generated.

Let  $\langle g_1, \dots, g_m \rangle$  be a finite family of generators of  $\Lambda_{\infty, \infty}(q)$ . Then, each  $g_i$  is the output of some central loop  $\mathbf{r}_i$  around  $q$ . By considering the maximal limits of these loops, we obtain that each generator is  $(N_q, N_q)$ -limited for some integer  $N_q$ . In particular,  $\Lambda_{N_q, N_q}(q) = \Lambda_{\infty, \infty}(q)$ . We conclude the proof by setting  $N$  as the maximum of the  $N_q$ 's over  $q$ .  $\square$

**Remark 4.6** *The above proof is non-constructive. In order to obtain an effective transformation, we should use an alternative proof.*

It is indeed possible, given  $q, \ell, r$ , to build a counter automaton which recognizes  $\Lambda_{\ell, r}(q)$ . Informally, the device performs a direct simulation of  $\mathcal{T}$  from state  $q$  but reads the emitted output rather than the input word, while assuming that only  $a$ 's are read by  $\mathcal{T}$ . In parallel, it uses the counter to store the distance of the simulated head position to the original position. This allows it to accept only when entering the state  $q$  with empty counter, that is, when the simulated computation is a central loop around  $q$ . Additionally, it can check that the counter value does not exceed  $\ell$  to the left and  $r$  to the right, using its finite control. This counter automaton can in turn be transformed into an equivalent finite automaton  $\mathcal{A}_{\ell, r}^{(q)}$  via Parikh's Theorem [28].

Finally, by successively testing equivalence of the automata  $\mathcal{A}_{\ell, r}^{(q)}$  while increasing  $\ell$  and  $r$ , it is possible to compute the constant  $N$  of the lemma. Unfortunately, this gives no bound on the size of this constant with respect to the size of  $\mathcal{T}$ . We leave this question as an open problem.

Combining Ingredients 1 and 2, namely [4] and Lemma 4.5, we obtain that the parametrized hits of a unary transducer define a rational transduction [10].

**Lemma 4.7** *Let  $\mathcal{T}$  be a unary two-way transducer. For each border points  $(q, s)$  and  $(q', s')$ , the transduction  $T_{q, s, q', s'}$  of productions of  $(q, s)$ -to- $(q', s')$  hits of  $\mathcal{T}$  is rational.*

We are now ready to state the characterization of unary two-way transducers [10].

**Theorem 4.8** *A unary transduction is realized by a two-way transducer if and only if it is in HAD.*

*Proof.* The result follows from Theorem 3.4, and Lemmata 4.3 and 4.7. □

As a consequence, every unary two-way transducer admits an equivalent rotating transducer.

**Corollary 4.9** *Unary two-way transducers are as expressive as unary rotating transducers.*

It is a natural question whether a similar results still holds when one of the alphabets only is unary. As the commutativity of the outputs is required by the main ingredients of the previous proof, it is not surprising to find counterexamples with unary input alphabets and nonunary output alphabets. Such an example is the transduction RLPREFIX defined by:

$$\text{RLPREFIX} = \{(a^n, a^m b^m) \mid n, m \in \mathbb{N}, m \leq n\}.$$

Informally, a two-way transducer realizing it can operate in three phases:

- Phase 1: rightward scanning of the input while emitting the symbol  $a$  at each step, until reaching some nondeterministically chosen position before the right endmarker;
- Phase 2: backward scanning of the input prefix while emitting the symbol  $b$  at each step, until reaching the left endmarker;
- Phase 3: acceptance by reaching the right endmarker without emitting any output symbol.

However, no rotating transducer realizes RLPREFIX [25].

Concerning transducers over nonunary input alphabets and unary output alphabets, we can also show that the statement of Theorem 4.8 does not hold. Notice that a counter example necessarily has to admit central loops, otherwise Ingredient 2 would be irrelevant. We define the transduction MULTONEBLOCK as follows:

$$\text{MULTONEBLOCK} = \{(u, a^{kn}) \mid u \in \{\#, a\}^* \text{ and } k, n \in \mathbb{N} \text{ such that } \#a^n\# \text{ is a factor of } u\}.$$

The transduction is an adaptation of UPOWER, in which a factor delimited by  $\#$ 's rather than the whole input word is raised to some power. It is an easy exercise to design a two-way transducer realizing it. However, it is more involved to show that there is no equivalent rotating transducer. Intuitively, this holds because the following two conditions are gathered. First,  $k$  and  $n$  are unbounded whence some computations require many passes over the input tape. Second, between two passes, a rotating transducer cannot store which factor delimited by  $\#$ 's has been selected on the previous passes, because there are an unbounded number of such factors. The formal proof of this separation result has required the investigation of the periods of the output language associated to an input word [25].

The two above examples implies that the unknown expressiveness of two-way transducers strictly extend the expressiveness of sweeping transducers as far as one of the alphabets has cardinality at least 2. In other words, the question marks in Table 2 indicate classes of transductions that strictly include MHAD.

**Perspective.** One possible direction towards characterizing the class of transductions realized by two-way transducers, is to adapt the *regular combinators* from [3, 17, 6], that already capture regular transductions. In this perspective, considering the case of commutative outputs seems more promising than the general case.

## 5. Origin Semantics

Transductions are not only sets of pairs of words. Indeed, they represent the result of some computation over an input word producing an output word. Hence, the output is implicitly related to the input. The *origin semantics* of transducers, introduced by Bojańczyk in [8], aims to make this correspondence more explicit. It is founded on the observation that each transducer actually provides more than a set of pairs of words. Indeed, from the transducer, one can also reconstruct the *origin information*, which says how positions of the output word originate from positions of the input word. Formally, the *origin* of an output position is the position of the input head when the corresponding output symbol was emitted. There are thus two semantics for transducers: one *standard semantics* where the output is a word, and an *origin semantics* where the output is a word with origin information. Identifying words with labeled paths, we can represent the latter as families of particular graphs, called *origin graphs*, which consists in:

1. a path with vertices labeled by  $\Sigma$  for the input word;

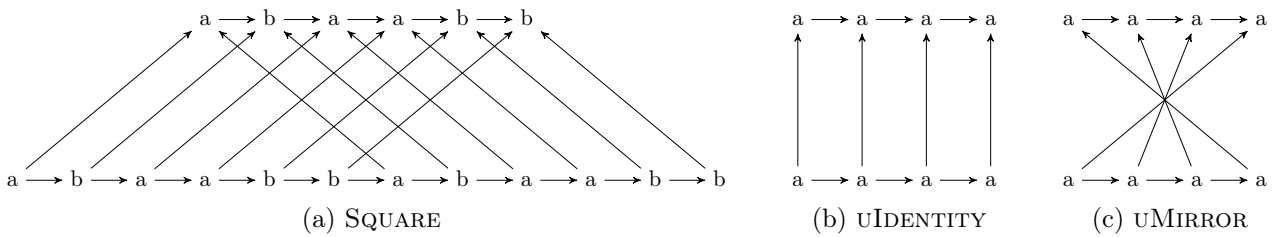


Figure 2:: Origin graphs from the origin semantics of transducers. The upper (resp. downer) path represent the input (resp. output), while the upward edges are the origin edges.

2. a second disjoint path with vertices labeled by  $\Delta$  for the output word;
3. *origin edges* from each output position towards some input position.

Figure 2 gives examples of origin graphs. A set of origin graphs is called an *origin transduction*. As shown in [8], the different equivalent formalisms for the class of regular transductions, namely functional two-way transducers, streaming string transducers [1] and MSO transductions [20], are consistent with origin information. Indeed, transforming one of these models into another one using the known constructions that proved their equivalences, yields a device which realizes the same transduction with the same origin information as the original one. In other words, the models do not only compute the same transduction but they compute it in a similar way.

The origin semantics of transducers is finer than the standard semantics in the sense that two transducers might be equivalent under the standard semantics, but not under the origin semantics. For instance, the restrictions to unary alphabets of the transducers realizing IDENTITY and MIRROR, respectively denoted UIDENTITY and UMIRROR, are equivalent for the standard semantics (indeed, the transduction realized is  $\{(a^n, a^n) \mid n \in \mathbb{N}\}$  in both cases), but their origin semantics differ (*cf.* Figures 2b and 2c).

In a very recent unpublished work on two-way transducers with origin semantics, Bose, Muscholl, Penelle, and Puppis have shown that the equivalence of nondeterministic transducers with origin is decidable [Personal communication], thus contrasting with the undecidability of their standard equivalence [24].

The origin semantics is richer than the standard semantics. It allows in particular to use structural graph properties, such as having bounded degree, to talk about family of origin graphs, namely origin transductions. It has been shown in [9] that an origin transduction is the origin semantics of a functional two-way transducers if and only if it has four properties:

- (1) it is MSO-definable as a set of coloured graph (*i.e.*, there exists an MSO formula which is true in exactly the origin graphs of the transduction);
- (2) it has bounded degree (equivalently, an input position might be the origin of a bounded number of output positions);
- (3) it is *origin functional*, *i.e.*, each input word, appears in at most one origin graph of the origin transduction;<sup>8</sup>

---

<sup>8</sup>Notice that this property is stronger than functionality for the standard semantics, *e.g.*, UIDENTITY  $\cup$

- (4) it has bounded *crossing*, which intuitively means that the origin mapping does not oscillate to much. More precisely, the *crossing* of an origin graph counts how many visits to an input position is required by a two-way transducer to realize the transduction. In particular, bounding it implies that each input position is visited only a bounded number of times whence there is no loop.

It thus seems natural to relax these structural properties in order to capture some classes of nonfunctional transductions. Still in [9], it has been proved that (1), (2) and (4) characterize the origin semantics of nondeterministic streaming string transducers. This model is orthogonal to two-way transducers. Indeed, a nondeterministic streaming string transducer may for instance realize the transduction SQUARED SUBWORD defined by:

$$\text{SQUARED SUBWORD} = \{(u, vv) \mid v \text{ is a subword of } u\} = \text{SQUARE} \circ \text{SUBWORD},^9$$

which cannot be realized by any two-way transducer, but, conversely, the model can realize only origin transductions with bounded degree, contrary to two-way transducers, *e.g.*, POWER.

So, although functional streaming string transducers and functional two-way transducers have same expressiveness [1], their nondeterministic versions define orthogonal classes of transductions. This is because the nondeterminism of the two models has a completely different nature. On the one hand, the nondeterminism of streaming string transducers allows to globally select some input position, *e.g.*, in SQUARED SUBWORD we can pre-select the input positions of the symbols that will form the subword. This is highlighted by the equivalence of the model with *nondeterministic MSO transducers* [2], in which the nondeterminism is precisely a global pre-selection of positions (also called *common guess*). On the other hand, the nondeterminism of two-way transducers allows computations to loop, thus emitting an unbounded repetition of some output factor.

In [9], the authors considered some combinations of these abilities, namely streaming string transducer with  $\varepsilon$ -moves, and two-way transducers with common guess. It was shown that when considering only origin semantics of the latter model which have bounded degree, one recovers exactly the origin semantics of nondeterministic streaming string transducers. In particular, the class of bounded-degree origin transductions defined by two-way transducers is strictly included in the class of origin transductions realized by nondeterministic streaming string transducers.

We believe that the origin semantics of two-way nondeterministic transducers is a promising direction towards describing the expressiveness of some relevant subclasses of the model, while recovering decidability results such as equivalence of devices. The origin semantics indeed provides a more suitable structure for relating transductions with logic. In this scope, it could be interesting to study the class of nonfunctional transductions that are definable in the decidable logic from [15], a logic speaking of transductions which characterizes regular transductions when restricted to functions. Conversely, it could be interesting to design new nondeterministic transducing models which realize some robust classes of origin transductions, for instance satisfying (1), (2), and having bounded pathwidth.

---

UMIRROR is not origin functional.

<sup>9</sup>Here,  $S \circ R$  denotes the composition of  $R$  and  $S$ , namely  $(x, y) \in S \circ R \iff \exists z \mid (x, z) \in R \wedge (z, y) \in S$ .



## Acknowledgements

The author wishes to express his gratitude to Christian Choffrut and Giovanni Pighizzini for several helpful comments and suggestions during the preparation of the paper.

## References

- [1] R. ALUR, P. ČERNÝ, Expressiveness of streaming string transducers. In: K. LODAYA, M. MAHAJAN (eds.), *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. LIPIcs: Leibniz International Proceedings in Informatics 8, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany, 2010, 1–12.
- [2] R. ALUR, J. V. DESHMUKH, Nondeterministic Streaming String Transducers. In: L. ACETO, M. HENZINGER, J. SGALL (eds.), *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011, Zürich, Switzerland, July 4-8, 2011. Part II*. Lecture Notes in Computer Science 6756, Springer, 2011, 1–20.
- [3] R. ALUR, A. FREILICH, M. RAGHOTHAMAN, Regular combinators for string transformations. In: *Computer Science Logic (CSL)/Logic in Computer Science (LICS)*. ACM, 2014, 9:1–10.
- [4] M. ANSELMO, Two-way automata with multiplicity. In: M. PATERSON (ed.), *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*. Lecture Notes in Computer Science 443, Springer, 1990, 88–102.
- [5] F. BASCHENIS, O. GAUWIN, A. MUSCHOLL, G. PUPPIS, Untwisting two-way transducers in elementary time. In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*. IEEE, 2017, 1–12.
- [6] N. BAUDRU, P.-A. REYNIER, From two-way transducers to regular function expressions. 2018. To appear in Proceedings of DLT 2018, Lecture Notes in Computer Science.
- [7] J. BERSTEL, *Transductions and context-free languages*. Vieweg+Teubner Verlag, 1979.
- [8] M. BOJAŃCZYK, Transducers with Origin Information. In: J. ESPARZA, P. FRAIGNIAUD, T. HUSFELDT, E. KOUTSOUPIAS (eds.), *Automata, Languages, and Programming (ICALP 2014). Part II*. Lecture Notes in Computer Science 8573, Springer, 2014, 26–37.
- [9] M. BOJAŃCZYK, L. DAVIAUD, B. GUILLON, V. PENELLE, Which classes of origin graphs are generated by transducers. In: I. CHATZIGIANNAKIS, P. INDYK, F. KUHN, A. MUSCHOLL (eds.), *Automata, Languages, and Programming (ICALP)*. LIPIcs: Leibniz International Proceedings in Informatics 80, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany, 2017, 114:1–13.
- [10] C. CHOFFRUT, B. GUILLON, An algebraic characterization of unary two-way transducers. In: E. CSUHAI-VARJÚ, M. DIETZFELBINGER, Z. ÉSIK (eds.), *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Part I*. Lecture Notes in Computer Science 8634, Springer, 2014, 196–207.

- [11] C. CHOFFRUT, B. GUILLON, Both ways rational functions. In: S. BRLEK, C. REUTENAUER (eds.), *Developments in Language Theory – 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings*. Lecture Notes in Computer Science 9840, Springer, 2016, 114–124.
- [12] M. CHYTIL, V. JÁKL, Serial composition of 2-way finite-state transducers and simple programs on strings. In: A. SALOMAA, M. STEINBY (eds.), *Automata, Languages and Programming, Fourth Colloquium, University of Turku, Finland, July 18-22, 1977, Proceedings*. Lecture Notes in Computer Science 52, Springer, 1977, 135–147.
- [13] B. COURCELLE, J. ENGELFRIET, *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach*. Encyclopedia of Mathematics and its Applications 138, Cambridge Univ. Press, 2012.
- [14] L. DANDO, S. LOMBARDY, From Hadamard expressions to weighted rotating automata and back. In: A. CARAYOL, C. NICAUD (eds.), *Implementation and Application of Automata – 22nd International Conference, CIAA 2017, Marne-la-Vallée, France, June 27-30, 2017, Proceedings*. Lecture Notes in Computer Science 10329, Springer, 2017, 163–174.
- [15] L. DARTOIS, E. FILIOT, N. LHOTE, Decidable logics for transductions and data words. *CoRR* abs/1701.03670 (2017).
- [16] L. DARTOIS, P. FOURNIER, I. JECKER, N. LHOTE, On reversible transducers. In: I. CHATZIGIANNAKIS, P. INDYK, F. KUHN, A. MUSCHOLL (eds.), *Automata, Languages, and Programming (ICALP)*. LIPIcs: Leibniz International Proceedings in Informatics 80, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 113:1–12.
- [17] V. DAVE, P. GASTIN, K. S. NARAYANAN, Regular transducer expressions for regular transformations. *CoRR* abs/1802.02094 (2018).
- [18] S. EILENBERG, *Automata, Languages and Machines*, A. Academic Press, 1974.
- [19] C. C. ELGOT, J. E. MEZEI, On relations defined by generalized finite automata. *IBM Journal of Research and Development* 9 (1965) 1, 47–68.
- [20] J. ENGELFRIET, H. J. HOOGEBOOM, MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)* 2 (2001) 2, 216–254.
- [21] E. FILIOT, O. GAUWIN, N. LHOTE, Logical and algebraic characterizations of rational transductions. *CoRR* abs/1705.03726 (2017).
- [22] E. FILIOT, O. GAUWIN, P.-A. REYNIER, F. SERVAIS, From two-way to one-way finite state transducers. In: *Logic in Computer Science (LICS)*. IEEE Computer Society, 2013, 468–477.
- [23] V. GEFFERT, B. GUILLON, G. PIGHIZZINI, Two-way automata making choices only at the endmarkers. *Information and Computation* 239 (2014), 71–86.
- [24] T. V. GRIFFITHS, The unsolvability of the equivalence problem for  $\wedge$ -free nondeterministic generalized machines. *Journal of the ACM (JACM)* 15 (1968) 3, 409–413.
- [25] B. GUILLON, Input- or output-unary sweeping transducers are weaker than their 2-way counterparts. *RAIRO – Theoretical Informatics and Applications (RAIRO: ITA)* 50 (2016) 4, 275–294.

- [26] B. GUILLON, *Two-wayness: automata and transducers*. PhD thesis, Université Paris Diderot, Paris 7 and Università degli Studi di Milano, 2016.
- [27] S. LOMBARDY, Two-way representations and weighted automata. *RAIRO - Theoretical Informatics and Applications* 50 (2016) 4, 331–350.
- [28] R. PARIKH, On context-free languages. *Journal of the ACM (JACM)* 13 (1966) 4, 570–581.
- [29] M. O. RABIN, D. S. SCOTT, Finite automata and their decision problems. *IBM Journal of Research and Development* 3 (1959) 2, 114–125.
- [30] J. SAKAROVITCH, *Elements of Automata Theory*. Cambridge University Press, 2009.
- [31] D. SOUZA, RODRIGO, Uniformisation of two-way transducers. In: D. HUTCHISON, T. KANADE, J. KITTLER, J. M. KLEINBERG, F. MATTERN, J. C. MITCHELL, M. NAOR, O. NIERSTRASZ, C. PANDU RANGAN, B. STEFFEN, M. SUDAN, D. TERZOPOULOS, D. TYGAR, M. Y. VARDI, G. WEIKUM, A.-H. DEDIU, C. MARTÍN-VIDE, B. TRUTHE (eds.), *Language and Automata Theory and Applications (LATA)*. Lecture Notes in Computer Science 7810, Springer, 2013, 547–558.



# ON THE APPLICATION OF WATSON-CRICK FINITE AUTOMATA FOR THE RESOLUTION OF BIOINFORMATIC PROBLEMS

(Extended Abstract)

José M. Sempere

DSIC, Universitat Politècnica de València,  
Camino de Vera s/n, 46022 Valencia, Spain  
jsempere@dsic.upv.es

Watson-Crick Finite Automata (WKFA) were formulated in the framework of DNA computing [1]. They are based on DNA recombination and the complementarity relationship between the double-stranded nucleotides to form biochemical bonds. From the beginning, it was established that WKFA recognize some language classes that are included in the context sensitive class. Therefore, the WKFA can recognize languages that could be applied to the modeling of languages that are useful for solving different tasks. In our case, we take advantage of the descriptive power of the languages accepted by WKFA. We apply them in the tasks of annotation, classification and prediction for genomic sequences (biosequences of DNA, RNA and proteins).

To address the integral design of a classifier or an annotation tool based on WKFA, we use a reduction technique based on a representation theorem that we proposed in [2]. This allows us to define the languages accepted by WKFA as intersections of linear languages and even linear languages. The main advantages of this reduction is that we can introduce characteristic features such as those shown in [3, 4] and we can approach the machine learning of these models with grammatical inference techniques such as those shown in [5, 6].

We will show how to build *de novo* a WKFA classifier for genomic sequences from a (finite) set of annotated examples and counterexamples. We will propose a solution for the bioinformatic tasks of annotation and classification of structural motifs in proteins such as the *coiled coils* and *transmembrane* motifs. We will show some experiments with real data, and we will also make a proposal for the modeling of annotation for pseudoknots in RNA that is still an open problem in structural bioinformatics.

## References

- [1] R. FREUND, GH. PĂUN, G. ROZENBERG, A. SALOMAA, Watson-Crick finite automata. In: H. RUBIN, D. HARLAN (eds.), *DNA Based Computers III, Pennsylvania, USA, June 23–25, 1997, Proceedings*. DIMACS 48, American Mathematical Society, USA, 1997, 305–317.
- [2] J. M. SEMPERE, A representation theorem for languages accepted by Watson-Crick finite automata. *Bulletin of the EATCS* (2004) 83, 187–191.
- [3] J. M. SEMPERE, On local testability in Watson-Crick finite automata. In: GY. VASZIL (ed.), *International Workshop on Automata for Cellular and Molecular Computing, Budapest, Hungary, August 31. Proceedings*. MTA SZTAKI, Hungary, 2007, 38–44.
- [4] J. M. SEMPERE, Exploring regular reversibility in Watson-Crick finite automata. In: M. SUGISAKA, H. TANAKA (eds.), *13th International Symposium on Artificial Life and Robotics AROB 2008, Beppu, Japan, January 31 to February 2, Proceedings*. 2008, 505–509.
- [5] J. M. SEMPERE, P. GARCÍA, A characterization of even linear languages and its application to the learning problem. In: R. CARRASCO, J. ONCINA (eds.), *Grammatical Inference and Applications*. Lecture Notes in Artificial Intelligence (LNAI) 862, Springer, 1994, 120–128.
- [6] J. M. SEMPERE, G. NAGARAJA, Learning a subclass of linear languages from positive structural information. In: V. HONAVAR, G. SLUTZKI (eds.), *Grammatical Inference*. Lecture Notes in Artificial Intelligence (LNAI) 1433, Springer, 1998, 162–174.

# CAUSAL DYNAMICS OF DISCRETE MANIFOLDS

Pablo Arrighi<sup>(A)</sup>      Clément Chouteau<sup>(B)</sup>  
Stefano Facchini<sup>(A)</sup>      Simon Martiel<sup>(C)</sup>

<sup>(A)</sup>Aix-Marseille Univ., Université de Toulon, CNRS, LIS, Marseille, and IXXI, Lyon, France  
{pablo.arrighi, stefano.facchini}@univ-amu.fr

<sup>(B)</sup>Inria, LSV, ENS Paris-Saclay 61, avenue du Président Wilson, 94235 Cachan Cedex, France

<sup>(C)</sup>Atos/Bull, Quantum R&D, 78340 Les Clayes-sous-Bois, France  
simon.martiel@atos.net

## **Abstract**

*We extend Cellular Automata to time-varying discrete geometries. In other words we formalize, and prove theorems about, the intuitive idea of a discrete manifold which evolves in time, subject to two natural constraints: the evolution does not propagate information too fast; and it acts everywhere the same. For this purpose we develop a correspondence between complexes and labeled graphs. In particular we reformulate the properties that characterize discrete manifolds amongst complexes, solely in terms of graphs. In dimensions  $n < 4$ , over bounded-star graphs, it is decidable whether a Cellular Automaton maps discrete manifolds into discrete manifolds.*

**Keywords:** *Causal graph dynamics, crystallizations, gems, balanced complexes, combinatorial manifolds, graph-local Pachner moves, bistellar, inverse shellings, homeomorphism, Regge-calculus, causal dynamical triangulations, spin networks.*

## **1. Introduction**

Discrete geometry refers to discretizations of continuous geometries, i.e., piecewise-linear manifolds, that can be abstracted as combinatorial objects such as simplicial complexes, etc. But it may also refer to mere graphs/networks equipped with their natural graph distance. This ambiguity is common in Computer Science, but also in Physics. For instance in discrete/quantized versions of General Relativity, spacetime is discretized as simplicial complexes (Regge-calculus) or in the basis of spin networks graphs (Loop Quantum Gravity). This raises the question of a thorough comparison between simplicial complexes and graphs.

A natural way of approaching this question is to seek to encode complexes into labeled graphs. Then, a natural way to encode a complex into a labeled graph, is to map: each simplex  $u$  into a vertex  $u$ ; each facet  $u.a$  of the simplex into a port  $u.a$  of the vertex  $u$ ; each gluing between facets

$u:a$  and  $v:b$  into an edge  $(u:a, v:b)$ ; each possible way of rotating/articulating this gluing as a label  $\gamma$  carried by this edge – see Fig. 1 and 2. We formalize this correspondence in Section 2. Notice that by ‘complex’, we really mean ‘pseudo-manifold’ here, i.e., each facet of a simplex is attached to one other facet at most. Notice also that the precise choices of ports may not matter, so long as the edges between them represent oriented gluings of simplices. There is, therefore, a local rotation symmetry.

This works well, but a non-often emphasized problem arises. Consider triangles hinging around a point, as in the bottom left of Fig. 1. The geometrical distance between the two extreme tetrahedrons is one, since they share a point. But the graph distance between their corresponding vertices is three, and the path between them could be made – they are not graph neighbors. There is a discrepancy between the two distances, which we characterize in Section 3. Faced with this discrepancy, we have two options.

One option is to forget about geometrical distance altogether. Indeed, if one thinks of each tetrahedron as a room (as in the movie *Cube*, say), then it is the graph distance that matters. In Section 4, we develop a theory of Causal Dynamics of Complexes (CDC). CDC evolve complexes in discrete time steps, subject to two natural constraints: the evolution does not propagate information too fast; and it acts everywhere the same. This is thanks to the concept of Causal Graph Dynamics (CGD), which we recall. We prove that the CGD which commute with local rotations, can always be implemented with rotation-commuting local rules – a property which in turn is decidable. Then, the previously developed correspondence between complexes and labeled graphs readily allows us to reinterpret these rotation-commuting CGD, as CDC. CDC are already interesting as a mathematically rigorous framework in which to cast the more pragmatical simplicial complex parallel rewrite system of [8], or in order to explore causal dynamics of Causal Dynamical Triangulations [1], à la [11].

Another option is to take geometrical distance into account. Then, looking at the complex at this larger scale, and in dimension 3 and above, unravels new concerns. For instance Fig. 5 has the topology of a pinched ball (think of a balloon compressed between two fingers until they touch each other). This is not a manifold, since the neighborhood of the compression point is not a ball. Thus, since the cycle length is arbitrary, the property of being a manifold, is non-local. To make matters worse, the two extreme tetrahedrons could have been glued in a torsioned manner, see Fig 5 again. A somewhat radical solution to these concerns is to restrict to complexes such that, even in the geometrical distance, neighborhoods are bounded. Then, the discrepancy between the geometrical and the graph distances is linearly bounded. Another motivation for considering these ‘bounded-star complexes’ is if the next state of a tetrahedron is computed from that of the geometrically neighboring tetrahedrons: we may want this neighborhood to be bounded, whether for practical purposes (e.g. efficiency of a finite-volume elements methods) or theoretical reasons (e.g. computability from the finiteness of the local update rule; finite-density as a physics postulate). Finally, this is a way to prevent sudden geometrical distance collapse – see Fig. 10.

In Section 5 we characterize manifolds. In continuous geometries, a manifold is characterized by the neighborhood of every point being homeomorphic to a ball. Over simplicial complexes, this translates into the neighborhood of every simplex being homeomorphic to a ball, where



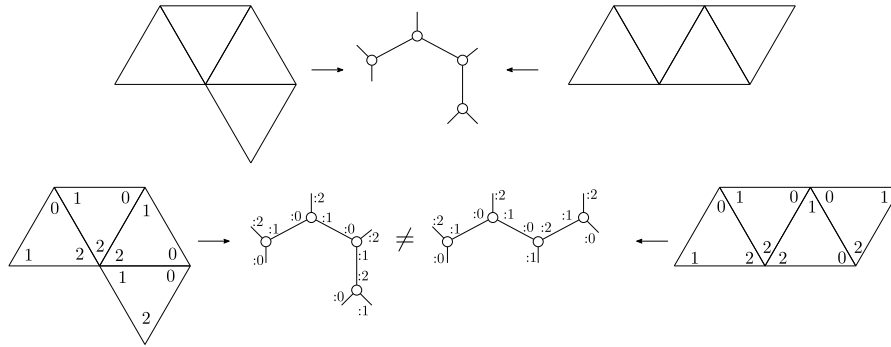


Figure 1: Complexes as graphs. *Top row.* The naive way to encode complexes as graphs is ambiguous. *Bottom row.* Encoding colored complexes instead lifts this ambiguity.

the notion of homeomorphism is captured by a finite set of local rewrite rules, often referred to as Pachner moves (technically, bistellar moves plus shellings and their inverses). These moves are reformulated in terms of graph moves; but the obtained graph moves are not graph-local. We show that a subset of these graph moves is just as expressive as the Pachner moves, whilst enjoying the property of being graph-local. That way, the properties that characterize discrete manifolds are reformulated in terms of graph-local moves.

In Section 6, we show that it is decidable whether a CGD is torsion-free bounded-star discrete-manifold preserving – in dimensions less than four. Then, the correspondence between complexes and labeled graphs readily allows us to reinterpret these, as Causal Dynamics of Discrete Manifolds (CDDM).

We conclude in Section 7 with a discussion of the result and their connection with past and future works, as well as a detailed comparison with the crystallizations/gems alternative.

## 2. Complexes as Graphs

The naive way is to map each simplex to a vertex  $v$ , and each gluing between facets to an edge  $\{u, v\}$ . The problem, then, is that we can no longer tell one facet from another, which leads to ambiguities (see Fig. 1 *Top row.*). A first solution attempt is to consider *colored simplicial complexes* instead. In these complexes, each of the  $n + 1$  points of a  $n$ -simplex has a different color. Now we can map each simplex to a vertex, and each gluing between facets to an edge, but now this edge  $\{u:p, v:q\}$  holds the colors of the points that are opposite the glued facets (see Fig. 1 *Bottom row.*). The problem, now, is that as soon as we consider 3-dimensional complexes, there are three different, rotated/articulated ways of gluing two tetrahedrons along two given facets (see Fig. 2). We must therefore provide a permutation  $\gamma$  telling us which points identifies with whom, on the edges. Because these permutations are not, in general, involutions, we must direct our edges. Odd permutations correspond to oriented gluings. Altogether this leads to the following definition, which is an elaborated version of [2, 3, 4]:

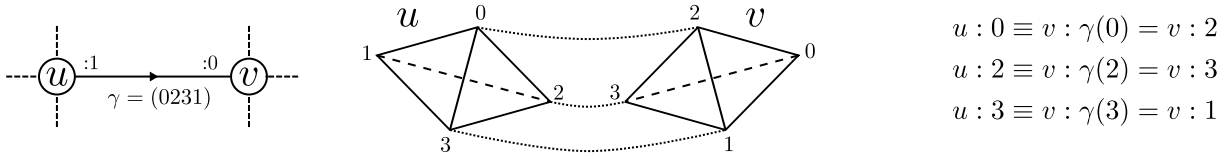


Figure 2: The different, rotated/articulated ways of gluing of two tetrahedrons along two given facets are specified on the edges.

**Definition 2.1 (Graphs, Disks)** Let  $V$  be an infinite countable set referred to as the ‘universe of names’. Let  $\Sigma$  be a finite set referred to as the ‘internal states’. Let  $n$  stand for the spatial dimension. Let  $\pi$  be  $0 \dots n + 1$  referred to as the ‘set of ports’. Let  $\Gamma$  be the  $(n + 2)!/2$  odd permutations of  $\pi$ , referred to as ‘gluings’. A graph  $G$  is given by

- A subset  $V(G)$  of  $V$  – whose elements are called vertices.
- A function  $\sigma : V(G) \rightarrow \Sigma$  associating to each vertex its label.
- A set  $S(G)$  of elements of the form  $(u:p)$  with  $u \in V(G)$ ,  $p \in \pi$  – whose elements are called semi-edges.
- A set  $E(G)$  of elements of the form  $(u:p, \gamma, v:q)$  with  $u, v \in V(G)$ ,  $p, q \in \pi$ ,  $\gamma \in \Gamma$  – whose elements are called edges.

This is with the conditions that

- if  $(u:p) \in S(G)$  then there is no  $(u:p, \gamma, v:q) \in E(G)$ .
- if  $(u:p, \gamma, v:q) \in E(G)$  then there is no other  $(u:p, \gamma', v':q') \in E(G)$ .
- each vertex has exactly  $n + 1$  ports, i.e. appearing in  $S(G) \cup E(G)$ .
- if  $(u:p, \gamma, v:q) \in E(G)$  then  $\gamma$  must map the  $n + 1$  ports of  $u$  into the  $n + 1$  ports of  $v$ , with  $\gamma(p) = q$ .
- if  $(u:p, \gamma, v:q) \in E(G)$  then  $(v:q, \gamma^{-1}, u:p) \in E(G)$ .

The set of graphs is denoted  $\mathcal{G}$ . Given a graph  $G$ , we write  $G_v^r$  for its disk of radius  $r$  centered on  $v$ , i.e. its subgraph induced by those vertices that lie at graph distance less or equal to  $r + 1$  from  $v$  in  $G$ , breaking outgoing edges into semi-edges. The set of disks of radius  $r$  is denoted  $\mathcal{D}^r$ .

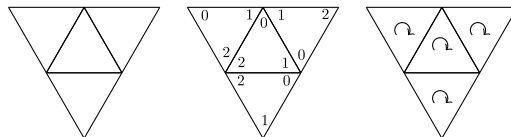


Figure 3: Complexes, Colored complexes, Oriented Complexes.

These graphs correspond to colored complexes, i.e. gluings of simplices whose points have colors. Colored simplicial complex are not uncommon, but certainly not as common as oriented complexes, however – see Fig. 3. If we wish to remove colors, we must allow for the simplices

to rotate freely. On graphs this corresponds to reshuffling the ports in  $\pi$  according to an even permutation, i.e. a rotation.

**Definition 2.2 (Vertex rotations and symmetries)** *Let  $G$  be a graph,  $u$  one of its vertex and  $r$  an even element of  $\Pi$ . Then, a vertex rotation  $r_u$  is the application of  $r$  at  $u$ . More precisely,  $G' = (r_u)G$ , is such that*

- $V(G') = V(G)$ .
- $E(G')$  and  $S(G')$  are the image of  $E(G)$  and  $S(G)$  under the map:
  - $(u:p, \gamma, v:q) \mapsto (u:r(p), \gamma \circ r^{-1}, v:q)$ .
  - $(v:q, \gamma^{-1}, u:p) \mapsto (v:q, r \circ \gamma^{-1}, u:r(p))$ .
  - $(u:p) \mapsto (u:r(p))$ .
- $\sigma'(u) = h(r)(\sigma(u))$ , whereas  $\sigma'(v) = \sigma(v)$  for  $v \neq u$ ,

where  $h$  is a given homomorphism between the group of permutations  $\Gamma$  over  $\pi$ , and a group of transformations  $h(\Gamma)$  over  $\Sigma$ . A rotation sequence  $\bar{r}$  is a finite composition  $\prod r_{u_i}^i$ , with  $r^i$  some rotations, and  $u_i$  some vertices. When  $s$  is an odd element of  $\Pi$ , we can similarly define a vertex symmetry  $s_u$  and a symmetry sequence  $\bar{s}$ . We use  $s_{ij}$  as a shorthand notation for the flip between  $i$  and  $j$ .

Oriented simplicial complexes correspond to the equivalence classes of our labeled graphs:

**Definition 2.3 (Rotation Equivalence)** *Two graphs  $G$  and  $H$  are rotation equivalent if and only if there exists a rotation sequence  $\bar{r}$  such that  $\bar{r}G = H$ .*

From now on and in the rest of this paper, we will let  $\Sigma = \emptyset$  in order to simplify notations – although all of the results of this paper carry through to graphs with internal states.

### 3. Graph Distance Versus Geometrical Distance

On the one hand in the world of simplicial complexes, two simplices are adjacent if they share a geometric point (a 0-face). On the other hand in the world of graphs, two vertices are adjacent if they share an edge. These two notions do not coincide, as shown in Fig. 1. In order to understand the interplay between geometrical and graph distances, we first express the notion of  $k$ -face of a given simplex, in graph terms. We then provide graph-based condition that tell whether the  $k$ -face is shared by another simplex.

The way we express the notion of  $k$ -face in terms of graphs is as follows. Consider Figure 2. Each port  $p$  can be interpreted, geometrically, as the point opposite to where the gluing occurs. Then, a  $k$ -face  $F$  can be described just by the set of ports–points that composes it.

**Definition 3.1 (Face)** *A  $k$ -face  $F$  at vertex  $u$  is a subset  $\{p_0, \dots, p_k\}$  of  $k + 1$  ports of a vertex  $u$ .*

Now, if a point  $p$  belongs to a  $k$ -face at  $u$ , and a simplex  $u'$  is glued on port  $p$ , this simplex no longer contains the  $k$ -face, as it excludes the point  $u : p$ . We can use this to characterize geometrically equivalent  $k$ -faces and hinges around them, i.e. paths along simplices that include them.

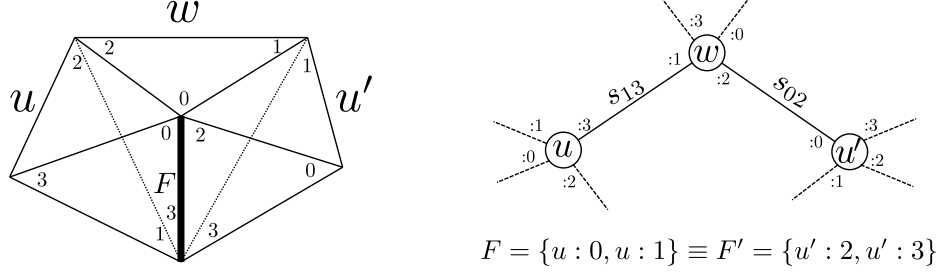


Figure 4: A hinge between  $F$  at  $u$  and  $F'$  at  $u'$ .

**Definition 3.2 (Hinges between equivalent faces)** *Two  $k$ -faces  $F$  at vertex  $u$  and  $F'$  at vertex  $u'$  are said to be equivalent if and only if they are related by a hinge, i.e. if and only if there exists is a path  $(u_i : p_i, \gamma_{i+1}, u_{i+1} : q_{i+1}) \in E(G)$  with  $i = 0 \dots m$ ,  $u_0 = u$ ,  $u_{m+1} = u'$ , such that :*

$$p_i, q_i \notin \left( \prod_{j=1}^i \gamma_j \right) (F) \quad \text{and} \quad F' = \left( \prod_{j=1}^m \gamma_j \right) (F) \quad (1)$$

where  $p_i = p_0, \dots, p_m$ , whereas  $q_i = q_1, \dots, q_{m+1}$ .

When a gluing occurs on port  $u : p$ , it ‘covers’ the points  $u : \pi \setminus \{p\}$ . Conversely, a  $k$ -face  $F$  at  $u$  is covered by all those gluings occurring at  $\pi \setminus F$ .

**Definition 3.3 (Border face)** *Given a  $k$ -face  $F$  at vertex  $u$ , consider every  $F'$  at  $u'$  that is equivalent to  $F$ . Its set of covering semi-edges is*

$$S(G) \cap \bigcup_{u'} (u' : \pi \setminus F').$$

If this set is non-empty,  $F$  is a border face.

Sometimes a hinge can be closed-up into a cyclic hinge in a way that identifies a  $k$ -face, with a rotated/articulated version of itself, as in Fig. 5.

**Definition 3.4 (Torsion)** *A torsion is a hinge around two distinct  $k$ -faces  $F$  and  $F'$  at  $u$ .*

Whilst such torsions may be useful in order to model certain kinds of parallel transport, we regard them as undesirable in this paper. Here is one tool to chase them out:

**Definition 3.5 (Normal form)** *A path  $\{u_i : p_i, \gamma_{i+1}, u_{i+1} : q_{i+1}\} \in E(G)$  with  $i = 0 \dots m$  is in normal form if and only if for all  $i$ ,  $\gamma_{i+1} = s_{p_i q_{i+1}}$  and if the  $n+1$  ports of  $u_i$  are  $\{p_i, r_1, \dots, r_n\}$ , then those of  $u_{i+1}$  are  $\{q_{i+1}, r_1, \dots, r_n\}$ .*

**Proposition 3.6** *A cyclic hinge that can be put in normal form, is torsion-free.*

*Proof.* Say that the hinge has been put in normal form. Pick  $F$  a  $k$  face at  $u_0$  such that the left equation (1) is verified. In normal form  $s_{p_i q_{i+1}}$  leaves  $r_1, \dots, r_n$  unchanged. Obviously  $p_0 \notin F$ , hence  $s_{p_0 q_1}(F) = F$ . And similarly for the next steps. Therefore the  $F'$  of the right equation (1) is  $F$ .  $\square$

Generally speaking, chasing out torsions is a difficult thing, because cyclic hinges may be arbitrary long. Unless we make further assumptions.

**Definition 3.7 (Star, Bounded-star)** *Consider a graph  $G$  and a vertex  $u$  in  $G$ . A vertex  $u'$  in  $G$  is said to be a geometrical neighbor of  $u$  if and only if they have an equivalent  $k$ -face. The star of  $u$  is the subgraph induced by  $u$  and its geometrical neighbors. It is denoted  $\text{Star}(G, u)$ . A graph  $G$  is said to be bounded-star of bound  $s$  if and only if its hinges are of length less than or equal to  $s$ .*

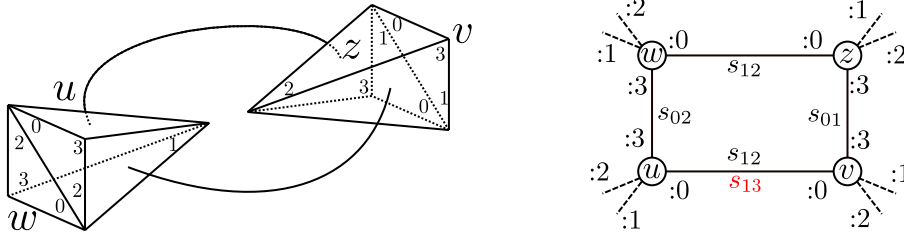


Figure 5: With the black gluing between  $u$  and  $v$ , the complex has the topology of a pinched ball, i.e. a doughnut whose hole has collapsed into a point. This constitutes an example of a pseudo-manifold (well-glued simplices) that is not a discrete manifold (the neighborhood of the pinch is not a ball). With the red gluing, the complex is torsioned. For instance, the 0-faces  $\{1\}$  and  $\{2\}$  at  $u$  are made equivalent in the sense of Def. 3.2.

## 4. Causal Dynamics of Complexes

We now recall the essential definitions of CGD, through their constructive presentation, namely as localizable dynamics. We will not detail, nor explain, nor motivate these definitions in order to avoid repetitions with [2, 3, 4]. Still, notice that in [2, 3, 4] this constructive presentation is shown equivalent to an axiomatic presentation of CGD, which establishes the full generality of this formalism. The bottom line is that these definitions capture all the graph evolutions which are such that information does not propagate information too fast and which act everywhere the same, see Fig. 6.

**Definition 4.1 (Isomorphism)** *An isomorphism is specified by a bijection  $R$  from  $V$  to  $V$  and acts on a graph  $G$  as follow:*

- $V(R(G)) = R(V(G))$
- $(u : k, \gamma, v : l) \in E(G) \Leftrightarrow (R(u) : k, R \circ \gamma \circ R^{-1}, R(v) : l) \in E(R(G))$

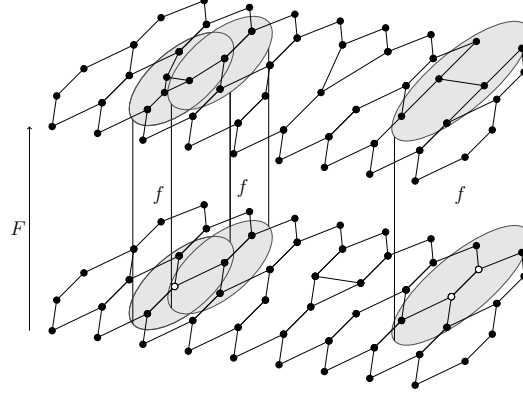


Figure 6: A *Causal Graph Dynamics*. The whole graph evolves in a causal (information propagates at a bounded speed) and homogeneous (same causes lead to same effects) manner. This was proven equivalent to applying a local function  $f$  to each subdisk of the input graph, producing small output graphs whose union make up the output graph.

- $(u : k) \in S(G) \Leftrightarrow (R(u) : k) \in S(R(G))$

Let  $b$  be an integer number, and  $\mathcal{F}(S)$  denote the finite subsets of a set  $S$ . We similarly define the isomorphism  $R^*$  specified by the isomorphism  $R$  as the function acting on graphs  $G$  such that  $V(G) \subseteq \mathcal{F}(V.\{\varepsilon, 1, \dots, b\})$ , so that  $R^*({u.i, v.j, \dots}) = \{R(u).i, R(v).j, \dots\}$ .

**Definition 4.2 (Consistent)** Consider two graphs  $G$  and  $H$ . Let  $K = V(G) \cap V(H)$  and  $L = V(G) \cup V(H)$ .  $G$  and  $H$  are consistent if and only if for all  $u : i$  in  $K : \pi$ , for all  $v : j$  in  $L : \pi$ ,

$$(u : i, \gamma, v : j) \in E(G) \vee (u : i) \in S(G) \iff (u : i, \gamma, v : j) \in E(H) \vee (u : i) \in S(H).$$

**Definition 4.3 (Local Rule)** A function  $f : \mathcal{D}^r \rightarrow \mathcal{G}$  is called a local rule if there exists some bound  $b$  such that:

- For all disks  $D$  and  $v' \in V(f(D)) \Rightarrow v' \subseteq V(D).\{\varepsilon, 1, \dots, b\}$ .
- For all graphs  $G$  and disks  $D_1, D_2 \subset G$ ,  $f(D_1)$  and  $f(D_2)$  are consistent.
- For all disks  $D$  and isomorphisms  $R$ ,  $f(R(D)) = R^*(f(D))$ , with  $R^*({u.i, v.j, \dots}) = \{R(u).i, R(v).j, \dots\}$ .

**Definition 4.4 (CGD)** [2, 3, 4] A function  $F$  from  $\mathcal{G}$  to  $\mathcal{G}$  is a localizable dynamics, a.k.a Causal Graph Dynamics, or CGD, if and only if there exists  $r$  a radius and  $f$  a local rule from  $\mathcal{D}^r$  to  $\mathcal{G}$  such that for every graph  $G$  in  $\mathcal{G}$ ,

$$F(G) = \bigcup_{v \in G} f(G_v^r).$$

To compute the image graph, a CGD could make use of the information carried out by the ports of the input graph. Thus, though the correspondence developed, they can readily be interpreted as “Causal Dynamics of Colored Complexes”. If we are interested in “Causal Dynamics of (Oriented) Complexes” instead, we need to make sure that  $F$  commutes with vertex-rotations.

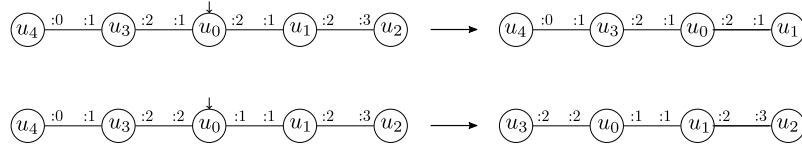


Figure 7: A non-rotation commuting local rule which induces a rotation commuting CGD.

**Definition 4.5 (Rotation-commuting dynamics)** *A CGD  $F$  is rotation-commuting if and only if for all graph  $G$  and all rotation sequence  $\bar{r}$  there exists a rotation sequence  $\bar{r}^*$  such that  $F(\bar{r}G) = \bar{r}^*F(G)$ . Such an  $\bar{r}^*$  is called a conjugate of  $\bar{r}$ .*

For local rules we will need a stronger version of this:

**Definition 4.6 (Strongly-rotation-commuting local rule)** *A local rule  $f$  is strongly-rotation-commuting if and only if for all intersecting pairs of disks  $G = D_1 \cup D_2$  and for all rotation sequence  $\bar{r}$ , the conjugate rotation sequences  $\bar{r}_1^*$  and  $\bar{r}_2^*$  defined through  $\bar{r}_i^*f(D_i) = f(\bar{r}D_i)$ ,  $i = 1, 2$  coincide on  $f(D_1) \cap f(D_2)$ .*

When is a CGD rotation-commuting? Can we decide, given the local rule  $f$  of a CGD  $F$ , whether  $F$  is rotation-commuting? The difficulty is that being rotation-commuting is a property of the global function  $F$ . Indeed, a first guess would be that  $F$  is rotation-commuting if and only if  $f$  is rotation-commuting, but this turns out to be false.

**Example 4.7 (Identity)** *Consider the local rule of radius 1 over graphs of degree 2 which acts as the identity in every cases but those given in Fig. 7. Because of these two cases, the local rule makes use the information carried out by the ports around the center of the neighborhood. It is not rotation-commuting. Yet, the CGD it induces is just the identity, which is trivially rotation-commuting.*

Thus, unfortunately, rotation-commuting  $F$  can be induced by non-rotation-commuting  $f$ . Still, there always exists a strongly-rotation-commuting  $f$  that induces  $F$ .

**Proposition 4.8** *Let  $F$  be a CGD.  $F$  is rotation-commuting if and only if there exists a strongly-rotation-commuting local rule  $f$  which induces  $F$ .*

*Proof.* (Outline). [ $\Leftarrow$ ] Trivial.

[ $\Rightarrow$ ] Given a rotation commuting CGD  $F$  induced by some local rule  $f$  that is not necessarily strongly rotation commuting itself, we construct a local rule  $\tilde{f}$  that is a strongly-rotation-commuting and still induces  $F$ . The construction uses the fact that  $F$  is rotation commuting to force  $\tilde{f}$  to adopt an homogeneous behavior over the sets of disks of the form  $\{\bar{r}D \mid \bar{r} \text{ a rotation sequence}\}$  (i.e rotation equivalent copies of the same disk).  $\square$

The point of this proposition is that having made this global property, local, makes it decidable.

**Proposition 4.9 (Decidability of rotation commutation)** *Given a local rule  $f$ , it is decidable whether  $f$  is strongly-rotation-commuting.*

*Proof.* There exists a simple algorithm to verify that  $f$  is strongly-rotation-commuting. Let  $r$  be the radius of  $f$ . We can check that for all disk  $D \in \mathcal{D}^r$  and for all vertex rotation  $r_u$ ,  $u \in V(D)$ , we have the existence of a rotation sequence  $\bar{r}$  such that  $f(r_u D) = \bar{r}^* f(D)$ .

As the graph  $f(D)$  is finite, there is finite number of rotation sequences  $\bar{r}^*$  to test. Notice that as  $f$  is a local rule, changing the names of the vertices in  $D$  will not change the structure of  $f(D)$  and thus we only have to test the commutation property on a finite set of disks.  $\square$

**Definition 4.10 (CDC)** *A Causal Dynamics of Complexes is a rotation-commuting CGD.*

## 5. Pachner Moves

*Bistellar moves.* Given a tetrahedron  $\Delta_3$ , there is a canonical way to obtain its border,  $\partial\Delta_3$ , as four glued triangles. In terms of graphs, given a single vertex of degree 4, there is a canonical way to obtain a graph made of four vertices of degree 3 that represents its border. This works as follows: 1. Interpret the vertex as a colored tetrahedron; so that each point has a color; 2. Reinterpret each facet as a vertex, and each gluing along a segment, as an edge between the ports of colors that of the points opposite the segment. This is the way we obtain:

**Definition 5.1** ( $\partial\Delta_{n+1}$ ) *We call the canonical sphere of dimension  $n$ , and denote  $\partial\Delta_{n+1}$ , the complete graph of size  $n+2$  having vertices  $v_0, \dots, v_{n+1}$  and edges of the form  $(v_i:j, s_{ij}, v_j:i)$  for  $i \neq j$  and  $i, j \in \{0, \dots, n+1\}$ .*

**Soundness.** All hinges are in normal form hence not torsioned.  $\square$

A triangle  $H$  can always be viewed as being a subcomplex of the boundary of a tetrahedron. Its complement with respect to the tetrahedron yields three other triangles  $H^*$ . More generally and in terms of graphs, whenever  $H$  is a subgraph of  $\partial\Delta_{n+1}$ , we can construct its complement  $H^*$  with respect to  $\partial\Delta_{n+1}$ .

When we have a triangle  $H$  lying inside a larger complex  $G$ , we can decide to replace  $H$  by  $H^*$  in  $G$ . This amounts to subdividing it into three, see Fig. 5. More generally and in terms of graphs, whenever  $H$  is an induced subgraph of  $\partial\Delta_{n+1}$  and lies inside a larger graph  $G$ , we can decide to replace  $H$  by  $H^*$  in  $G$ . A bistellar move does exactly that: it replaces a piece a sphere by its complement, it is intuitive therefore that it is a homeomorphism:

**Definition 5.2 (Bistellar move  $G.H$ )** *Let  $G$  be a graph and  $H$  be a subgraph of  $G$  such that  $H$  is a strict subset of a  $\partial\Delta_{n+1}$ , and  $(G \setminus H) \cap \partial\Delta_{n+1} = \emptyset$ . Let us call  $\bar{s}$  the symmetry sequence  $(s_{01})_{u \in H^*}$ . The graph  $G.H$  is the graph where  $H$  has been replaced by  $\bar{s}H^*$  as follows. First, add  $\bar{s}H^*$  to the graph. Second, for each edge  $e = (v:p, \gamma, u:q)$  between a vertex  $v$  of  $H$  and a vertex  $u$  of  $G \setminus H$ , notice there is a unique edge  $e' = (v':p', \gamma', v:p) \in E(\bar{s}\partial\Delta_{n+1})$ , and replace both  $e$  and  $e'$  by the edge  $(v':p', \gamma \circ \gamma', u:q)$ . Similarly, for every semi-edge  $e = (v:p)$  of  $H$ , notice there is a unique edge  $e' = (v':p', \gamma', v:p) \in E(\bar{s}\partial\Delta_{n+1})$ , and replace both  $e$  and  $e'$  by the semi-edge  $(v':p')$ . Third, remove the vertices of  $H$ .*



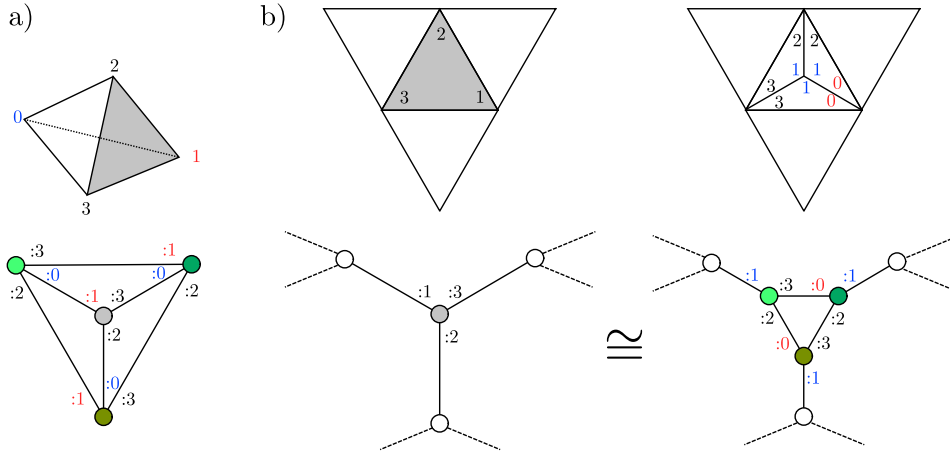


Figure 8: a) The canonical sphere of dimension 2. b) The bistellar move obtained by taking as the subgraph  $H$ , the single vertex in gray.

Notice that  $\bar{s}$  flips the orientation of  $H^*$  relative to  $H$  in order to match the orientation of  $G$ , and that the use of  $s_{01}$  for this purpose is without loss of generality, as one should also allow for vertex rotations.

*Shellings.* Given a 2-dimensional complex having a triangle with two of its sides on the boundary, we can decide to grow the complex by gluing two sides of a new triangle there. Similarly and in terms of graphs, whenever a vertex of degree  $n + 1$  has  $k$  free ports, we can decide to connect them with  $k$  ports of a new, otherwise unconnected vertex. A graph-local inverse shelling indeed consists in adding a new vertex to a graph by connecting it to a vertex having free ports:

**Definition 5.3 (Graph-local (inverse) shellings)** *Let  $G$  be a graph and  $u$  a vertex of  $G$ . Let  $S$  be a subset of at most  $n$  free ports of  $u$ , i.e. such that  $(u : p) \in S(G)$  for all  $p \in S$ . The graph  $G.S$  is the graph where a fresh vertex  $v$  has been added, as well as edges  $(u : p, s_{01}, v : s_{01}(p))$ . We say that  $G.S$  is an graph-local inverse shelling of  $G$ , and conversely that  $G$  is a graph-local shelling of  $G.S$ .*

There is difference, however, between this graph-local notion of shelling and the standard notion of shelling upon complexes. Indeed, as was pointed out in Section 2, in a 2-dimensional complex two boundary segments may be consecutive without this locality being apparent in the corresponding graph. Standard inverse shellings are definitely more general, as they allow gluing a fresh triangle there. Phrased in terms of graphs, they translate into:

**Definition 5.4 (Standard (inverse) shellings)** *Let  $G$  be a graph,  $u$  be a vertex of  $G$ , and  $F$  be a border  $k$ -face at  $u$ , having exactly  $n - k$  covering semi-edges  $(u_i : p_i)$ . The graph  $G.F$  is the graph where a fresh vertex  $v$  has been added, and each semi-edge  $(u_i : p_i)$  has been replaced by an edge  $(u_i : p_i, s_{01}, v : s_{01}(p_i))$ , without creating any torsion. We say that  $G.F$  is a standard inverse shelling of  $G$ , and conversely that  $G$  is a standard shelling of  $G.F$ .*

As an example of this definition, consider filling, with a new tetrahedron  $v$ , the hole in the ball

at the top of Fig. 9 a), as in the top of Fig. 9 c). The 0-face  $F$  stands for the geometrical point that will be covered by  $v$ . As the three covering semi-edges of  $F$  will be replaced by edges,  $F$  will no longer be a border face. Indeed, although  $v$  introduces a new semi-edge, that one is not a covering semi-edge of  $F$ .

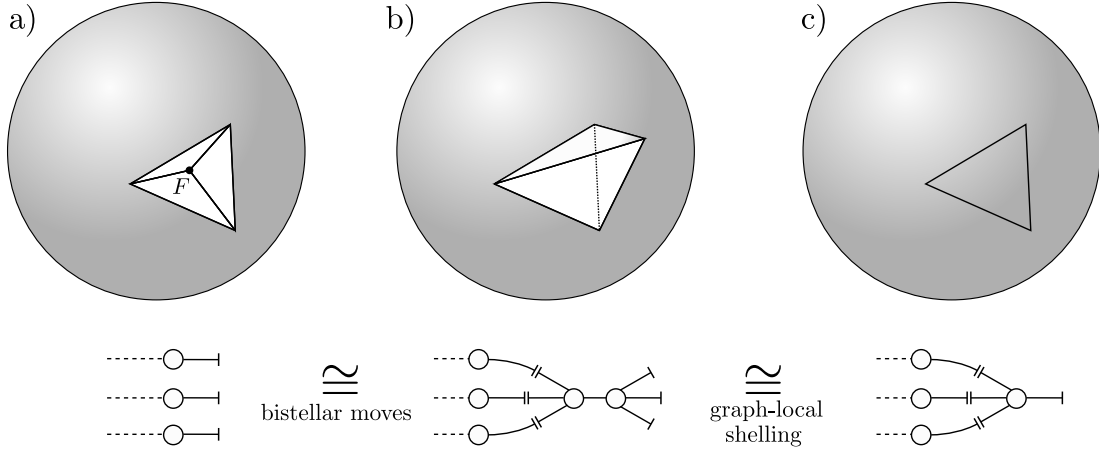


Figure 9: The standard inverse shelling obtained via bistellar moves and a graph-local shelling.

Fortunately, standard (inverse) shellings can always be recovered from a succession of rotations, Bistellar moves, graph-local (inverse) shellings:

**Definition 5.5 (Graph-local Pachner moves)** We call graph-local Pachner moves the union of vertex rotations, bistellar moves and graph-local (inverse) shellings.

**Proposition 5.6 (Recovering standard shellings)** Standard (inverse) shellings are compositions of graph-local Pachner moves.

*Proof.* Consider a graph  $G$  with a border  $k$ -face  $F$  having exactly  $n - k$  covering semi-edges as in Fig. 9 a). We want to perform the standard inverse shelling  $G.F$ , adding a fresh vertex  $v$ , using only graph-local Pachner moves. As an intermediate step, consider  $G'$  the graph  $G.F.S$  where  $S$  is the set of semi-edges of  $v$ , as in Fig. 9 b). The graphs  $G$  and  $G'$  are homeomorphic and have the same border, therefore they are related by a sequence of bistellar moves, as was shown by [6]. Finally, by a graph-local shelling we obtain  $G.F$  as in 9 c).  $\square$

In the setting of simplicial complexes, Pachner moves [15, 14] are well-known to generate all the homeomorphisms between combinatorial manifolds, and only the homeomorphisms. As a corollary of the above proposition the same holds true for graph-local Pachner moves:

**Definition 5.7 (Discrete manifold)** A graph  $G$  of degree  $|\pi| = n + 1$  is discrete manifold if and only if for each vertex  $u \in V(G)$ , there exists a sequence of graph-local Pachner moves sending  $Star(G, u)$  onto  $\Delta_n$ .

**Corollary 5.8 (Homeomorphism)** Consider  $M$  and  $M'$  two piecewise-linear manifolds, and let  $G$  and  $G'$  be the discrete manifolds obtained as their respective triangulations into simplicial

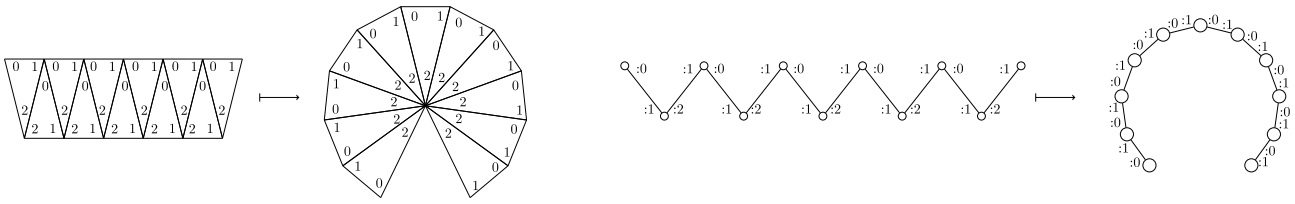


Figure 10: An unwanted evolution: sudden collapse in geometrical distance. *Left:* in terms of complexes. *Right:* In terms of graph representation.

*complexes.  $M$  and  $M'$  are piecewise-linearly homeomorphic if and only if  $G$  and  $G'$  are related by a sequence of graph-local Pachner moves.*

Notice that, albeit expensive computationally, it is decidable whether a  $n < 4$ -dimensional complex is homeomorphic to  $\Delta_n$ , see [13] and [12] (Proposition 3.1). Homeomorphism in general becomes undecidable for  $n \geq 4$  [13]. Notice also that discrete manifolds are not always simplicial complexes. For instance, the self-glued triangle is not a simplicial complex, as points of the same simplex get identified. In dimension  $n \leq 2$ , this remark seems innocuous, as any discrete manifold is related, via Pachner moves, to a simplicial complex. In dimension  $n = 3$ , we conjecture that this is still the case if and only if each simplex has no more than two identified points, and that one extra move suffices to make this true in all cases.

## 6. Causal Dynamics of Discrete Manifolds

The results in this section crucially rely on the following lemma:

**Lemma 6.1 (Past subgraph)** [3] *Consider  $F$  a CGD induced by the local rule  $f$  of radius  $r$  (i.e. diameter  $d = 2r + 1$ ). Consider a graph  $G$ , a vertex  $v$  in  $G$ , a vertex  $v'$  in  $f(G_v^r)$ , and a disk  $F(G_v^{r'})$  (i.e. of diameter  $d' = 2r' + 1$ ). Then this disk is a subgraph of  $F(G_v^{2rr'+r+r'})$ . Notice that the disk  $G_v^{2rr'+r+r'}$  has diameter  $d'' = d'd$ .*

*Bounded-star preserving.* We will now restrict to CGD so that they preserve the property of a graph being bounded-star. Indeed, we have seen that graph distance between two vertices does not always correspond to the geometrical distance between the two triangles that they represent. With CDC, we were guaranteeing that information does not propagate too fast with respect to the graph distance, but not with respect to the geometrical distance. The fact that the geometrical distance is less than or equal to the graph distance is falsely reassuring: the discrepancy can still lead to unwanted phenomenon as depicted in Fig. 10.

Of course we may choose not to care about geometrical distance. But if we do care, then we must not let that happen. One solution is to make the graphs are  $s$ -bounded-star. This will relate the geometrical distance and the graph distance by a factor  $s$ . As a consequence, the guarantee that information does not propagate too fast with respect to graph distance will induce its counterpart in geometrical distance. This will forbid the sudden collapse in

geometrical distance of Fig. 10. More generally it will enforce a bounded-density of information principle. Of course, we must then ensure that the CGD we use preserve  $s$ -bounded-star graphs:

**Definition 6.2 (Bounded-star preserving)** *A CGD  $F$  is bounded-star preserving with bound  $s$  if and only if for all  $s$ -bounded-star graph  $G$ ,  $F(G)$  is also  $s$ -bounded-star.*

**Proposition 6.3** *Consider  $F$  a CGD induced by a local rule  $f$  of radius  $r$ .  $F$  is bounded-star preserving with bound  $s = 2r'$  if and only if for any  $s$ -bounded-star  $D$  in  $\mathcal{D}^{2rr'+r+r'}$ ,  $F(D)$  is also  $s$ -bounded-star. Therefore, given a local rule  $f$ , it is decidable whether its induced  $F$  is bounded-star preserving.*

*Proof.*  $[\Rightarrow]$  Trivial.  $[\Leftarrow]$  By contradiction suppose that there is an  $s$ -bounded-star graph  $G$  such that  $F(G)$  has an hinge  $h$  of size  $s' = 2r' + 1$ , and yet that all  $s$ -bounded-star disks of radius  $2rr' + r + r'$  are mapped into  $s$ -bounded-star graphs. Next, take  $v'$  in the middle of  $h$ , and  $v$  in  $G$  such that  $v'$  in  $f(G_v^r)$ . By Lemma 6.1,  $h$  appears in  $F(G_v^{2rr'+r+r'})$ , which contradicts our hypothesis.  $\square$

*Torsion-free preserving.* Second, amongst bounded-star preserving CGD, we will restrict to those that preserve the property of not having torsion.

**Definition 6.4 (Torsion-free preserving)** *An  $s$ -bounded-star preserving CGD  $F$  is torsion-free preserving if and only if for all  $s$ -bounded-star graph  $G$  without torsion,  $F(G)$  is without torsion.*

**Proposition 6.5** *Consider  $F$  an  $s$ -bounded-star CGD induced by a local rule  $f$  of radius  $r$ .  $F$  is torsion-free preserving if and only if for any  $s$ -bounded-star  $D$  in  $\mathcal{D}^{2rr'+r+r'}$  without torsion,  $F(D)$  is also without torsion. Therefore, given a local rule  $f$ , it is decidable whether its induced  $F$  is torsion-free preserving.*

*Proof.* As for Proposition 6.3.  $\square$

*Discrete-manifold preserving.* Third, amongst torsion-free bounded-star preserving CGD, we will restrict to those that preserve the property of being a discrete manifold.

**Definition 6.6 (Discrete-manifold preserving)** *An torsion-free  $s$ -bounded-star preserving CGD  $F$  is discrete-manifold preserving if and only if for all  $s$ -bounded-star discrete manifold  $G$ , then  $F(G)$  is a discrete manifold.*

**Proposition 6.7** *Consider  $F$  a torsion-free  $s$ -bounded-star preserving CGD induced by a local rule  $f$  of radius  $r$ .  $F$  is discrete-manifold preserving if and only if for any  $s$ -bounded-star discrete manifold  $D$  in  $\mathcal{D}^{2rr'+r+r'}$ ,  $F(D)$  is also a discrete manifold. Therefore, in dimension  $n \leq 3$ , given a local rule  $f$ , it is decidable whether its induced  $F$  is discrete-manifold preserving.*

*Proof.* As for Proposition 6.3. Checking whether  $F(G_v^{2rr'+r+r'})$  is a discrete-manifold is indeed possible in dimension  $n \leq 3$ , cf. [13] and [12] (Proposition 3.1).  $\square$

**Definition 6.8 (CDDM)** *A Causal Dynamics of Discrete Manifolds is a torsion-free  $s$ -bounded-star discrete-manifold preserving CGD.*

## 7. Conclusion

*Results in context.* In [2, 3, 4] two of the authors, together with Dowek and Nesme, generalized cellular automata theory to arbitrary, time-varying graphs. I.e. they formalized the intuitive idea of a labeled graph which evolves in time, subject to two natural constraints: the evolution does not propagate information too fast; and it acts everywhere the same. Some fundamental facts of Cellular Automata theory were shown to carry through, for instance that these Causal Graph Dynamics (CGD) admit a characterization as continuous functions and that their inverses are also CGD.

The motivation for developing these CGD was to “free Cellular Automata off the grid”, so as to be able to model any situation where agents interact with their neighbors synchronously, leading to a global dynamics in which the states of the agents can change, but also their topology, i.e. the notion of who is next to whom. A first motivating example was that of a mobile phone network. A second example was that of particles lying on a surface and interacting with one another, but whose distribution influences the topology the surface (cf. Heat diffusion in a dilating material, discretized General Relativity [16]). However, CGD seemed quite appropriate for modeling the first situation (or at least a stochastic version of it), but not the second. Indeed, having freed Cellular Automata off the grid, one could no longer interpret arbitrary graphs as surface, in general.

The present paper solves this problem by proposing a rigorous definition of “Causal Dynamics of Complexes” (CDC) and “Causal Dynamics of Discrete Manifolds” (CDDM). Essentially this shows that CGD can be “tied up again to complexes and even to discrete manifolds”, at the cost of additional restrictions: rotation-commutation (CDC), bounded-star preservation, torsion-free preservation, discrete-manifold preservation (CDDM). The first restriction allows us to freely rotate simplices. The second restriction allows us to map geometrical distances into graph distances. The third restriction makes sure that no torsion gets introduced. The fourth restriction makes sure that the neighborhood of every point remains a ball. The first and second are decidable independently. Imposing the second makes the third decidable, and fourth, but in dimensions  $n < 4$  only. An earlier version investigated the 2-dimensional case in order to gain intuitions [5]. This paper provides its non-trivial generalization to  $n$  dimensions: the third and fourth conditions, for instance, were vacuous in the 2-dimensional case. In order to tackle it, we translated the notion of manifold homeomorphism the vocabulary of labeled graphs.

Notice that, since these CDC and CDDM are a specialization of CGD by construction, several theoretical results about them follow as mere corollary from [2, 3, 4] – that we have not mentioned. For instance, CDC/CDDM of radius 1 are universal, composable, characterized as the set of continuous functions from complexes to complexes with respect to the Gromov-Hausdorff-Cantor metric upon isomorphism classes. These results deserve to be made more explicit, but they already are indicators of the generality of the model.

*Comparison with Crystallizations/Gems.* This paper conducted a thorough comparison between discrete geometries and graphs, by investigating the natural encoding of complexes into

their dual graphs. This encoding was made precise. The discrepancy between geometrical distance and graph distance was analyzed. The notion of manifold was characterized, through a graph-local version of Pachner moves. Another, very well-developed correspondence between simplicial complexes and labeled graphs goes under the name of ‘crystallizations’ [7]. Phrased in the vocabulary of Def. 2.1, this means restricting to bipartite graphs (i.e. w.r.t. to labels in  $\Sigma = \{0, 1\}$ , say) that are edge-colored (i.e. edges are between equal ports) and have, as gluings, the identity. Because this gluing is an even permutation, 0-labeled vertices are oriented one-way, and 1-labeled vertices are oriented the other way. These constraints may seem cumbersome at first; for instance constructing a sphere becomes much more involved than Def. 5.1. Yet, a closer look shows a key advantage: by construction, crystallizations do not have torsion. The subset of crystallizations that represent discrete manifolds is usually referred to as ‘gems’ (i.e. graph-encoded manifolds). Homeomorphism between gems can again be captured by moves. Traditionally the moves that have been studied are the so-called ‘dipole moves’, but unfortunately these are not graph-local (a global condition needs be checked prior to application). Lately, however, [9] developed an equivalent of Bistellar moves, called ‘cross-flips moves’, which captures homeomorphism between closed discrete manifolds, in a graph-local way. This has been extended to discrete manifolds with borders in [10] – but the (inverse) shellings are again not graph-local. Yet, [10] also contains the gems-version the result by [6] that allowed us to prove that graph-local (inverse) shellings are enough. Thus, all the results of this paper can readily be ported to crystallizations/gems. Still, there will be a price to pay: the number of cross-flip moves is in  $O(2^n)$  [10], whereas bistellar moves grow as  $O(n)$ .

## Acknowledgements

This work has been funded by the ANR-12-BS02-007-01 TARMAC grant and the STICAmSud project 16STIC05 FoQCoSS. The authors acknowledge enlightening discussions Gilles Dowek, Ivan Izmetiev, Pascal Lienhardt, Luca Lionni, Christian Mercat, Michele Mulazzani, Zizhu Wang.

## References

- [1] J. AMBJØRN, J. JURKIEWICZ, R. LOLL, Emergence of a 4D world from causal quantum gravity. *Physical Review Letters* 93 (2004) 13, 131301.
- [2] P. ARRIGHI, G. DOWEK, Causal graph dynamics. In: A. CZUMAJ, K. MEHLHORN, A. M. PITTS, R. WATTENHOFER (eds.), *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*. Lecture Notes in Computer Science 7392, Springer, 2012, 54–66.
- [3] P. ARRIGHI, G. DOWEK, Causal graph dynamics (long version). *Information and Computation* 223 (2013), 78–93.
- [4] P. ARRIGHI, S. MARTIEL, V. NESME, Cellular automata over generalized Cayley graphs. *Mathematical Structures in Computer Science* 18 (2018), 340–383.

- [5] P. ARRIGHI, S. MARTIEL, Z. WANG, Causal dynamics of discrete surfaces. In: *Proceedings of Developments in Computational Models (DCM 2013), Buenos Aires, August 2013*. EPTCS 144, 2013, 30–40.
- [6] M. R. CASALI, A note about bistellar operations on PL-manifolds with boundary. *Geometriae Dedicata* 56 (1995) 3, 257–262.
- [7] M. FERRI, C. GAGLIARDI, L. GRASSELLI, A graph-theoretical representation of PL-manifolds – a survey on crystallizations. *Aequationes Mathematicae* 31 (1986) 1, 121–141.
- [8] J. GIAVITTO, A. SPICHER, Topological rewriting and the geometrization of programming. *Physica D: Nonlinear Phenomena* 237 (2008) 9, 1302–1314.
- [9] I. IZMESTIEV, S. KLEE, I. NOVIK, Simplicial moves on balanced complexes. *Advances in Mathematics* 320 (2017), 82–114.
- [10] M. JUHNKE-KUBITZKE, L. VENTURELLO, Balanced shellings and moves on balanced manifolds. *Preprint arXiv:1804.06270* (2018).
- [11] A. KLALES, D. CIANCI, Z. NEEDELL, D. A. MEYER, P. J. LOVE, Lattice gas simulations of dynamical geometry in two dimensions. *Physical Review E* 82 (2010) 4, 046705.
- [12] G. KUPERBERG, Algorithmic homeomorphism of 3-manifolds as a corollary of geometrization. *Preprint arXiv:1508.06720* (2015).
- [13] F. LAZARUS, A. DE MESMAY, Undecidability in topology. 2017.  
<http://www.gipsa-lab.fr/~francis.lazarus/Enseignement/compuTopo7.pdf>
- [14] W. R. LICKORISH, Simplicial moves on complexes and manifolds. *Geometry and Topology Monographs* 2 (1999) 299–320, 314.
- [15] U. PACHNER, PL homeomorphic manifolds are equivalent by elementary shellings. *European Journal of Combinatorics* 12 (1991) 2, 129–145.
- [16] R. SORKIN, Time-evolution problem in Regge calculus. *Physical Review D* 12 (1975) 2, 385–396.





# ON REGULAR EXPRESSIONS WITH BACKREFERENCES AND TRANSDUCERS

Martin Berglund<sup>(A)</sup>      Frank Drewes<sup>(B)</sup>  
Brink van der Merwe<sup>(C)</sup>

<sup>(A)</sup>Department of Information Science, Center for AI Research (CSIR),  
Stellenbosch University, South Africa  
`pberglund@sun.ac.za`

<sup>(B)</sup>Department of Computing Science, Umeå University, Sweden  
`drewes@cs.umu.se`

<sup>(C)</sup>Department of Computer Science, Stellenbosch University, South Africa  
`abvdm@cs.sun.ac.za`

## **Abstract**

*Modern regular expression matching software features many extensions, some general, while some are very narrowly specified. Here we consider the generalization of adding a class of operators which can be described by, e.g. finite-state transducers. Combined with backreferences, they enable new classes of languages to be matched. The addition of finite-state transducers is shown to make membership testing undecidable. Following this result, we study the complexity of membership testing for various restricted cases of the model.*

## **1. Introduction**

In this paper we consider generalizations of various common feature additions in practical regular expression matching software. Notably we include expressions with backreferences (which we abbreviate REb here), an extension which allows the regular expression to “capture” literal substrings as part of its matching procedure, and then “backreference” a previously captured string to match an exact copy of it in a different position of the string, as investigated in [3], [5] and [9], for example. Furthermore, in most matching engines (Java, Perl, etc.) the subexpression `(?i)` matches the empty string, but enables *case-insensitive* matching for a subexpression, meaning that `(?i)(.*)\1` matches any  $\alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_n$  where, for each  $i$ ,  $\alpha_i$  and  $\beta_i$  are the same letter up to one (perhaps) being lowercase and the other uppercase. Several similar features exist (such as collating different representations of Unicode symbols), which can all be naturally expressed as a transduction of the matched string. To generalize this we here permit *transducer subexpressions*, obtained by allowing the application of some string-to-string transducer to subexpressions. A transducer subexpression  $t(E)$  describes the language of strings obtained by applying the transducer  $t$  to the language matched by  $E$ . We

call these extended expressions, obtained by adding backreferences and transducers, regular expressions with backreferences and transducers (REbt). For the most part, the transducers considered will be finite-state transducers or restrictions thereof.

Beyond the transducer-like features of existing engines the REbt (and the various restricted subclasses we consider) can also describe some frequently encountered non-context-free languages. According to Dassow et al. [4] the three most commonly encountered non-context-free features in formal languages are reduplication, i.e., the ability to express languages of the form  $L_{RD} = \{ww \mid w \in \Sigma^*\}$ , multiple agreements, described by languages of the form  $L_{MA} = \{a^n b^n c^n \mid n \geq 1\}$ , and cross agreements, as given by languages of the form  $L_{CA} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$ . The language  $L_{RD}$  can be described by REb, but neither  $L_{MA}$  nor  $L_{CA}$  can, which for a restricted class of REb follows from the pumping lemma in [3], and can be more generally derived from [9]. The language matched by the example Java expression `(?i)(.*)\1` cannot be matched by any REb either, as is evidenced by the sublanguage  $\{a^n b A^n B \mid n \geq 0\}$  combined with the fact that the languages matched by REb are closed under intersection with regular languages, as shown in Theorem 21 in [9]. The REbt matching these languages are quite simple, but the full formalism turns out to be very powerful. This establishes the goal of the paper, i.e., finding natural restrictions of REbt which can still match  $L_{MA}$  and  $L_{CA}$ , can be tested for membership with a computational complexity not too distant from REb, and may be considered “natural”.

After definitions given in Section 2, and the unrestricted case being shown to have undecidable membership in Section 3, the remaining sections explore various restrictions: Section 4 forbids the capture of transducer preimages and considers permitting only non-deleting transducers. Section 5 forbids transducers in capturing cycles (where a capturing cycle captures a submatch and then later backreferences this capture as part of another submatch by the same capturing subexpression), and requires the transducers occurring in captures to be functional. Finally, Section 6 considers permitting only a single top-level transducer.

## 2. Definitions

Denote by  $\mathbb{N}$  the set of natural numbers, excluding 0,  $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ , and by  $[k]$ , with  $k \in \mathbb{N}$ , the set  $\{j \mid 1 \leq j \leq k\}$ . An alphabet is a finite set of symbols. For sets  $S$  and  $T$  we write  $S \uplus T$  to denote the union of these sets, assumed to be disjoint. Let  $\varepsilon$  denote the empty string and  $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ , where  $\Sigma$  is an alphabet, and for a string  $w \in \Sigma^*$ , let  $\text{substr}(w)$  be the set of all substrings of  $w$ , i.e.,  $\text{substr}(\varepsilon) = \{\varepsilon\}$ , and if  $w_i \in \Sigma$  for  $1 \leq i \leq n$ , then  $\text{substr}(w_1 \dots w_n) = \{\varepsilon\} \cup \{w_i \dots w_j \mid 1 \leq i \leq j \leq n\}$ , in particular,  $\varepsilon, w \in \text{substr}(w)$ . Given a notion of expressions, defined inductively, we denote by  $\text{subexps}(E)$  the set of all subexpressions of the expression  $E$ , that is,  $\text{subexps}(E)$  is the set of expressions used to obtain  $E$  inductively, including duplicate expressions when the same subexpression appears at different places in  $E$ .

For a partial function  $f: A \rightarrow B$ , let  $\text{dom}(f)$  denote its domain,  $\text{range}(f)$  its range, and  $g = f[x \mapsto y]$  denote the partial function such that  $g(x) = y$  but  $g(z) = f(z)$  for all  $z \neq x$ . A partial function  $f$  with  $\text{dom}(f) = \emptyset$  is denoted by  $\perp$ . Let  $f[x \mapsto \perp]$  denote the function

resulting when removing  $x$  from the domain of  $f$ . For a set  $A$ , we denote its cardinality by  $|A|$ .

To keep our notion of regular expressions with backreferences and transducers general, we define it relative to a set  $\Theta$  of string transducers. When we say that  $\Theta$  is a class of transducers we mean that every element  $t$  of  $\Theta$  denotes a transduction  $\mathcal{L}(t)$  on  $\Sigma^*$  for some alphabet  $\Sigma$ , i.e.,  $\mathcal{L}(t)$  is a binary relation  $\mathcal{L}(t) \subseteq \Sigma^* \times \Sigma^*$ . As a transducer  $t \in \Theta$  is a denotation of a transduction, it has length, namely its length when written down as a string. We denote this length by  $|t|$ . Clearly, classes of transducers that specify the same transductions may differ regarding, e.g. their succinctness, and thus also with respect to their computational complexity.

We say that a transducer  $t$  is:

- *non-deleting* if there is a constant  $c \in \mathbb{N}$  such that  $|u| \leq c|v| + c$  for all  $(u, v) \in \mathcal{L}(t)$ , we call the smallest such  $c$  the *non-deletion constant* of  $t$ ,
- *non-generating* if the transducer defined by the inverse relation  $\mathcal{L}(t)^{-1}$  is non-deleting, and
- *functional* if  $\mathcal{L}(t)$  is a partial function, i.e.,  $|\{v \in \Sigma^* \mid (u, v) \in \mathcal{L}(t)\}| \leq 1$  for all  $u \in \Sigma^*$ .

**Definition 2.1** *Let  $\Theta$  be a class of transducers. For input and backreference alphabets  $\Sigma$  and  $\Phi$ ,  $\alpha \in \Sigma_\varepsilon$ ,  $\phi \in \Phi$ , and  $t \in \Theta$  a transducer on  $\Sigma$ , the set of regular expressions with backreferences and transducers (over  $\Theta$ ),  $REbt_{\Sigma, \Phi}$ , is obtained inductively from the following subexpressions: (1)  $\emptyset$ ; (2)  $\alpha$ ; (3)  $(F \mid G)$ ; (4)  $(F \cdot G)$ ; (5)  $(F^*)$ ; (6)  $(\uparrow_\phi)$ ; (7)  $([\phi F]_\phi)$ ; and (8)  $t(F)$ , where  $F, G \in REbt_{\Sigma, \Phi}$ . We call the  $REbt$  that can be constructed using rules 1–7 regular expressions with backreferences ( $REb$  or  $REb_{\Sigma, \Phi}$ ), using 1–5 and 8, regular expressions with transducers ( $REt$  or  $REt_\Sigma$ ), and, using 1–5, regular expressions ( $RE$  or  $RE_\Sigma$ ).*

For  $E \in REbt$ , we denote by  $|E|$  the length of  $E$  as a string, but letting each transducer symbol  $t$  in  $|E|$  contribute length  $|t|$  (i.e., not just 1), and by  $env_{\Phi, \Sigma}$  (or simply  $env$ , when  $\Phi$  and  $\Sigma$  is understood) the set of all partial functions from  $\Phi$  to  $\Sigma^*$ . We refer to these partial functions as *environments*, since they keep track of which substring, in  $\Sigma^*$ , from the input string, is bound to a given backreference symbol  $\phi \in \Phi$ . The empty environment, i.e., the partial function in  $env$  with empty domain, is denoted by  $\perp$ .

**Definition 2.2** *For  $E \in REbt_{\Sigma, \Phi}$ , we define the matching relation  $\mathcal{M}(E) \subseteq env_{\Sigma, \Phi} \times \Sigma^* \times env_{\Sigma, \Phi}$  inductively on the structure of  $E$ , as follows.*

1.  $\emptyset$  if  $E = \emptyset$ ;
2.  $\{(f, \alpha, f) \mid f \in env_{\Sigma, \Phi}\}$  if  $E = \alpha$  with  $\alpha \in \Sigma_\varepsilon$ ;
3.  $\mathcal{M}(F) \cup \mathcal{M}(G)$  if  $E = (F \mid G)$ ;
4.  $\{(f, vw, g) \mid (f, v, f') \in \mathcal{M}(F), (f', w, g) \in \mathcal{M}(G)\}$  if  $E = (F \cdot G)$ ;
5.  $\mathcal{M}(\varepsilon) \cup \{(f, vw, g) \mid (f, v, f') \in \mathcal{M}(F^*), (f', w, g) \in \mathcal{M}(F)\}$ , or the least fixed point of  $\mathcal{M}(E) = \mathcal{M}(\varepsilon) \cup \mathcal{M}(E \cdot F)$ , if  $E = (F^*)$ ;
6.  $\{(f, w, f'[\phi \mapsto w]) \mid (f, w, f') \in \mathcal{M}(F)\}$  if  $E = ([\phi F]_\phi)$  with  $\phi \in \Phi$ ;
7.  $\{(f, w, f) \mid f \in env_{\Sigma, \Phi}, f(\phi) = w\}$  if  $E = (\uparrow_\phi)$ ;
8.  $\{(f, w, f') \mid (f, v, f') \in \mathcal{M}(F), (v, w) \in \mathcal{L}(t)\}$  if  $E = (t(F))$  for some  $t \in \Theta$ .

The language matched by a REbt  $E$ , denoted  $\mathcal{L}(E)$ , is defined as the set  $\mathcal{L}(E) = \{w \mid (\perp, w, f) \in \mathcal{M}(E), f \in \text{env}_{\Sigma, \Phi}\}$ .

Let  $\text{td}(E)$  denote the set of all transducers occurring in  $E$ . For technical convenience, we assume that every transducer in  $\text{td}(E)$  is referred to only once in  $E$ . Hence, two distinct subexpressions  $t(F)$  and  $t'(F')$  have  $t \neq t'$  even though we may of course have  $\mathcal{L}(t) = \mathcal{L}(t')$ .

As usual, when writing an expression as a string some parentheses may be elided using the rule that Kleene closure ‘\*’ takes precedence over concatenation ‘·’, which takes precedence over union ‘|’. In addition, outermost parenthesis and parenthesis in subexpressions of the form  $([\phi E]_\phi)$  and  $(\uparrow_\phi)$ , may be dropped, and  $E_1 \cdot E_2$  abbreviated as  $E_1 E_2$ . Naturally, the brackets which denote a capturing group may not be elided.

**Example 2.3** A simple class of transducers over  $\Sigma^*$ , corresponding to finite-state transducers with only one state, is the set of all  $t = (\alpha_1 : \beta_1, \dots, \alpha_k : \beta_k)$  where  $k \in \mathbb{N}$  and  $\alpha_1, \beta_1, \dots, \alpha_k, \beta_k \in \Sigma_\varepsilon$ . The transduction denoted by  $t$  is

$$\mathcal{L}(t) = \{(\alpha_{i_1} \cdots \alpha_{i_n}, \beta_{i_1} \cdots \beta_{i_n}) \mid n \in \mathbb{N}, i_1, \dots, i_n \in [k]\}.$$

Taking  $E_{MA} = [{}_1 a^*]_1 t_b(\uparrow_1) t_c(\uparrow_1)$  and  $E_{CA} = [{}_1 a^*]_1 [{}_2 b^*]_2 t_c(\uparrow_1) t_d(\uparrow_2)$  from  $\text{REbt}_{\{a,b,c,d\}, \{1,2\}}$  with  $t_b = a : b$ ,  $t_c = a : c$  and  $t_d = b : d$  yields  $\mathcal{L}(E_{MA}) = L_{MA}$  and  $\mathcal{L}(E_{CA}) = L_{CA}$ , with  $L_{MA}$  and  $L_{CA}$  as given in the introduction.

We often let  $\Sigma$  and  $\Phi$  indicate arbitrary input and backreference alphabets respectively, and may then also drop them, writing REbt instead of  $\text{REbt}_{\Sigma, \Phi}$ .

The subset REb of REbt is equivalent to the semantics originally given by Aho in [1], which agrees fully with the behavior of many popular software implementations (e.g. Boost, the .NET standard library implementation, the PCRE library [2]), and form a superset of many more (e.g. the Java and Python implementations). The semantics considered by Schmid in [9] is also closely related, with one difference being that subexpressions of the form  $[\phi \cdots \uparrow_\phi \cdots]_\phi$  are not permitted by Schmid (but are here, in Aho, and in most implementations). Schmid also differs from Aho, while agreeing with other important theoretical work [3], in having  $\uparrow_\phi$  match the empty string if  $\phi$  has not yet been captured (i.e. they let  $\mathcal{L}(E) = \{w \mid (\perp_\varepsilon, w, f) \in \mathcal{M}(E)\}$  where  $\perp_\varepsilon(\phi) = \varepsilon$  for all  $\phi \in \Phi$ ). We again adopt the Aho approach to align with the software practice, also noting that Aho semantics can simulate the use of  $\perp_\varepsilon$ , by first “initializing” all symbols from  $\Phi$  to  $\varepsilon$  (using a leading sequence of subexpressions  $[\phi \varepsilon]_\phi$  for all  $\phi \in \Phi$ ).

### 3. Unrestricted Language Classes

The use of transducers without severe restrictions unsurprisingly gives rise to a Turing complete formalism. To make this precise, let FST denote the class of all one-way finite-state string transducers. More precisely, FST is the set of all  $t = (Q, \Sigma, q_0, \delta, F)$  where (1)  $Q$  is a *finite*

set of states, (2)  $\Sigma$  is the input and output alphabet, (3)  $q_0 \in Q$  is the initial state, (4)  $\delta \subseteq Q \times \Sigma_\varepsilon \times \Sigma_\varepsilon \times Q$  is the transition relation, and (5)  $F \subseteq Q$  is the set of final states.

A computation of such an FST  $t$  is a sequence  $(q_1, \alpha_1, \beta_1, q'_1), \dots, (q_n, \alpha_n, \beta_n, q'_n)$  of zero or more transitions, having  $q'_i = q_{i+1}$  for all  $i \in [n-1]$ . The transduction  $\mathcal{L}(t) \subseteq \Sigma^* \times \Sigma^*$  consists of all  $(v, w)$  such that there exists a computation  $(q_1, \alpha_1, \beta_1, q'_1), \dots, (q_n, \alpha_n, \beta_n, q'_n)$  with  $q_0 = q_1$ ,  $q'_n \in F$ ,  $v = \alpha_1 \cdots \alpha_n$  and  $w = \beta_1 \cdots \beta_n$ .

**Theorem 3.1** *For every recursively enumerable language  $L$  there exists an  $E \in \text{REbt}$  over FST such that  $\mathcal{L}(E) = L$ . Consequently the membership problem is undecidable for REbt over FST.*

*Proof.* For a Turing machine  $M$  with input alphabet  $\Gamma$ , choose a representation of the configurations of  $M$  as strings  $w \in \Sigma^*$ , where  $\Sigma \supseteq \Gamma$ , such that we can construct

- a transducer  $t_{\text{init}} \in \text{FST}$  such that  $(w, c) \in \mathcal{L}(t_{\text{init}})$  if  $c \in \Sigma^*$  is the initial configuration of  $M$  when starting with  $w \in \Gamma^*$  as input,
- a transducer  $t_{\text{acc}} \in \text{FST}$  such that  $(c, c) \in \mathcal{L}(t_{\text{acc}})$  if  $c$  is the concatenation of configurations of  $M$ , with only the last configuration being accepting, and
- a transducer  $t_{\text{step}} \in \text{FST}$  such that  $(c, c') \in \mathcal{L}(t_{\text{step}})$  if  $M$  can go from the configuration  $c$  to the configuration  $c'$  in a single step.

This is easy for any reasonable string representation of configurations:  $t_{\text{init}}$  adds a tape head and state at the front,  $t_{\text{acc}}$  checks for an accepting state, and  $t_{\text{step}}$  performs one of a finite number of constant substring rewritings around the tape head, implementing the rules of  $M$ .

Then, take  $\Phi = \{\phi\}$  and define  $E_u \in \text{REbt}_{\Sigma, \Phi}$  to be

$$[\phi\Gamma^*]_\phi D([\phi t_{\text{init}}(\uparrow\phi)]_\phi t_{\text{acc}}([\phi t_{\text{step}}(\uparrow\phi)]_\phi^*)),$$

where  $D \in \text{FST}$  deletes the entire input (and outputs  $\varepsilon$ ).

Thus, the first subexpression selects and captures any input string  $w$ . The subexpression  $D(\dots)$  simulates a computation of  $M$  on  $w$  to either fail or, if  $M$  accepts, yield  $\varepsilon$ .  $\square$

**Corollary 3.2** *Theorem 3.1 holds even if  $E$  is required to be a REbt over functional non-generating FSTs.*

*Proof.* The FSTs used in the proof of Theorem 3.1 are already non-generating. Further, we can, without loss of generality, pick  $M$  to be a deterministic Turing machine, at which point the natural way of constructing the transducers will make them functional.  $\square$

The rest of the paper studies restrictions of REbt which we consider to be natural, and which make matching more tractable while including REb and retaining the ability to match e.g.  $L_{\text{MA}}$  and  $L_{\text{CA}}$ . Tractability of restrictions must be judged relative to the known NP-completeness of the uniform membership problem for REb [1], forming a lower bound. The non-uniform membership problem for REb can be decided in PTIME, and if  $|\Phi|$  is bounded, the same holds true for the uniform membership problem.

**Algorithm 1** Membership decision procedure for  $\text{REb}_{\Sigma, \Phi}$ 


---

```

procedure MEMBERSHIP( $w \in \Sigma^*, E \in \text{REb}_{\Sigma, \Phi}$ )
  ▷ Returns true if  $w \in \mathcal{L}(E)$ 
  Let  $\text{env}(w) = \{f \in \text{env}_{\Sigma, \Phi} \mid \text{range}(f) \subseteq \text{substr}(w)\}$ 
  Let  $T : \text{env}(w) \times \text{substr}(w) \times \text{subexps}(E) \times \text{env}(w) \rightarrow \{\text{true}, \text{false}\}$ 
   $T(f, v, E, f') \leftarrow \text{false}$  for all  $(f, v, E, f') \in \text{dom}(T)$ 
   $T(f, \alpha, \alpha, f) \leftarrow \text{true}$  for all  $\alpha \in \Sigma_\varepsilon$  and  $f \in \text{env}(w)$ 
   $T(f, \varepsilon, F^*, f) \leftarrow \text{true}$  for all  $F^*$  and  $f$ 
  repeat
    if  $T(f, v_1, F^*, g) \wedge T(g, v_2, F, f') = \text{true}$  for some  $v_1, v_2$  then and  $g$ 
       $T(f, v_1v_2, F^*, f') \leftarrow \text{true}$ 
    if  $T(f, v_1, F, g) \wedge T(g, v_2, G, f') = \text{true}$  for some  $v_1, v_2$  and  $g$  then
       $T(f, v_1v_2, F \cdot G, f') \leftarrow \text{true}$ 
    if  $T(f, v, F, f') \vee T(f, v, G, f') = \text{true}$  then
       $T(f, v, F \mid G, f') \leftarrow \text{true}$ 
    if  $T(f, v, F, g) = \text{true}$  then
       $T(f, v, [\phi F]_\phi, g[\phi \mapsto v]) \leftarrow \text{true}$ 
    if  $f(\phi) = v$  then
       $T(f, v, \uparrow_\phi, f) \leftarrow \text{true}$ 
  until no additional function values of  $T$  were set to true
  return true if  $T(\perp, w, E, f)$  equals true for some  $f$ 

```

---

**Lemma 3.3** For  $E \in \text{REb}_{\Sigma, \Phi}$  and  $w \in \Sigma^*$  we may decide whether  $w \in \mathcal{L}(E)$  by using Algorithm 1. This algorithm runs in time polynomial in  $|w|$  and  $|E|$ , with a polynomial of degree  $O(|\Phi|)$ .

*Proof.* The steps of the algorithm correspond directly to the semantics given in Definition 2.2.

The claimed bound  $(|w| + |E|)^{O(|\Phi|)}$  on the running time can be verified by noting that  $\text{dom}(T)$  is of size  $|\text{env}(w)|^2 |E|^{\binom{|w|+1}{2}}$ , and  $|\text{env}(w)| \leq (1 + \binom{|w|+1}{2})^{|\Phi|}$ , since a function in  $\text{env}(w)$  maps each  $\phi \in \Phi$  to one of the at most  $\binom{|w|+1}{2}$  substrings, or leaves it undefined. With  $|\Phi|$  bounded, this makes  $|\text{dom}(T)|$  polynomial in  $|w|$  and  $|E|$ , therefore the algorithm can be performed in a polynomial number of steps (scanning  $\text{dom}(T)$  for a way to use one of the rules to set another cell to true, halting if a full scan results in no new true cells).  $\square$

Further, only regular languages are matched by  $\text{REt}$  over FST (as the class of regular languages is closed under FST), but the expressions are succinct.

**Lemma 3.4** For  $E \in \text{REt}$  over FST it is PSPACE-complete to decide whether  $\varepsilon \in \mathcal{L}(E)$ . In general, uniform membership testing for expressions in  $\text{REt}$  is PSPACE-complete.

*Proof.* PSPACE-hardness can be seen by a reduction from the (complement of the) PSPACE-complete problem FINITE AUTOMATON INTERSECTION EMPTINESS [6], where the instances are sets of finite automata  $\{A_1, \dots, A_n\}$  and the question is whether  $\mathcal{L}(A_1) \cap \dots \cap \mathcal{L}(A_n) = \emptyset$ . For each  $A_i$ ,  $i \in [n]$ , construct the FST  $t_i$  with  $\mathcal{L}(t_i) = \{(w, w) \mid w \in \mathcal{L}(A_i)\}$ , and let  $E =$

$D(t_1(\cdots t_n(\Sigma^*)\cdots))$  where  $D$  is the FST that takes every input to  $\varepsilon$ . Then the intersection is non-empty if and only if  $\varepsilon \in \mathcal{L}(E)$ .

The general problem can be solved in a straightforward way by constructing a product automaton which simulates all active transducers at once. Explicitly constructing such an automaton requires exponential space (as its states would be the Cartesian product of the states of all transducers and automata for the subexpressions). However, one can incrementally construct this product automaton, remembering only a single (product) state and its outgoing transitions at every step, while doing a nondeterministic search (recall that deterministic and nondeterministic PSPACE are equal) for an accepting path, to solve the problem in PSPACE.  $\square$

## 4. Limiting Capturing and Deletions

In limiting deletions and nesting of transducers, we find further use for Algorithm 1 after extending it to handle transducers. A key observation to achieve this, is to note that for any  $(f, v, F, f') \in \text{dom}(T)$  in Algorithm 1, where  $F$  contains no captures (and thus  $f = f'$ ), the evaluation of  $T(f, v, F, f)$  can be done without knowing the values of  $T(f, v, F', f)$ , for proper subexpressions  $F'$  of  $F$ , by constructing an NFA that is language equivalent to  $F$  (after replacing backreferences  $\phi$  in  $F$  by  $f(\phi)$ ). We can thus evaluate *simple* table cells – cells corresponding to (maximal) subexpressions without captures – in a separate step before we enter the loop in Algorithm 1.

In order to do this efficiently, the nesting of transducer applications must be bounded. We first formalize the notion of nesting depth.

**Definition 4.1** *The nesting depth  $nd(E)$  of  $E \in \text{REbt}$  is defined inductively as  $nd(E) = 0$  if  $E \in \text{REb}$ , and  $nd(E) = \max\{1 + nd(F) \mid t \in \text{td}(E), t(F) \in \text{subexps}(E)\}$  otherwise.*

---

**Algorithm 2** Membership decision procedure for the subclass of  $\text{REbt}_{\Sigma, \Phi}$  in Theorem 4.2

---

**procedure** MEMBERSHIP( $w \in \Sigma^*$ ,  $E \in \text{REbt}_{\Sigma, \Phi}$ )

▷ Returns true if  $w \in \mathcal{L}(E)$

▷ Modify  $\text{dom}(T)$  in Algorithm 1 as follows

Let  $\text{subexps}'(E) \subseteq \text{subexps}(E)$ , where  $\text{subexps}'(E)$  includes only (a) subexpressions that contain captures and (b) their immediate subexpressions.

Let  $T : \text{env}(w) \times \text{substr}(w) \times \text{subexps}'(E) \times \text{env}(w) \rightarrow \{\text{true}, \text{false}\}$ .

▷ Evaluate simple cells first, which here includes *all* transducer applications.

Evaluate  $T(f, v, F, f')$  for all  $(f, v, F, f')$  such that  $F$  contains no captures.

...

**repeat**

...

**until** no additional function values of  $T$  were set to true

**return** true if  $T(\perp, w, E, f)$  equals true for some  $f$

---

**Theorem 4.2** *For all  $E \in \text{REbt}_{\Sigma, \Phi}$  over FST such that no  $t(F) \in \text{subexprs}(E)$  with  $t \in \text{td}(E)$  contains a capture, the non-uniform membership problem can be solved in polynomial time using Algorithm 2, whereas the uniform membership problem is NP-complete for  $\text{nd}(E)$  bounded and PSPACE-complete in general.*

*Proof.* The requirement on subexpressions in  $\text{td}(E)$  ensures that all  $(f, v, F, f') \in \text{dom}(T)$ , in Algorithm 2, are such that  $v$  is a substring of  $w$  (as in Algorithm 1), and thus our environments can stay functions from  $\Phi$  to  $\text{substr}(w)$ , instead of being functions from  $\Phi$  to  $\Sigma^*$ . The restriction on  $\text{nd}(E)$  is required in order to evaluate cells of the form  $(f, v, E', f')$ , where  $E'$  contains no captures, in polynomial time (in  $|E|$ , where the degree of the polynomial is in  $\mathcal{O}(\text{nd}(E))$ ). In the uniform case membership in REb is NP-complete as it is NP-hard [1] and in NP. For containment in NP, observe that accepting  $w$  requires to guess a particular match. Hence, only a polynomial (in  $|E|$  and  $|w|$ ) number of cells in  $T$  need to be set, bounded by the number of subexpressions times the number of substrings of  $w$ . These cells can be nondeterministically chosen, verifying the match by applying Algorithm 2 to those cells only. Thus uniform membership is in PSPACE, and thus PSPACE-complete by Lemma 3.4.  $\square$

Next we restrict the type of transducers allowed in expressions.

**Definition 4.3** *Let  $n\text{-REbt}$  denote the set of all  $E \in \text{REbt}$  such that all  $t \in \text{td}(E)$  are non-deleting.*

In the uniform case the complexity of the membership problem for  $n\text{-REbt}$  remains quite high, but it sheds some light on how one may further rein the complexity in.

**Lemma 4.4** *Uniform membership for  $E \in n\text{-REbt}$  over FST is EXPSPACE-hard in general and PSPACE-hard for all fixed  $\text{nd}(E) \geq 2$ .*

*Proof.* For any fixed Turing machine  $T$  running in space  $f(|w|)$ , construct  $t_{\text{init}}$ ,  $t_{\text{step}}$  and  $t_{\text{acc}}$  as in the proof of Theorem 3.1. Given an input string  $w$  we argue how to construct an expression  $E \in n\text{-REbt}$  such that  $\varepsilon \in \mathcal{L}(E)$  if and only if  $T$  accepts  $w$ . For a polynomial and exponential  $f$  this characterizes PSPACE and EXPSPACE respectively.

For  $k \in \mathbb{N}$ , let  $D_k \in \text{FST}$  be a non-deleting transducer such that  $\mathcal{L}(D_k) = \{(u, a^{\lfloor u/k \rfloor}) \mid u \in \Sigma^*\}$ , where  $a$  is an arbitrarily chosen symbol in  $\Sigma$  and  $\lfloor \cdot \rfloor$  denotes integer division. Note that  $D_k$  can be constructed using  $k + 1$  states.

If  $f$  is a polynomial, let  $k = f(|w|)$  and let

$$E = D_k([\phi t_{\text{init}}(w)]_\phi)(D_k([\phi t_{\text{step}}(\uparrow_\phi)]_\phi))^* D_k(t_{\text{acc}}(\uparrow_\phi)).$$

Then  $\varepsilon \in \mathcal{L}(E)$  if and only if  $T$  accepts  $w$  using at most  $f(|w|)$  tape cells, the simulation working the same way as in Theorem 3.1, with  $D_k$  deleting all remnants of the computation. Clearly,  $E$  can be constructed in polynomial time and has nesting depth 2.

If  $f$  is the exponential  $c^n$  construct  $E$  as above, but replace each subexpression  $D_k(E')$  with the subexpression  $D_c^{|w|}(E')$ , i.e., using  $|w|$  nested applications of  $D_c$  to reduce up to  $c^{|w|}$  symbols to  $\varepsilon$ . This makes  $\text{nd}(E) = |w| + 1$ , but the reduction remains polynomial.  $\square$



**Lemma 4.5** *The uniform membership problem for  $E \in n\text{-REbt}$  over any class  $\Theta$  of transducers can be decided in deterministic space  $\mathcal{O}(|E|^2 c^{2nd(E)} (|w| + c)^2)$ , where  $c$  is the maximum of 2 and the largest non-deletion constant in  $td(E)$ , provided that the membership problem  $(u, v) \in t$ , for transducers  $t \in \Theta$ , can be decided in nondeterministic space  $\mathcal{O}(|t|(|u| + |v|))$ .*

*Proof (sketch).* For any string  $w$  we can check whether  $w \in \mathcal{L}(E)$  in the following way. Let  $\{t_1, \dots, t_k\} = td(E)$ , and let  $c$  be as in the statement of the lemma, i.e.  $c \geq 2$  and  $(u, v) \in \mathcal{L}(t_i)$  implies  $|u| \leq c|v| + c$  for  $1 \leq i \leq k$ . Let  $L : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  be the function defined as  $L(n) = c^{nd(E)}n + (c^{nd(E)} + c^{nd(E)-1} + \dots + c)$ . Note that  $L(n) \leq c^{nd(E)}(n + c)$  and let  $m(n) = c^{nd(E)}(n + c)$ .

For  $E$  to match  $w$  it must do so without any subexpression matching a string longer than  $L(|w|)$ , and  $L(|w|) \leq m(|w|)$ . This follows by observing that every subexpression is, by definition, surrounded by at most  $nd(E)$  transducers, and by applying the inequality  $|u| \leq c|v| + c$ , recursively,  $nd(E)$  times. A string longer than  $L(|w|)$  being matched by a subexpression thus results in the overall string matched being longer than  $w$ .

We can now determine whether  $w \in \mathcal{L}(E)$  by performing a nondeterministic search across the expression, remembering only a single search state  $(p, f, e)$  consisting of three things:

1. The current position  $p$  reached in  $E$  (viewing  $E$  as a string).
2. Whenever entering a subexpression a string of length at most  $m(|w|)$  is nondeterministically guessed and recorded in a table  $f$  mapping subexpressions to strings, inserting a marker  $\triangleright$  in each  $f(E')$  to record the position up to which the string has been matched so far.
3. The current environment  $e \in \text{env}_{\Phi, \Sigma}$ , recording the strings so far captured (if any) and these too never need more space than  $|\Phi| \cdot (m(|w|) + 1)$ .

Informally, we start in state  $(p_0, f, \perp)$  where  $p_0$  is the leftmost position in  $E$ , setting  $f$  to map  $E$  to  $\triangleright w$ . Then we nondeterministically walk the expression, guessing a new string to enter into  $f$  whenever we enter a subexpression, verifying the guess, updating the parent marker (and the current environment if a capture), and simulating transducers as needed, whenever we exit a subexpression. If the rightmost position can be reached with  $f(E) = w \triangleright$ , then  $w$  is matched.

A state uses space  $|E| + |E| \cdot m(|w|) + |\Phi| \cdot (m(|w|) + 1)$ , so the procedure runs in nondeterministic space  $\mathcal{O}(|E|m(|w| + c))$ . Note for all transducer evaluations  $(u, v) \in t$  that need to be decided to determine if  $w \in \mathcal{L}(E)$ , we have  $|u| \leq m(|w| + c)$ ,  $|v| \leq m(|w| + c)$ , and also  $|t| \leq |E|$ . Thus, applying Savitch's theorem [8], we obtain a deterministic procedure for which  $\mathcal{O}(|E|^2 c^{2nd(E)} (|w| + c)^2)$  space is sufficient.  $\square$

**Theorem 4.6** *Uniform membership for  $E \in n\text{-REbt}$  over FST is EXPSPACE-complete in general and PSPACE-complete for all fixed  $nd(E) \geq 2$ .*

*Proof.* Combine Lemma 4.4 and 4.5.  $\square$

The non-uniform variant of the problem is not surprisingly a bit less complex, but remains NP-complete.

**Lemma 4.7** *The non-uniform membership problem for  $n$ -REbt over  $\Theta$  is in NP, provided that deciding membership is in NP for every  $\theta \in \Theta$ .*

*Proof.* For any (fixed)  $E \in n$ -REbt, if  $w$  is an input string, then we can check whether  $w \in \mathcal{L}(E)$  in nondeterministic polynomial time by the following procedure. Let  $c$  be the largest non-deletion constant in  $\text{td}(E)$  and  $m = c^{nd(E)}$ . By the same argument as in Lemma 4.5 no subexpression (more precisely: none of the matching relations inductively used in matching, as defined by Definition 2.2) in  $E$  matches a string longer than  $m(|w| + c)$  when  $E$  matches  $w$ . Additionally, fewer than  $(|w| + c) \cdot |\text{subexps}(E)|$  instances of a subexpression matching a string longer than  $m$  can occur, as such strings contribute to the length of the overall string matched (the  $|\text{subexps}(E)|$  accounts for nested subexpressions involved in the match of part of a substring, or transducer preimage, matched by a larger subexpression).

Start by nondeterministically choosing a string  $v$  of length  $m(|w| + c)^2 \cdot |\text{subexps}(E)|$ , and construct  $v'$  to be a string containing as a substring *every* string of length at most  $m$  (the length of  $v'$  is thus exponential in  $m$ , but is fixed as it depends only on  $E$ ), and let  $w' = vv'$ . Then modify Algorithm 1 by adding the following step to the repeat-until loop:

**if**  $T(f, u, F, f') = \text{true}$  and  $(u, v) \in \mathcal{L}(t)$  **then**  
      $T(f, v, t(F), f') \leftarrow \text{true}$

The resulting algorithm, applied to the input string  $|w'|$ , sets  $T(\perp, w, E, f)$  to true for some  $f$  if and only if  $E$  matches  $w$ . This procedure runs in nondeterministic polynomial time as  $w'$  is polynomial in length when  $E$  is taken to be fixed (as  $m$  is then constant). This works because any subexpression matching a string of length at most  $m$  can find that string in the  $v'$  section of  $w'$ , and the at most  $(|w| + c) \cdot |\text{subexps}(E)|$  subexpressions matching strings of length at most  $m(|w| + c)$  will have their strings nondeterministically generated in the  $v$  section of  $w'$ .  $\square$

**Lemma 4.8** *The non-uniform membership problem for  $n$ -REbt $_{\Sigma, \Phi}$  over  $\Theta$  is NP-hard if  $\Theta$  contains all single state FST.*

*Proof.* We demonstrate NP-hardness by a reduction from the NP-hard LONGEST COMMON SUBSEQUENCE problem [7], the instances of which are the tuples  $(\{w_1, \dots, w_m\}, n)$ ,  $\{w_1, \dots, w_m\} \subseteq \mathcal{L}((a|b)^*)$ ,  $n \in \mathbb{N}$ , such that there exists a string  $v$  of length  $n$  which forms a subsequence of  $w_i$  for all  $i$ . Take  $\Sigma = \{a, b, \#, x\}$  and  $\Phi = \{\text{guess}\}$ , let  $t$  be the transducer  $a : x, b : x$  and  $s$  the transducer  $a : a, b : b, \varepsilon : a, \varepsilon : b$ , then the expression  $E_{\text{lcs}} = t([\text{guess}(a|b)^*]_{\text{guess}})(\#(s(\uparrow_{\text{guess}})))^*$  matches the string  $x^n \# w_1 \# \dots \# w_m$  if and only if  $w_1, \dots, w_m$  are strings in  $\{a, b\}^*$  with a common subsequence of length  $n$ . To see this, note that the initial  $x^n$  means that a string  $w \in \{a, b\}^n$  must be captured by the capturing group ‘guess’. Thus, each  $w_i$  must be matched by inserting  $as$  and  $bs$  into  $w$ , making  $w$  a common subsequence of each of  $w_1, \dots, w_m$ . As such any instance of LONGEST COMMON SUBSEQUENCE can be decided by checking whether  $x^n \# w_1 \# \dots \# w_m \in \mathcal{L}(E_{\text{lcs}})$ .  $\square$

This expression  $E_{\text{lcs}}$ , used in the previous proof, will be reused near-verbatim to demonstrate NP-hardness of membership in fln- and nt-REbt (to be defined in the next section).

**Theorem 4.9** *The non-uniform membership problem for  $n\text{-REbt}_{\Sigma, \Phi}$  over  $\Theta$  is NP-complete for every  $\Theta$  containing all single state FST, provided that the membership problem of each transducer in  $\Theta$  is in NP.*

*Proof.* Combine Lemma 4.7 and 4.8. □

## 5. Limiting Nesting and Non-Functional Behavior

In this section we consider restrictions to the way in which REbt may nest the application of transducers, and in the process also consider the restriction to functional transducers, the combination of which gives rise to a convenient normal form. The choice of restrictions is driven by the intuition that the construction in Theorem 3.1 relies on the subexpression  $[\phi t_{\text{step}}(\uparrow\phi)]_{\phi}^*$  to match complex languages by the iterated application of  $t_{\text{step}}$ . By syntactically avoiding the iterated application of a transducer to previous output produced by the same transducer, we obtain a class of languages with much better properties.

**Definition 5.1** *For  $E \in \text{REbt}_{\Sigma, \Phi}$  (over an arbitrary class of transducers) let  $\Sigma_{td(E)} = \Sigma \uplus \{\langle t, \cdot \rangle_t \mid t \in td(E)\}$  and define  $\tau(E) \in \text{REb}_{\Sigma_{td(E)}, \Phi}$  as the expression obtained by replacing every subexpression of the form  $t(F)$ , where  $t \in td(E)$ , with  $\langle t \cdot F \cdot \rangle_t$ . Let  $\tau^{-1}$  be the inverse transformation, so  $\tau^{-1}(\tau(E)) = E$  for all  $E$ .*

Note that each string in  $\mathcal{L}(\tau(E))$  is a valid expression in  $\text{RE}_{\Sigma_{td(E)}}$ . Thus,  $\tau^{-1}\mathcal{L}(\tau(E))$  denotes the set of expressions in  $\text{REt}_{\Sigma}$  obtained by applying  $\tau^{-1}$  to each expression in  $\mathcal{L}(\tau(E))$ . Furthermore,  $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$  is the union of all languages obtained by applying  $\mathcal{L}$  to each expression in  $\tau^{-1}\mathcal{L}(\tau(E))$ .

Let us define the first restriction, which forbids the capture of the output of non-functional transducers.

**Definition 5.2** *An expression  $E \in \text{REbt}$  is functional if every transducer that occurs in a capturing subexpression is functional (i.e.,  $t \in td(F)$  for some  $[\phi F]_{\phi} \in \text{subexprs}(E)$  only if  $t$  is functional). We denote the subset of REbt containing precisely the functional expressions as  $f\text{-REbt}$ .*

**Lemma 5.3** *In general,  $\mathcal{L}(E) \subseteq \mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$ , but  $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E))) = \mathcal{L}(E)$  for  $E \in f\text{-REbt}$ .*

*Proof (sketch).* Strings  $w \in \mathcal{L}(\tau(E))$  contain substrings of the form  $\langle tv \rangle_t$ , precisely where a transducer  $t$  is applied to obtain strings in  $\mathcal{L}(E)$ . Applying  $\tau^{-1}$  recovers the transducers (turns the substrings  $\langle tv \rangle_t$  back into  $t(v)$ ), and then applying  $\mathcal{L}$  evaluates all transducers. In general,  $\mathcal{L}(E) \subseteq \mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$ , since in  $\mathcal{L}(E)$  subexpressions containing transducers first apply the transducer before (potentially) copying an output of the transducer, while in  $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$ , subexpressions containing transducers may get copied before applying the transducers. Hence, nondeterministic transducers may turn the copies into different output strings. However, the restriction to functional transducers prevents this effect, ensuring equality for  $E \in f\text{-REbt}$ . □

**Example 5.4** In this example we show that the equality  $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E))) = \mathcal{L}(E)$  of Lemma 5.3 does not hold for REbt in general. Let  $E = [{}_1f(a^*)]_1\uparrow_1$ , with  $f$  given by  $a:a, a:b$ . Then  $\mathcal{L}(\tau(E)) = \{\langle_f a^n \rangle_f \langle_f a^n \rangle_f \mid n \in \mathbb{N}_0\}$ , thus  $\mathcal{L}(\tau^{-1}(\langle_f a^n \rangle_f \langle_f a^n \rangle_f)) = \mathcal{L}(f(a^n)f(a^n)) = (a|b)^{2n}$ , whereas  $\mathcal{L}(E) = \{ww \mid w \in \mathcal{L}((a|b)^*)\}$ . That is, in  $\mathcal{L}(E)$  the transducer is applied before it gets copied by the capturing group and backreference, whereas  $\tau$  “hides” the transducer as a string, letting it be copied before it is applied, after which  $\tau^{-1}$  is applied to recover the two transducers obtained by copying, and  $\mathcal{L}$  then evaluates them independently.

Next we construct a transducer  $T$  such that under certain restrictions, expressions over FST satisfy  $\mathcal{L}(T(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$ .

**Definition 5.5** For  $c \in \mathbb{N}$  and  $E \in \text{REbt}_{\Sigma, \Phi}$  over FST, with  $\text{td}(E) = \{t_1, \dots, t_n\}$ , where  $t_i = (Q_i, \Sigma, q_{0,i}, \delta_i, F_i)$ , for  $i \in [n]$ , let  $T_{E,c} = (Q, \Sigma', q_0, \delta, F)$  be the transducer defined as follows: (i)  $Q = \{q \in (Q_1 \uplus \dots \uplus Q_n)^* \mid |q| \leq c\}$ ; (ii)  $\Sigma' = \Sigma_{\text{td}(E)} = \Sigma \uplus \{\langle_t, \rangle_t \mid t \in \text{td}(E)\}$ ; (iii)  $q_0 = \varepsilon$ ; (iv)  $F = \{\varepsilon\}$ ; and (v)  $\delta = \delta_\zeta \cup \delta_\gamma \cup \delta_\varepsilon \cup \delta_\Sigma$  where:

- $\delta_\zeta = \{(q, \langle_t, \varepsilon, q \cdot q_{0,i} \rangle \mid i \in [n], q \in Q, |q| + 1 \leq c\}$ ;
- $\delta_\gamma = \{(q_1 \dots q_{k-1} q_k, \rangle_{t_i}, \varepsilon, q_1 \dots q_{k-1}) \mid i \in [n], q_1 \dots q_k \in Q, q_k \in F_i\}$ ;
- $\delta_\varepsilon = \{(\varepsilon, \alpha, \alpha, \varepsilon) \mid \alpha \in \Sigma'\}$ ;
- $\delta_\Sigma$  is defined inductively over the length  $k$  of state sequences, as follows: for  $\alpha, \beta \in \Sigma_\varepsilon$ ,  $(q_1 \dots q_k, \alpha, \beta, q'_1 \dots q'_k) \in \delta_\Sigma$  if for some  $\alpha' \in \Sigma_\varepsilon$  and  $i \in [n]$ ,
  - $(q_k, \alpha, \alpha', q'_k) \in \delta_i$ ; and,
  - $(q_1 \dots q_{k-1}, \alpha', \beta, q'_1 \dots q'_{k-1}) \in \delta_\varepsilon \cup \delta_\Sigma$ .

Informally, without the length restriction enforced by  $c$  in  $T_{E,c}$ , i.e., letting  $c = \infty$ , we have  $\mathcal{L}(T_{E,\infty}(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$ . Next we define a restriction to ensure that a (finite) value can be selected for  $c$  such  $\mathcal{L}(T_{E,c}(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$ .

**Definition 5.6** An expression  $E \in \text{REbt}$  such that every  $t \in \text{td}(E)$  occurs only once in  $E$ , is loop-free if  $\mathcal{L}(\tau(E))$  contains no subexpression of the form  $\langle_t \dots \langle_t \dots \rangle_t \dots \rangle_t$ , i.e., there are no nested subexpressions  $\langle_t \dots \rangle_t$  for any  $t \in \text{td}(E)$ . We denote the set of all loop-free REbt by  $l\text{-REbt}$ .

**Lemma 5.7** For  $E \in l\text{-REbt}_{\Sigma, \Phi}$  over FST, we have  $\mathcal{L}(T_{E,|\text{td}(E)|}(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$ .

*Proof.* Set  $c$  to be the maximum number of nested transducers in  $\tau^{-1}(v)$  over any  $v \in \mathcal{L}(\tau(E))$ , i.e.,  $c \leq |\text{td}(E)|$  by Definition 5.6. We prove that  $T_{E,c}$  simulates all transducers running at each point of an input string. Thus  $T_{E,c}$  produces the same output strings as would be obtained by first having  $\tau^{-1}$  recover the transducers, and then evaluating them by using  $\mathcal{L}$ . This can be seen by induction on the number of transducers in  $E$ . Assume  $T_{E,c}$  can go from state  $q_1 \dots q_k$  to  $q'_1 \dots q'_k$  while reading  $v$  and producing  $w$  as output, and that  $q_i$  (and thus also  $q'_i$ ) is a state from  $t_i$ , for  $i \in [k]$ . Then there exists strings  $v_0, \dots, v_k$ , with  $v_0 = w$  and  $v_k = v$ , such that transducer  $t_i$ , for  $i \in [k]$ , can go from state  $q_i$  to  $q'_i$  when reading  $v_i$  and producing  $v_{i-1}$  as output. Add to this the brackets  $\langle_{t_i}, \rangle_{t_i}$ , instructing  $T_{E,c}$  when to start and stop and check

for acceptance of transducer  $t_i$ , and make  $T_{E,c}$  act as the identity (the rules in  $\delta_\varepsilon$ ) when no transducers are simulated, to complete the picture.

It is sufficient to pick  $c$  to be equal to the  $|\text{td}(E)|$ , since this is an upper bound for the nesting depth for pairs of transducer brackets, i.e., symbols of the form  $\langle_t, \rangle_t$ , for strings matched by  $\tau(E)$ , given that  $E \in \text{fl-REbt}_{\Sigma, \Phi}$ . The transducer  $T_{E,c}$  reaches a state sequence  $q_1 \cdots q_k$  precisely when the combined effect of  $k$  (distinct) transducers are being simulated by  $T_{E,c}$ , and  $|\text{td}(E)|$  places an upper bound on the number of transducers being applied simultaneously.  $\square$

**Corollary 5.8** *For all  $E \in \text{fl-REbt}_{\Sigma, \Phi}$  (i.e. expressions fulfilling both Definitions 5.2 and 5.6) over FST we have  $\mathcal{L}(E) = \mathcal{L}(T_{E,|\text{td}(E)|}(\tau(E)))$ , and thus every fl-REbt can be put in a normal form  $t(E')$  where  $E' \in \text{REb}$  (i.e. an expression with only a single top-level transducer).*

*Proof.* This result combines Lemma 5.3 with Lemma 5.7.  $\square$

**Definition 5.9** *Let t-REbt denote the subset of REbt which are in the normal form of Corollary 5.8.*

**Remark 5.10** *Lemma 4.8 demonstrates that non-uniform membership for fl-REbt and thus t-REbt (both over FST) is NP-hard, since  $E_{1cs}$  used in the proof of Lemma 4.8 is in fl-REbt (although the transducer  $s$  in  $E_{1cs}$  is not functional, output of  $s$  is not captured), and the above corollary can be used to convert  $E_{1cs}$  into an expression t-REbt.*

## 6. The Membership Problem for t-REbt

We show that the class t-REbt, and thus also fl-REbt, has a decidable membership problem, but it is complex, even with the input string fixed.

**Corollary 6.1 (of Lemma 3.4)** *For  $E \in \text{t-REbt}_{\Sigma, \Phi}$  over FST it is PSPACE-hard to decide whether  $\varepsilon \in \mathcal{L}(E)$ .*

*Proof.* Modify Lemma 3.4 by letting  $E = D([\Sigma^*]_1 t_1(\uparrow_1) \cdots t_n(\uparrow_1)) \in \text{fl-REbt}$ , with  $t_1, \dots, t_n$  as in the proof of Lemma 3.4.

Then  $\varepsilon \in \mathcal{L}(E)$  iff the intersection  $\mathcal{L}(A_1) \cap \dots \cap \mathcal{L}(A_n)$ , again with the  $A_i$  as in (the proof of) Lemma 3.4, is non-empty. The expression  $E$  can be converted into t-REbt normal form by Corollary 5.8. Note that  $T_{E,2}$  (i.e.  $c = 2$ ) needs to be constructed, and the resulting expression is therefore polynomial in the size of  $E$ . Note that the proof of Lemma 5.7 in fact shows that  $c$  can be chosen as the maximum number of nested transducers in  $\tau^{-1}(v)$  over any  $v \in \mathcal{L}(\tau(E))$ , which in this case is  $c = 2$ , instead of  $|\text{td}(E)| = n + 1$ .  $\square$

Next we show that the membership problem for t-REbt (and by extension fl-REbt) can be decided in polynomial space. The approach works by, for a given input string and  $t(F) \in \text{t-REbt}$ , computing the preimage of  $t$  on  $w$ , and intersecting this regular language with  $\mathcal{L}(F)$ . To achieve this within polynomial space, however, it is necessary to not expand captures and backreferences.

**Definition 6.2** For  $E \in REbt_{\Sigma, \Phi}$  we define  $\sigma(E) \in REt_{\Sigma, \emptyset}$  with  $\Sigma_{\Phi} = \Sigma \uplus \{\llbracket \cdot \rrbracket_{\phi}^i, \lrcorner_{\phi} \mid \phi \in \Phi, i \in [n]\}$ , where  $n$  is equal to the maximum number of capturing expressions on the same capturing symbol, by making the following substitutions in  $E$ .

1. Replace every subexpression of the form  $\lrcorner_{\phi}$ , with  $\lrcorner_{\phi}$ .
2. Replace the  $i$ th subexpression (ordered for example from left to right based on the position of the opening capturing bracket) of the form  $\llbracket \cdot \rrbracket_{\phi}$ , by  $\llbracket \cdot \rrbracket_{\phi}^i$ .

Let  $\sigma^{-1}$  be the inverse transformation of  $\sigma$ , i.e.  $\sigma^{-1}(\sigma(E)) = E$ , and extend  $\sigma^{-1}$  to sets of expressions in the obvious way.

For a finite automaton  $A$ , over alphabet  $\Sigma$ , and  $n \in \mathbb{N}$ , we define an automaton  $C_{A, \Phi, n}$ , which will be used to determine if  $\mathcal{L}(A) \cap \mathcal{L}(E)$ , for  $E \in REb$ , is non-empty. To simplify our constructions, and since it will not make our results less general, we assume that  $A$  has no  $\varepsilon$ -transitions. Recall that we use  $\perp$  to denote the partial function with empty domain.

**Definition 6.3** For  $\Sigma_{\Phi}$  and  $n$  as in Definition 6.2, and an automaton  $A = (Q, \Sigma, q_0, \delta, F)$ , let  $C_{A, \Phi, n} = (Q', \Sigma_{\Phi}, q'_0, \delta', F')$  be the automaton where  $Q' = Q \times (\Phi \rightarrow 2^{Q \times Q}) \times ((\Phi \times [n]) \rightarrow 2^{Q \times Q})$ ,  $q_0 = (q_0, \perp, \perp)$ ,  $F' = \{(q, C, M) \in Q \mid q \in F\}$ , and  $((q, C, M), \alpha, (q', C', M')) \in \delta'$  if one of the following holds:

1.  $(q, \alpha, q') \in \delta$ ,  $C' = C$ , and  $M' = \{(\phi, i, (p, p'')) \mid (\phi, i, (p, p')) \in M, (p', \alpha, p'') \in \delta\}$ ,
2.  $\alpha = \llbracket \cdot \rrbracket_{\phi}^i$ ,  $q = q'$ ,  $C' = C$ , and  $M' = M[(\phi, i) \mapsto \{(p, p) \mid p \in Q\}]$ ,
3.  $\alpha = \lrcorner_{\phi}^i$ ,  $q = q'$ ,  $M' = M[(\phi, i) \mapsto \perp]$ , and  $C' = C[\phi \mapsto M(\phi, i)]$  or
4.  $\alpha = \lrcorner_{\phi}$ ,  $(q, q') \in C(\phi)$ ,  $C' = C$ , and, for all  $\phi \in \Phi$  and  $i \in [n]$  we have  $M'(\phi, i) = \{(p, p'') \mid (p, p') \in M(\phi, i), (p', p'') \in C(\phi)\}$ .

In the next result we extend  $\mathcal{L}$  to be also applied to a set of expressions, and to denote the union of languages defined by the expressions. As the full proof is quite technical, we provide only a sketch that should be sufficient to convey the idea.

**Lemma 6.4** Let  $A$  be an automaton,  $E \in REb$ , and  $n \in \mathbb{N}$ , with  $n$  equal to the the maximum number of capturing expressions on the same capturing symbol, in  $E$ . Then we have  $\mathcal{L}(\sigma^{-1}(\mathcal{L}(C_{A, \Phi, n}) \cap \mathcal{L}(\sigma(E)))) = \mathcal{L}(A) \cap \mathcal{L}(E)$ .

*Proof (sketch).* For  $w \in \mathcal{L}(\sigma(E))$ , note that  $\mathcal{L}(\sigma^{-1}(w))$  is a single string in  $\mathcal{L}(E)$  (as  $\sigma^{-1}$  recovers the captures and backreferences, and  $\mathcal{L}$  then evaluates them).  $C_{A, \Phi, n}$ , running on  $w$  simulates  $A$  running on  $\mathcal{L}(\sigma^{-1}(w))$ . This can be demonstrated by induction on the transitions  $C_{A, \Phi, n}$  takes on a string  $w$ . Specifically, if  $C_{A, \Phi, n}$  reaches  $(q, C, M)$  on a prefix  $v$  of  $w$  then  $A$  reaches  $q$  on  $\mathcal{L}(\sigma^{-1}(v))$  (to aid intuition we extend  $\sigma^{-1}$  to remove unmatched brackets, making it defined for all  $v$ ). Start by noting that  $C_{A, \Phi, n}$  starts in state  $(q_0, \perp, \perp)$  and  $A$  starts in state  $q_0$ . The inductive step then follows from the four different types of transitions in Definition 6.3:

1. This simulates a step where  $A$  goes from  $q$  to  $q'$  reading  $\alpha \in \Sigma$ , updating both the current state and recording the effect on each state relation in the table  $M$ , inductively recording the behavior of  $A$  on this substring if it is later repeated by a backreference.

2. Here,  $A$  takes no step, as the bracket is removed by  $\sigma^{-1}$ , but the indicated capture is initialized to  $\{(p, p) \mid p \in Q\}$  in the table  $M$ , representing the behavior of  $A$ , starting in any state, on the string  $\varepsilon$  (the string captured so far).
3. In this case, no step in  $A$  is taken (as the bracket is removed by  $\sigma^{-1}$ ), but the corresponding capture in the table  $M$  is transferred to the table  $C$  (completing the capture).
4. On the backreference  $\uparrow_\phi$  the relation recorded in  $C(\phi)$  is retrieved and used to update the current state, simulating any sequence of transitions  $A$  can take from the current state on the string currently captured with backreference symbol  $\phi$ .  $\square$

**Theorem 6.5** *The emptiness of  $\mathcal{L}(E) \cap \mathcal{L}(A)$ , for  $E \in \text{REb}$  and  $A$  a finite automaton, can be decided in PSPACE.*

*Proof.* As  $\mathcal{L}(\sigma(E))$  and  $\mathcal{L}(C_{A,\Phi,n})$  are regular languages, a standard product automaton can be constructed for  $\mathcal{L}(\sigma(E)) \cap \mathcal{L}(C_{A,\Phi,n})$ . While  $C_{A,\Phi,n}$  is potentially large, emptiness can be decided in polynomial space by performing a nondeterministic search (as nondeterministic polynomial space equals polynomial space) for an accepting computation by incrementally constructing each, polynomially sized, state as it is visited, forgetting it again in the next step.  $\square$

**Theorem 6.6** *The uniform membership problems for t-REbt and fl-REbt (both over FST) are PSPACE-complete.*

*Proof.* For t-REbt (and thus for fl-REbt) hardness is established in Corollary 6.1. To see that uniform membership for t-REbt is in PSPACE, consider an expression  $E = t(F)$ , for  $t \in \text{FST}$  and  $F \in \text{REb}$ . To check whether  $w \in \mathcal{L}(E)$ , construct a finite automaton  $A$  with  $\mathcal{L}(A) = \{v \mid (w, v) \in \mathcal{L}(t)\}$  (this can be done with standard techniques, producing an automaton  $A$  polynomial in size in  $|t| \cdot |w|$ ). Then if  $\mathcal{L}(A) \cap \mathcal{L}(F) \neq \emptyset$ , we have  $w \in \mathcal{L}(E)$ , which we can check in polynomial space by Theorem 6.5. This procedure extends to fl-REbt by additionally constructing  $T_{E,c}$ , as in Corollary 5.8, in an incremental fashion.  $\square$

A uniform membership problem in PSPACE improves vastly on the unrestricted case, and the top-level transducer appears to be a very natural formalism. More importantly, fairly minor further restrictions recover the easier membership problems established for REb.

**Theorem 6.7** *The uniform and non-uniform membership problem for nt-REbt (where nt-REbt denotes  $n\text{-REbt} \cap t\text{-REbt}$ ) is NP-complete.*

*Proof.* Take  $E = t(F) \in \text{nt-REbt}$  and let  $w$  be the input string. Since  $t$  is nondeleting,  $|t| \cdot |w| \geq \max\{|v| \mid (w, v) \in \mathcal{L}(t)\}$ , so we can nondeterministically choose a  $v$  with  $(w, v) \in \mathcal{L}(t)$  and apply Lemma 3.3 to check in nondeterministic polynomial time if  $v \in \mathcal{L}(F)$ . NP-hardness in the non-uniform case is established by Lemma 4.8 (see Remark 5.10).  $\square$

## 7. Summary and Future Work

**Summary.** We have (i) proposed an extension of regular expressions with backreferences by additional transducers; (ii) established that this makes membership testing intractable; and

(iii) explored various restrictions to form a practical basis for use in software. By Example 2.3 all restrictions can match  $L_{RD}$ ,  $L_{MA}$ , and  $L_{CA}$ , but offer different levels of membership testing complexity and expressiveness. For immediate integration in an existing backtracking matching engine the restriction in Theorem 4.2 appears to be the obvious choice, with no transducer preimage ever captured, the matching procedure requiring only minor additional work compared to matching plain REb. Further, the relative tractability of the nondeleting class demonstrates that one source of intractability is the ability gained by an unrestricted use of transducers to erase every trace of an arbitrarily complex computation that has been made. However, the reduction of fl-REbt to t-REbt shows that the latter, despite being very simple, can capture many natural situations. Small additional restrictions can then be applied to obtain highly tractable subclasses.

**Future work.** The precise expressiveness of the classes should be considered, several gaps exist beyond what follows naturally from what we have done here; f-REbt  $\equiv$  REbt; fl-REbt  $\equiv$  t-REbt; n-, fl-/t-REbt all being strict subclasses of f-REbt/REbt and strict superclasses of REb. The subclasses should also be compared with respect to succinctness, and there remain some open questions regarding computational complexity (e.g. non-uniform membership for t-REbt).

## References

- [1] A. AHO, Algorithms for Finding Patterns in Strings. In: J. VAN LEEUWEN (ed.), *Handbook of Theoretical Computer Science (Vol. A)*. MIT Press Cambridge, MA, USA, 1990, 255–300.
- [2] M. BERGLUND, B. VAN DER MERWE, *Re-examining regular expressions with backreferences*, 2018. Preprint.
- [3] C. CÂMPEANU, K. SALOMAA, S. YU, A formal study of practical regular expressions. *International Journal of Foundations of Computer Science* 14 (2003) 6, 1007–1018.
- [4] J. DASSOW, R. PÄUN, A. SALOMAA, Grammars with Controlled Derivations. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages: Volume 2. Linear Modeling Background and Application*. Springer, 1997, 101–154.
- [5] D. FREYDENBERGER, M. SCHMID, Deterministic regular expressions with back-references. In: H. VOLLMER, B. VALLÉE (eds.), *34th Symposium on Theoretical Aspects of Computer Science, STACS*. Leibniz International Proceedings in Informatics (LIPIcs) 66, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 1–14.
- [6] D. KOZEN, Lower bounds for natural proof systems. In: *18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1977, 254–266.
- [7] D. MAIER, The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)* 25 (1978) 2, 322–336.
- [8] W. SAVITCH, Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* 4 (1970) 2, 177 – 192.
- [9] M. SCHMID, Characterising REGEX languages by regular languages equipped with factor-referencing. *Information and Computation* 249 (2016), 1–17.



# POSTSELECTING PROBABILISTIC FINITE STATE RECOGNIZERS AND VERIFIERS

Maksims Dimitrijevs<sup>(A,B)</sup>    Abuzer Yakaryılmaz<sup>(A,B)</sup>

<sup>(A)</sup>University of Latvia, Faculty of Computing  
Raiņa bulvāris 19, Rīga, LV-1586, Latvia

<sup>(B)</sup>University of Latvia, Center for Quantum Computer Science  
Raiņa bulvāris 19, Rīga, LV-1586, Latvia

md09032@lu.lv

abuzer@lu.lv

## **Abstract**

*In this paper, we investigate the computational and verification power of bounded-error postselecting realtime probabilistic finite state automata (PostPFAs). We show that PostPFAs using rational-valued transitions can do different variants of equality checks and they can verify some nonregular unary languages. Then, we allow them to use real-valued transitions (magic-coins) and show that they can recognize uncountably many binary languages by help of a counter and verify uncountably many unary languages by the help of a prover. We also present some corollaries on probabilistic counter automata.*

**Keywords:** *Postselection, probabilistic automata, interactive proof systems, unary languages, counter automata.*

## **1. Introduction**

Postselection is the ability to give a decision by assuming that the computation is terminated with pre-determined outcome(s) and discarding the rest of the outcomes. In [1], Aaronson introduced bounded-error postselecting quantum polynomial time and proved that it is identical to the unbounded-error probabilistic polynomial time. Later, postselecting quantum and probabilistic finite automata models have been investigated in [15, 16, 18, 19]. It was proved that postselecting realtime finite automata are equivalent to a restricted variant of two-way finite automata, called restarting realtime automata [17]. Later, it was also shown that these two automata models are also equivalent to the realtime automata that have the ability to send a classical bit through CTCs (closed timelike curves) [12, 13].

In this paper, we focus on bounded-error postselecting realtime probabilistic finite automata (PostPFAs) and present many algorithms and protocols by using rational-valued and real-valued transitions. Even though PostPFA is a restricted variant of two-way probabilistic finite

automaton (2PFA), our results may be seen as new evidences that PostPFAs can be as powerful as 2PFAs.

We show that PostPFAs with rational-valued transitions can recognize different variants of “equality” language  $\{a^n b^n \mid n > 0\}$ . Then, based on these results, we show that they can verify certain unary nonregular languages. Remark that bounded-error 2PFAs cannot recognize unary nonregular languages [10].

When using real-valued transitions (so-called magic coins), probabilistic and quantum models can recognize uncountably many languages by using significantly small space and in polynomial time in some cases [14, 5, 6, 8]. In the same direction, we examine PostPFAs using real-valued transitions and show that they can recognize uncountably many binary languages by using an extra counter. When interacting with a prover, we obtain a stronger result that PostPFAs can recognize uncountably many unary languages. We also present some corollaries for probabilistic counter automata.

In the next section, we provide the notations and definitions used in the paper. Then, we present our results on PostPFAs using rational-valued transitions in Section 3 and on PostPFAs using real-valued transitions in Section 4. In each section, we also separate recognition and verification results under two subsections.

As a related work, we recently present similar verification results for 2PFAs that run in polynomial expected time in [8]. Even though here we get stronger results for some cases (i.e., PostPFA is a restricted version of 2PFA), if we physically implement PostPFA algorithms and protocols presented in this paper, the expected running time will be exponential.

## 2. Background

We assume that the reader is familiar with the basics of fundamental computational models and automata theory.

For any alphabet  $A$ ,  $A^*$  is the set of all finite strings defined on alphabet  $A$  including the empty string and  $A^\infty$  is set of all infinite strings defined on alphabet  $A$ . We fix symbols  $\clubsuit$  and  $\$$  as the left and the right end-marker. The input alphabet not containing  $\clubsuit$  and  $\$$  is denoted  $\Sigma$  and the set  $\tilde{\Sigma}$  is  $\Sigma \cup \{\clubsuit, \$\}$ . For any given string  $w \in \Sigma^*$ ,  $|w|$  is its length,  $w[i]$  is its  $i$ -th symbol ( $1 \leq i \leq |w|$ ), and  $\tilde{w} = \clubsuit w \$$ . For any natural number  $i$ ,  $binary(i)$  denotes unique binary representation.

Our realtime models operate in strict mode: any given input, say  $w \in \Sigma^*$ , is read as  $\tilde{w}$  from the left to the right and symbol by symbol without any pause on any symbol.

Formally, a postselecting realtime probabilistic finite state automaton (PostPFA)  $P$  is a 6-tuple

$$P = (\Sigma, S, \delta, s_I, s_{pa}, s_{pr}),$$

where

- $S$  is the set of states,
- $\delta : S \times \tilde{\Sigma} \times S \rightarrow [0, 1]$  is the transition function described below,
- $s_I \in S$  is the starting state, and,
- $s_{pa} \in S$  and  $s_{pr} \in S$  are the postselecting accepting and rejecting states ( $s_{pa} \neq s_{pr}$ ), respectively.

We call any state other than  $s_{pa}$  or  $s_{pr}$  non-postselecting.

When  $P$  is in state  $s \in S$  and reads symbol  $\sigma \in \tilde{\Sigma}$ , then it switches to state  $s' \in S$  with probability  $\delta(s, \sigma, s')$ . To be a well-formed machine, the transition function must satisfy that

$$\text{for any } (s, \sigma) \in S \times \tilde{\Sigma}, \quad \sum_{s' \in S} \delta(s, \sigma, s') = 1.$$

Let  $w \in \Sigma^*$  be the given input. The automaton  $P$  starts its computation when in state  $s_I$ . Then, it reads the input and behaves with respect to the transition function. After reading the whole input,  $P$  is in a probability distribution, which can be represented as a stochastic vector, say  $v_f$ . Each entry of  $v_f$  represents the probability of being in the corresponding state.

Due to postselection, we assume that the computation ends either in  $s_{pa}$  or  $s_{pr}$ . We denote the probabilities of being in  $s_{pa}$  and  $s_{pr}$  as  $a(w)$  and  $r(w)$ , respectively. It must be guaranteed that  $a(w) + r(w) > 0$ . (Otherwise, postselection cannot be done.) Then, the decision is given by normalizing these two values:  $w$  is accepted and rejected with probabilities

$$\frac{a(w)}{a(w) + r(w)} \text{ and } \frac{r(w)}{a(w) + r(w)},$$

respectively. We also note that the automaton  $P$  ends its computation in non-postselecting state(s) (if there is any) with probability  $1 - a(w) - r(w)$ , but the ability of making postselection discards this probability (if it is non-zero).

By making a simple modification on a PostPFA, we can obtain a restarting realtime PFA (restartPFA) [17]:

- each non-postselecting state is called restarting state,
- postselecting accepting and rejecting states are called accepting and rejecting states, and then,
- if the automaton ends in a restarting state, the whole computation is started again from the initial configuration (state).

The analysis of accepting and rejecting probabilities for the input remains the same and so both models have the same accepting (and rejecting) probabilities on every input.

Moreover, if we have  $a(w) + r(w) = 1$  for any input  $w \in \Sigma^*$ , then the automaton is simply a probabilistic finite automaton (PFA) since making postselection or restarting mechanism does not have any effect on the computation or decision.

Language  $L \subseteq \Sigma^*$  is said to be recognized by a PostPFA  $P$  with error bound  $\epsilon$  if

- any member is accepted by  $P$  with probability at least  $1 - \epsilon$ , and,
- any non-member is rejected by  $P$  with probability at least  $1 - \epsilon$ .

We can also say that  $L$  is recognized by  $P$  with bounded error or recognized by bounded-error PostPFA  $P$ .

In this paper, we also focus on one-way private-coin interactive proof systems (IPS) [2], where the verifier always sends the same symbol to prover. Since the protocol is one-way, the whole responses of the prover can be seen as an infinite string and this string is called as (membership) certificate. Since the prover always sends a symbol when requested, the certificates are assumed to be infinite. The automaton reads the provided certificate in one-way mode and so it can make pauses on some symbols of the certificate.

Formally, a PostPFA verifier  $V$  is a 7-tuple

$$V = (\Sigma, \Upsilon, S, \delta, s_I, s_{pa}, s_{pr}),$$

where, different from a PostPFA,  $\Upsilon$  is the certificate alphabet, and the transition function is extended as  $\delta : S \times \tilde{\Sigma} \times \Upsilon \times S \times \{0, 1\} \rightarrow [0, 1]$ . When  $V$  is in state  $s \in S$ , reads input symbol  $\sigma \in \tilde{\Sigma}$ , and reads certificate symbol  $v \in \Upsilon$ , it switches to state  $s' \in S$  and makes the action  $d \in \{0, 1\}$  on the certificate with probability  $\delta(s, \sigma, v, s', d)$ , where the next (respectively, the same) symbol of the certificate is selected for the next step if  $d = 1$  (respectively,  $d = 0$ ).

To be a well formed machine, the transition function must satisfy that

$$\text{for any } (s, \sigma, v) \in S \times \tilde{\Sigma} \times \Upsilon, \quad \sum_{s' \in S, d \in \{0,1\}} \delta(s, \sigma, v, s', d) = 1.$$

Let  $w \in \Sigma^*$  be the given input. For a given certificate, say  $c_w \in \Upsilon^\infty$ ,  $V$  starts in state  $s_I$  and reads the input and certificate in realtime and one-way modes, respectively. After finishing the input, it gives its decision like a standard PostPFA.

Language  $L \subseteq \Sigma^*$  is said to be verified by a PostPFA  $V$  with error bound  $\epsilon$  if the following two conditions (called completeness and soundness) are satisfied:

1. For any member  $w \in L$ , there exists a certificate, say  $c_w$ , such that  $V$  accepts  $w$  with probability at least  $1 - \epsilon$ .
2. For any non-member  $w \notin L$  and for any certificate  $c \in \Upsilon^\infty$ ,  $V$  always rejects  $w$  with probability at least  $1 - \epsilon$ .

We can also say that  $L$  is verified by  $V$  with bounded error. If every member is accepted with probability 1, then it is also said that  $L$  is verified by  $V$  with perfect completeness.

A two-way probabilistic finite automaton (2PFA) [11] is a generalization of a PFA which can read the input more than once. For this purpose, the input is written on a tape between two end-markers and each symbol is accessed by the read-only head of the tape. The head can either stay on the same symbol or move one square to the left or to the right by guaranteeing not to

leave the end-markers. The transition function is extended to determine the head movement after a transition. A 2PFA is called sweeping PFA if the direction of the head is changed only on the end-markers. The input is read from left to right, and then right to left, and so on.

A 2PFA can also be extended with an integer counter or a working tape – such model is called two-way probabilistic counter automaton (2PCA) or probabilistic Turing machine (PTM), respectively.

A 2PCA reads a single bit of information from the counter, i.e., whether its value is zero or not, as a part of a transition; and then, it increases or decreases the value of counter by 1 or does not change the value after the transition.

The working tape contains only blank symbols at the beginning of the computation and it has a two-way read/write head. On the work tape, a PTM reads the symbol under the head as a part of a transition, and then, it overwrites the symbol under the head and updates the position of head by at most one square after the transition.

Sweeping or realtime (postselecting) variants of these models are defined similarly.

For non-realtime models, the computation is terminated after entering an accepting or rejecting state. Additionally, for non-realtime postselecting models, there is another halting state for non-postselecting outcomes.

A language  $L$  is recognized by a bounded-error PTM (or any other variant of PTM) in space  $s(n)$ , if the maximum number of visited cells on the work tape with non-zero probability is not more than  $s(n)$  for any input with length  $n$ . If we replace the PTM with a counter automaton, then we take the maximum absolute value of the counter.

We denote the set of integers  $\mathbb{Z}$  and the set of positive integers  $\mathbb{Z}^+$ . The set  $\mathcal{I} = \{I \mid I \subseteq \mathbb{Z}^+\}$  is the set of all subsets of positive integers and so it is an uncountable set (the cardinality is  $\aleph_1$ ) like the set of real numbers ( $\mathbb{R}$ ). The cardinality of  $\mathbb{Z}$  or  $\mathbb{Z}^+$  is  $\aleph_0$  (countably many).

For  $I \in \mathcal{I}$ , the membership of each positive integer is represented as a binary probability value:

$$p_I = 0.x_101x_201x_301 \cdots x_i01 \cdots, \quad x_i = 1 \leftrightarrow i \in I.$$

The coin landing on head with probability  $p_I$  is named  $\text{coin}_I$ .

### 3. Rational-valued Postselecting Models

In this section, our recognizers and verifiers use only rational-valued transition probabilities.

### 3.1. PostPFA Algorithms

Here we mainly adopt and also simplify the techniques presented in [9, 3, 17]. We start with a simple language:  $\text{EQUAL} = \{0^m 10^n \mid m > 0\}$ . It is known that  $\text{EQUAL}$  is recognized by PostPFAs with bounded error [17, 19], but we still present an explicit proof which will be used in the other proofs.

**Fact 1** For any  $x < \frac{1}{2}$ ,  $\text{EQUAL}$  is recognized by a PostPFA  $P_x$  with error bound  $\frac{2x}{2x+1}$ .

*Proof.* Let  $w = 0^m 10^n$  be the given input for some  $m, n > 0$ . Any other input is rejected deterministically.

At the beginning of the computation,  $P_x$  splits the computation into two paths with equal probabilities. In the first path,  $P_x$  says “A” with probability  $Pr[A] = x^{2m+2n}$ , and, in the second path, it says “R” with probability  $Pr[R] = \left(\frac{x^{4m} + x^{4n}}{2}\right)$ .

In the first path,  $P_x$  starts in a state, say  $s_A$ . Then, for each symbol 0, it stays in  $s_A$  with probability  $x^2$  and quits  $s_A$  with the remaining probability. Thus, when started in  $s_A$ , the probability of being in  $s_A$  upon reaching on the right end-marker is

$$\underbrace{x^2 \cdot x^2 \cdot \dots \cdot x^2}_m \cdot \underbrace{x^2 \cdot x^2 \cdot \dots \cdot x^2}_n = x^{2m} \cdot x^{2n} = x^{2m+2n}.$$

In the second path, we assume that  $P_x$  starts in a state, say  $s_R$ , and then immediately switches to two different states, say  $s_{R1}$  and  $s_{R2}$ , with equal probabilities. For each 0 until the symbol 1,  $P_x$  stays in  $s_{R1}$  with probability  $x^4$  and quits  $s_{R1}$  with the remaining probability. After reading symbol 1, it switches from  $s_{R1}$  to  $s'_{R1}$  and stays there until the right end-marker. Thus, when started in  $s_{R1}$ , the probability of being in  $s'_{R1}$  upon reaching on the right end-marker is  $x^{4m}$ .

When in  $s_{R2}$ ,  $P_x$  stays in  $s_{R2}$  on the first block of 0's. After reading symbol 1, it switches from  $s_{R2}$  to  $s'_{R2}$ , and then, for each 0, it stays in  $s'_{R2}$  with probability  $x^4$  and quits  $s'_{R2}$  with the remaining probability. Thus, when started in  $s_{R2}$ , the probability of being in  $s'_{R2}$  upon reaching on the right end-marker is  $x^{4n}$ . Therefore, when started in state  $s_R$ , the probability of being in  $s'_{R1}$  or  $s'_{R2}$  upon reaching on the right end-marker is

$$\frac{x^{4m} + x^{4n}}{2}.$$

It is easy to see that if  $m = n$ , then  $Pr[A] = Pr[R] = x^{4m}$ . On the other hand, if  $m \neq n$ , then

$$\frac{Pr[R]}{Pr[A]} = \frac{\frac{x^{4m} + x^{4n}}{2}}{x^{2m+2n}} = \frac{x^{2m-2n}}{2} + \frac{x^{2n-2m}}{2} > \frac{1}{2x^2}$$

since either  $(2m - 2n)$  or  $(2n - 2m)$  is a negative even integer.

On the right end-marker,  $P_x$  enters  $s_{pa}$  and  $s_{pr}$  with probabilities  $Pr[A]$  and  $(x \cdot Pr[R])$ , respectively. Hence, if  $w$  is a member, then  $a(w)$  is  $x^{-1}$  times of  $r(w)$ , and so,  $w$  is accepted with probability

$$\frac{x^{-1}}{1+x^{-1}} = \frac{1}{x+1}.$$

If  $w$  is not a member, then  $r(w)$  is at least  $\frac{1}{2x}$  times of  $a(w)$ , and so,  $w$  is rejected with probability at least

$$\frac{(2x)^{-1}}{1+(2x)^{-1}} = \frac{1}{2x+1}.$$

Thus, the error bound  $\epsilon$  is  $\frac{2x}{2x+1}$ , i.e.

$$\epsilon = \max\left(1 - \frac{1}{x+1}, 1 - \frac{1}{2x+1}\right) = 1 - \frac{1}{2x+1} = \frac{2x}{2x+1},$$

which is less than  $\frac{1}{2}$  when  $x < \frac{1}{2}$ . (Remark that  $\epsilon \rightarrow 0$  when  $x \rightarrow 0$ .) □

We continue with language EQUAL-BLOCKS =  $\{0^{m_1}10^{n_1}10^{m_2}10^{n_2}1 \dots 10^{m_t}10^{n_t} \mid t > 0\}$ .

**Theorem 3.1** *For any  $x < \frac{1}{2}$ , EQUAL-BLOCKS is recognized by a PostPFA  $P_x$  with error bound  $\frac{2x}{2x+1}$ .*

*Proof.* Let  $w = 0^{m_1}10^{n_1}10^{m_2}10^{n_2}1 \dots 10^{m_t}10^{n_t}$  be the given input for some  $t > 0$ , where for each  $i \in \{1, \dots, t\}$  both  $m_i$  and  $n_i$  are positive integers. Any other input is rejected deterministically.

Similar to the previous proof, after reading whole input,  $P_x$  says ‘‘A’’ with probability

$$Pr[A] = \underbrace{(x^{2m_1+2n_1})}_{a_1} \underbrace{(x^{2m_2+2n_2})}_{a_2} \dots \underbrace{(x^{2m_t+2n_t})}_{a_t}$$

and says ‘‘R’’ with probability

$$Pr[R] = \underbrace{\left(\frac{x^{4m_1} + x^{4n_1}}{2}\right)}_{r_1} \underbrace{\left(\frac{x^{4m_2} + x^{4n_2}}{2}\right)}_{r_2} \dots \underbrace{\left(\frac{x^{4m_t} + x^{4n_t}}{2}\right)}_{r_t}.$$

Here  $P_x$  can easily implement both probabilistic events by help of internal states. As analyzed in the previous proof, for each  $i \in \{1, \dots, t\}$ , either  $a_i = r_i$  or  $r_i$  is at least  $\frac{1}{2x^2}$  times greater than  $a_i$ . Thus, if  $w$  is a member, then  $Pr[A] = Pr[R]$ , and, if  $w$  is not a member, then

$$\frac{Pr[R]}{Pr[A]} > \frac{1}{2x^2}.$$

On the right end-marker,  $P_x$  enters  $s_{pa}$  and  $s_{pr}$  with probabilities  $Pr[A]$  and  $(x \cdot Pr[R])$ , respectively. Hence, we obtain the same error bound as given in the previous proof. □

Let  $f$  be the linear mapping  $f(m) = am + b$  for some nonnegative integers  $a$  and  $b$ , and, let EQUAL-BLOCKS( $\mathbf{f}$ ) =  $\{0^{m_1}10^{f(m_1)}10^{m_2}10^{f(m_2)}1 \dots 10^{m_t}10^{f(m_t)} \mid t > 0\}$  be a new language.

**Theorem 3.2** For any  $x < \frac{1}{2}$ , EQUAL-BLOCKS( $f$ ) is recognized by a PostPFA  $P_x$  with error bound  $\frac{2x}{2x+1}$ .

*Proof.* Let  $w = 0^{m_1}10^{n_1}10^{m_2}10^{n_2}1 \dots 10^{m_t}10^{n_t}$  be the given input for some  $t > 0$ , where for each  $i \in \{1, \dots, t\}$  both  $m_i$  and  $n_i$  are positive integers. Any other input is rejected deterministically.

In the above proofs, the described automata make transitions with probabilities  $x^2$  or  $x^4$  when reading a symbol 0. Here  $P_x$  makes some additional transitions:

- Before starting to read a block of 0's,  $P_x$  makes a transition with probability  $x^{2b}$  or  $x^{4b}$ .
- After reading a symbol 0,  $P_x$  makes a transition with probability  $x^{2a}$  or  $x^{4a}$ .

Thus, after reading a block of  $m$  0's,  $P_x$  can be designed to be in a specific event with probability  $x^{2am+2b} = x^{2f(m)}$  or  $x^{4am+4b} = x^{4f(m)}$ , where  $m > 0$ .

Therefore,  $P_x$  is constructed such that, after reading whole input, it says "A" with probability

$$Pr[A] = \underbrace{(x^{2f(m_1)+2n_1})}_{a_1} \underbrace{(x^{2f(m_2)+2n_2})}_{a_2} \dots \underbrace{(x^{2f(m_t)+2n_t})}_{a_t}$$

and says "R" with probability

$$Pr[R] = \underbrace{\left(\frac{x^{4f(m_1)} + x^{4n_1}}{2}\right)}_{r_1} \underbrace{\left(\frac{x^{4f(m_2)} + x^{4n_2}}{2}\right)}_{r_2} \dots \underbrace{\left(\frac{x^{4f(m_t)} + x^{4n_t}}{2}\right)}_{r_t}.$$

Then, for each  $i \in \{1, \dots, t\}$ , if  $n_i = f(m_i)$ ,  $a_i = r_i = x^{4f(m_i)}$ , and, if  $n_i \neq f(m_i)$ ,

$$\frac{r_i}{a_i} = \frac{\frac{x^{4f(m_i)} + x^{4n_i}}{2}}{x^{2f(m_i)+2n_i}} = \frac{x^{2f(m_i)-2n_i}}{2} + \frac{x^{2n_i-2f(m_i)}}{2} > \frac{1}{2x^2}.$$

As in the above algorithms, on the right end-marker,  $P_x$  enters  $s_{pa}$  and  $s_{pr}$  with probabilities  $Pr[A]$  and  $(x \cdot Pr[R])$ , respectively. Hence, we obtain the same error bound as given in the previous proofs.  $\square$

As an application of the last result, we present a PostPFA algorithm for language

$$\text{LOG} = \{010^{2^1}10^{2^2}10^{2^3} \dots 0^{2^{m-1}}10^{2^m} \mid m > 0\},$$

which was also shown to be recognized by 2PFAs [9].

**Theorem 3.3** For any  $x < \frac{1}{2}$ , LOG is recognized by a PostPFA  $P_x$  with error bound  $\frac{2x}{2x+1}$ .

*Proof.* Let  $0^{2^0}10^{m_1}10^{m_2}1 \dots 10^{m_t}$  be the given input for  $t > 1$ , where  $m_1 = 2^1$ . The decision on any other input is given deterministically.

After reading whole input,  $P_x$  says "A" with probability

$$Pr[A] = \underbrace{(x^{4m_1+2m_2})}_{a_1} \underbrace{(x^{4m_2+2m_3})}_{a_2} \dots \underbrace{(x^{4m_{t-1}+2m_t})}_{a_{t-1}}$$



and says “ $R$ ” with probability

$$Pr[R] = \underbrace{\left(\frac{x^{8m_1} + x^{4m_2}}{2}\right)}_{r_1} \underbrace{\left(\frac{x^{8m_2} + x^{4m_3}}{2}\right)}_{r_2} \cdots \underbrace{\left(\frac{x^{8m_{t-1}} + x^{4m_t}}{2}\right)}_{r_{t-1}}.$$

In the previous languages, the blocks are nicely separated, but for language LOG the blocks are overlapping. Therefore, we modify the previous methods. As described in the first algorithm,  $P_x$  splits the computation into two paths with equal probabilities at the beginning of the computation. In the first path, the event happening with probability  $Pr[A]$  is implemented by executing two parallel procedures: The first procedure produces the probabilities  $a_i$ 's where  $i$  is odd and the second procedure produces the probabilities  $a_i$ 's where  $i$  is even. Similarly, in the second path, the event happening with probability  $Pr[R]$  is implemented by also executing two parallel procedures. Thus, the previous algorithm is also used for LOG by using the solution for overlapping blocks.  $\square$

In [9], the following padding argument was given:

**Fact 2** [9] *If a binary language  $L$  is recognized by a bounded-error PTM in space  $s(n)$ , then the binary language  $\text{LOG}(L)$  is recognized by a bounded-error PTM in space  $\log(s(n))$ , where*

$$\text{LOG}(L) = \{0(1w_1)0^{2^1}(1w_2)0^{2^2}(1w_3)0^{2^3} \cdots 0^{2^{m-1}}(1w_m)0^{2^m} \mid w = w_1 \cdots w_m \in L\}.$$

Similarly, we can easily obtain the following two corollaries.

**Corollary 3.4** *If a binary language  $L$  is recognized by a bounded-error PostPTM in space  $s(n)$ , then the binary language  $\text{LOG}(L)$  is recognized by a bounded-error PostPTM in space  $\log(s(n))$ .*

**Corollary 3.5** *If a binary language  $L$  is recognized by a bounded-error PostPCA in space  $s(n)$ , then the binary language  $\text{LOG}(L)$  is recognized by a bounded-error PostPCA in space  $\log(s(n))$ .*

### 3.2. PostPFA Protocols

In this section, we present PostPFA protocols for the following two nonregular unary languages:  $\text{UPOWER} = \{0^{2^m} \mid m > 0\}$  and  $\text{USQUARE} = \{0^{m^2} \mid m > 0\}$ . These languages are known to be verified by 2PFA verifiers [8] and private alternating realtime automata [4]. Here, we use similar protocols but with certain modifications for PostPFAs.

**Theorem 3.6**  *$\text{UPOWER}$  is verified by a PostPFA  $V_x$  with perfect completeness, where  $x < 1$ .*

*Proof.* Let  $w_m$  be the  $m$ -th shortest member of  $\text{UPOWER}$  ( $m > 0$ ) and let  $w = 0^n$  be the given string for  $n > 1$ . (If the input is empty string or 0, then it is rejected deterministically.)

The verifier expects the certificate to be composed by  $t > 0$  block(s) followed by symbol \$, and each block has form of  $0^+1$  except the last one which is 1. The verifier also never checks a

new symbol on the certificate after reading a \$ symbol. Let  $c_w$  be the given certificate in this format:

$$c_w = u_1 \cdots u_{t-1} u_t \$ \$^*,$$

where for each  $j \in \{1, \dots, t-1\}$ ,  $u_j \in \{0^+1\}$ , and  $u_t = 1$ . Any other certificate is detected deterministically, and then, the input is rejected. Let  $u_w = u_1 \cdots u_{t-1} u_t \$$  and  $l_j = |u_j|$ .

The verifier checks that (1)  $l_j$  is twice of  $l_{j+1}$  for each  $j \in \{1, \dots, t-2\}$ , (2) each block except the last one contains at least one 0 symbol, (3) the last block is 1, and (4)  $|w| = |u_w|$ . Remark that these conditions are satisfied only for members: The expected certificate for  $w_m$  is

$$c_{w_m} = \underbrace{0^{2^{m-1}-1}1}_{1st\ block} \underbrace{0^{2^{m-2}-1}1}_{2nd\ block} \cdots 1 \underbrace{0001}_{\dots} \underbrace{01}_{\dots} \underbrace{1}_{m-th\ block} \$ \$^*$$

and the length of all blocks and a single \$ symbol is  $2^{m-1} + 2^{m-2} + \cdots + 2^1 + 2^0 + 1 = 2^m$ . In other words,  $l_1 = \frac{|w|}{2}$ ,  $l_2 = \frac{|w|}{4}$ ,  $\dots$ ,  $l_m = \frac{|w|}{2^m}$ .

At the beginning of the computation,  $V_x$  splits the computation into two paths with equal probabilities, called the accepting path and the main path. In the accepting path, the computation ends in  $s_{pa}$  with probability  $\frac{x}{2^t}$  and in some non-postselecting state with the remaining probability. Since there are  $t$  blocks, it is easy to obtain this probability. This is the path in which  $V_x$  enters  $s_{pa}$ . Therefore,  $a(w) = \frac{x}{2^{t+1}}$  (the accepting path is selected with probability  $\frac{1}{2}$ ).

During reading the input and the certificate, the main path checks (1) whether  $|w| = |u_w|$ , (2) each block of the certificate except the last one contains at least one 0 symbol, and (3) the last block is 1. If one of checks fails, the computation ends in state  $s_{pr}$ . The main path also creates subpaths for checking whether  $l_1 = \frac{|w|}{2}$ ,  $l_2 = \frac{l_1}{2}$ ,  $\dots$ ,  $l_{m-1} = \frac{l_{m-2}}{2}$ . After the main path starts to read a block starting with 0 symbol, it creates a subpath with half probability and stays in the main path with remaining probability. Thus, the main path reaches the right end-marker with probability  $\frac{1}{2^t}$ . On the other hand, the  $j$ -th subpath is created with probability  $\frac{1}{2^{j+1}}$ , where  $1 \leq j \leq t-1$ .

The first subpath tries to read  $2l_1$  symbols from the input. If there are exactly  $2l_1$  symbols, i.e.  $2l_1 = |w|$ , then the test is successful and the computation is terminated in a non-postselecting state. Otherwise, the test is failed and the computation is terminated in state  $s_{pr}$ .

The second path is created after reading  $l_1$  symbols from the input. Then, the second subpath also tries to read  $2l_2$  symbols from the input. If there are exactly  $2l_2$  symbols, i.e.  $l_1 + 2l_2 = |w|$ , then the test is successful and the computation is terminated in a non-postselecting state. Otherwise, the test is failed and the computation is terminated in state  $s_{pr}$ .

The other subpaths behave exactly in the same way. The last  $((t-1)$ -th) subpath checks whether  $l_1 + l_2 + \cdots + l_{t-2} + 2l_{t-1} = |w|$ . If all previous tests are successful, then  $l_{t-1} = \frac{l_{t-2}}{2} = \frac{|w|}{2^{t-1}}$ .

It is clear that if  $w$  is a member, say  $w_m$ , and  $V_x$  reads  $w_m$  and  $c_{w_m}$ , then  $a(w) = \frac{x}{2^{m+1}}$ . On the other hand, neither the main path nor any subpath enters state  $s_{pr}$  with some non-zero probability. Therefore, any member is accepted with probability 1.

If  $w$  is not a member, then one of the checks done by the main path and the subpaths is failed and so  $V_x$  enters  $s_{pr}$  with non-zero probability. The probability of being in  $s_{pr}$  at the end, i.e.  $r(w)$ , is at least  $\frac{1}{2^t}$ . Thus,

$$\frac{r(w)}{a(w)} \geq \frac{\frac{1}{2^t}}{\frac{x}{2^{t+1}}} = \frac{2}{x}.$$

Therefore, any non-member is rejected with probability at least  $\frac{2}{2+x}$ . □

In the above proof, the verifier can also check deterministically whether the number of blocks is a multiple of  $k$  or not for some  $k > 1$ . Thus, we can easily conclude the following result.

**Corollary 3.7**  $\text{UPOWER}_k = \{0^{2^{km}} \mid m > 0\}$  is verified by a PostPFA with perfect completeness.

**Theorem 3.8**  $\text{USQUARE}$  is verified by a PostPFA  $V_x$  with perfect completeness, where  $x < 1$ .

*Proof.* The proof is very similar to the above proof. Let  $w_m$  be the  $m$ -th shortest member of  $\text{USQUARE}$  ( $m > 1$ ). Let  $w = 0^n$  be the given input for  $n > 3$ . (The decisions on the shorter strings are given deterministically.) The verifier expects to obtain a certificate composed by  $t$  blocks:

$$c_w = a^{m_1} b^{m_2} a^{m_3} \dots d^{m_t} \text{\$}\text{\$}^*,$$

where  $d$  is  $a$  ( $b$ ) if  $t$  is odd (even). Let  $u_w = a^{m_1} b^{m_2} a^{m_3} \dots d^{m_t} \text{\$}$ . The verifier never reads a new symbol after reading  $u_w$  on the certificate.

The verifier checks the following equalities:

$$m_1 = m_2 = \dots = m_t = t + 1$$

and

$$|w| = m_1 + m_2 + \dots + m_t + (t + 1).$$

If we substitute  $m_1$  with  $m$  in the above equalities, then we obtain that  $|w| = (m-1)m + m = m^2$  and so  $w = w_m$ .

At the beginning of the computation,  $V_x$  splits into the accepting path and the main path with equal probabilities, and, as a result of the accepting path, it always enters  $s_{pa}$  with probability  $a(w) = \frac{x}{2^{t+1}}$ .

In the following paths, if the comparison is successful, then the computation is terminated in a non-postselecting state, and, if it is not successful, then the computation is terminated in state  $s_{pr}$ . The main path checks the equality  $|w| = m_1 + m_2 + \dots + m_t + (t + 1)$ .

For each  $j \in \{1, \dots, t\}$ , the main path also creates a subpath with probability  $\frac{1}{2}$  and remains in the main path with the remaining probability. The  $j$ -th subpath checks the equality

$$|w| = m_j + m_1 + \dots + m_t,$$

where  $m_j$  is added twice.

If all comparisons in the subpaths are successful, then we have

$$m_1 = m_2 = \cdots = m_t = m$$

for some  $m > 0$ . Additionally, if the comparison in the main path is successful, then we obtain that  $t = m - 1$ . Thus,  $w = w_m$ . Therefore, any member is accepted with probability 1 by help of the proof composed by  $(m - 1)$  blocks and the length of each block is  $m$ .

If  $w$  is not a member, then one of the comparisons will not be successful. (If all are successful, then, as described above, the certificate should have  $(m - 1)$  blocks of length  $m$  and the input has length  $m^2$ .) The minimum value of  $r(w)$  is at least  $\frac{1}{2^{t+1}}$  and so  $\frac{r(w)}{a(w)} \geq \frac{1}{x}$ . Therefore, any non-member is rejected with probability at least  $\frac{1}{x+1}$ .  $\square$

## 4. Postselecting Models Using Magic Coins

In this section, we allow recognizers and verifiers to use real-valued transition probabilities. We use a fact presented in our previous paper [5].

**Fact 3** [5] *Let  $x = x_1x_2x_3\cdots$  be an infinite binary sequence. If a biased coin lands on head with probability  $p = 0.x_101x_201x_301\cdots$ , then the value  $x_k$  is determined correctly with probability at least  $\frac{3}{4}$  after  $64^k$  coin tosses, where  $x_k$  is guessed as the  $(3k+3)$ -th digit of the binary number representing the total number of heads after the whole coin tosses.*

### 4.1. Algorithms Using Magic Coins

Previously, we obtained the following result.

**Fact 4** [5] *Bounded-error linear-space sweeping PCAs can recognize uncountably many languages in subquadratic time.*

For the language  $L$  recognized by a sweeping PCA, we can easily design a sweeping PCA that recognizes  $\text{LOG}(L)$  by using the same idea given for PostPFAs. Since PostPFAs are equivalent to restart-PFAs and restart-PFAs can also be implemented by sweeping PFAs, we can reduce linear space to logarithmic space given in the above result with exponential slowdown, i.e. padding part of the input can be recognized by restart-PFA with exponential expected time.

**Corollary 4.1** *Bounded-error log-space sweeping PCAs can recognize uncountably many languages in exponential expected time.*

We can iteratively apply this idea and obtain new languages with better and better space bounds. We can define  $\text{LOG}^k(L)$  as  $\text{LOG}(\text{LOG}^{k-1}(L))$  for  $k > 1$  and then we can follow that  $\text{LOG}^k(L)$  can be recognized by a bounded-error sweeping PCA that uses  $O(\log^k(n))$  space on the counter.

**Corollary 4.2** *The cardinality of languages recognized by bounded-error sweeping PCAs with arbitrary small non-constant space bound is uncountably many.*

Now, we show how to obtain the same results by restricting sweeping reading mode to the restarting realtime reading mode or realtime reading mode with postselection. We start with the recognition of the following nonregular binary language, a modified version of DIMA [5]:

$$\text{DIMA3} = \{0^{2^0} 10^{2^1} 10^{2^2} 1 \dots 10^{2^{6k-2}} 110^{2^{6k-1}} 11^{2^{6k}} (0^{2^{3k}-1} 1)^{2^{3k}} \mid k > 0\}.$$

**Theorem 4.3** *For any  $x < \frac{1}{3}$ , DIMA3 is recognized by linear-space PostPCA  $P_x$  with error bound  $\frac{x}{1+x}$ .*

*Proof.* Let  $w$  be the given input of the form

$$w = 0^{t_1} 10^{t_2} 1 \dots 10^{t_{m-1}} 110^{t_m} 11^{t'_0} 0^{t'_1} 10^{t'_2} 1 \dots 10^{t'_n} 1,$$

where  $t_1 = 1$ ,  $m$  and  $n$  are positive integers,  $m$  is divisible by 6, and  $t_i, t'_j > 0$  for  $1 \leq i \leq m$  and  $0 \leq j \leq n$ . (Otherwise, the input is rejected deterministically.)

$P_x$  splits computation into four paths with equal probabilities. In the first path, with the help of the counter,  $P_x$  makes the following comparisons:

- for each  $i \in \{1, \dots, \frac{m}{2}\}$ , whether  $2t_{2i-1} = t_{2i}$ ,
- for each  $j \in \{1, \dots, \frac{n}{2}\}$ , whether  $t'_{2j-1} = t'_{2j}$ .

In the second path, with the help of the counter,  $P_x$  makes the following comparisons:

- for each  $i \in \{1, \dots, \frac{m}{2} - 1\}$ , whether  $2t_{2i} = t_{2i+1}$ ,
- whether  $2t_m = t'_0$  (this also helps to set the counter to 0 for the upcoming comparisons),
- for each  $j \in \{1, \dots, \frac{n}{2} - 1\}$ , whether  $t'_{2j} = t'_{2j+1}$ .

In the third path,  $P_x$  checks whether  $1 + \sum_{i=1}^m t_i = n + \sum_{j=1}^n t'_j$ . In the fourth path  $P_x$  checks, whether  $t'_1 + 1 = n$ .

It is easy to see that all comparisons are successful if and only if  $w \in \text{DIMA3}$ .

If every comparison in a path is successful, then  $P_x$  enters  $s_{pa}$  with probability  $\frac{x}{3}$  in the path. If it is not, then  $P_x$  enters  $s_{pr}$  with probability 1 in the path. Therefore, if  $w \in \text{DIMA3}$ , then  $w$  is accepted with probability 1 since  $r(w) = 0$ . If  $w \notin \text{DIMA3}$ , then the maximum accepting probability is obtained when  $P_x$  enters  $s_{pr}$  only in one of the paths. That is,  $\frac{r(x)}{a(x)} = \frac{\frac{1}{4} \cdot \frac{x}{3}}{3 \cdot \frac{1}{4} \cdot \frac{x}{3}} = \frac{1}{x}$ . Thus,  $w$  is rejected with probability at least  $\frac{1}{1+x}$ . The error bound is  $\frac{x}{1+x}$ .  $\square$

**Theorem 4.4** *Linear-space PostPCAs can recognize uncountably many languages with error bound  $\frac{2}{5}$ .*

*Proof.* Let  $w_k$  be the  $k$ -th shortest member of DIMA3 for  $k > 0$ . For any  $I \in \mathcal{I}$ , we define the following language:

$$\text{DIMA3}(I) = \{w_k \mid k > 0 \text{ and } k \in I\}.$$

We follow our result by presenting a PostPCA, say  $P_{I,y}$ , to recognize DIMA3( $I$ ), where  $y < \frac{1}{19}$ . Let  $w$  be the given input of the form

$$w = 0^{t_1} 10^{t_2} 1 \dots 10^{t_{m-1}} 110^{t_m} 11^{t'_0} 0^{t'_1} 10^{t'_2} 1 \dots 10^{t'_n} 1,$$

where  $t_1 = 1$ ,  $m$  and  $n$  are positive integers,  $m$  is divisible by 6, and  $t_i, t'_j > 0$  for  $1 \leq i \leq m$  and  $0 \leq j \leq n$ . (Otherwise, the input is rejected deterministically.)

At the beginning of the computation,  $P_{I,y}$  splits into two paths with equal probabilities. In the first path,  $P_{I,y}$  executes the PostPCA  $P_y$  for DIMA3 described in the proof above with the following modification: in each path of  $P_y$ , if every comparison is successful, then  $P_y$  enters state  $s_{pa}$  with probability  $\frac{y}{16}$  ( $P_y$  enters path with probability  $\frac{1}{4}$ , and then enters state  $s_{pa}$  with probability  $\frac{y}{4}$ ), and, if it is not, then  $P_y$  enters state  $s_{pr}$  with probability 1.

In the second path,  $P_{I,y}$  sets the value of counter to  $T = 1 + \sum_{j=1}^m t_j$  by reading the part of the input  $0^{t_1} 10^{t_2} 1 \dots 10^{t_{m-1}} 110^{t_m} 1$ . Remark that if  $w \in \text{DIMA3}$ ,  $T$  is  $64^k$  for some  $k > 0$ . Then,  $P_{I,y}$  attempts to toss  $\text{coin}_I$   $T$  times. After each coin toss, if the result is a head (resp., tail), then  $P_{I,y}$  moves on the input two symbols (respectively, one symbol). If  $H$  is the number of total heads, then  $P_{I,y}$  reads  $(T - H) + 2H = T + H$  symbols. During attempt to read  $T + H$  symbols, if the input is finished, then the computation ends in state  $s_{pr}$  with probability 1 in this path. Otherwise,  $P_{I,y}$  guesses the value  $x_k$  with probability at least  $\frac{3}{4}$  (described in details at the end of the proof) and gives a parallel decision with probability  $y$ , i.e., if the guess is 1 (resp., 0), then it enters state  $s_{pa}$  (resp.,  $s_{pr}$ ) with probability  $y$ .

If  $w \in \text{DIMA3}(I)$ , then the probability of entering state  $s_{pa}$  is  $(4 \cdot \frac{y}{16})$  in the first path and at least  $\frac{3y}{4}$  in the second path. The probability of entering  $s_{pr}$  in the second path is at most  $\frac{y}{4}$ . Thus,  $w$  is accepted with probability at least  $\frac{4}{5}$ .

If  $w \notin \text{DIMA3}(I)$ , then we have two cases:

Case 1:  $w \in \text{DIMA3}$ . In this case, the probability of entering state  $s_{pa}$  is  $(4 \cdot \frac{y}{16})$  in the first path and at most  $\frac{y}{4}$  in the second path. The probability of entering  $s_{pr}$  in the second path is at least  $\frac{3y}{4}$ . Thus,  $w$  is rejected with probability  $\frac{3}{5}$ .

Case 2:  $w \notin \text{DIMA3}$ . In this case, the probability of entering state  $s_{pr}$  is at least  $\frac{1}{8}$  in the first path and this is at least 4 times of the total probability of entering state  $s_{pa}$ , which can be at most

$$\frac{1}{2} \cdot 3 \cdot \frac{y}{16} + \frac{1}{2}y = \frac{19y}{32} < \frac{1}{32}$$

for  $y < \frac{1}{19}$ . Then, the input is rejected with probability greater than  $\frac{4}{5}$ .

As can be seen from the above analysis, when  $w \notin \text{DIMA3}$ , guessing the correct value of  $x_k$  is insignificant. Therefore, in the following part, we assume that  $w \in \text{DIMA3}$  when explaining how to guess  $x_k$  correctly. Thus, we assume that  $w = w_k$ :

$$w_k = 0^{2^0} 10^{2^1} 10^{2^2} 1 \dots 10^{2^{6k-2}} 110^{2^{6k-1}} 11^{2^{6k}} (0^{2^{3k}-1})^{2^{3k}}$$

for  $k > 0$ . In the second path,  $P_{I,y}$  tosses  $\text{coin}_I$   $T = 64^k$  times and it can read  $64^k + H$  symbols from the input. In other words, it reads  $H$  symbols from the part  $w'_k = (0^{2^{3k}-1})^{2^{3k}}$ . Here we

use the analysis similar to one presented in [8]. We can write  $H$  as

$$H = i \cdot 8^{k+1} + j \cdot 8^k + q = (8i + j)8^k + q,$$

where  $i \geq 0$ ,  $j \in \{0, \dots, 7\}$ , and  $q < 8^k$ .

Due to Fact 3,  $x_k$  is the  $(3k + 3)$ -th digit of  $\text{binary}(H)$  with probability  $\frac{3}{4}$ . In other words,  $x_k$  is guessed as 1 if  $j \in \{4, \dots, 7\}$ , and as 0, otherwise.  $P_{I,y}$  sets  $j = 0$  at the beginning. We can say that for each head, it consumes a symbol from  $w'_k$ . After reading  $8^k$  symbols, it updates  $j$  as  $(j + 1) \bmod 8$ . When the value of counter reaches zero,  $P_{I,y}$  guesses  $x_k$  by checking the value of  $j$ .  $\square$

Now we can combine Corollary 3.5 and Theorem 4.4 to obtain new results for hierarchy of uncountable probabilistic classes.

**Corollary 4.5** *The cardinality of languages recognized by bounded-error PostPCAs with arbitrary small non-constant space bound is uncountably many.*

## 4.2. Protocols Using Magic Coins

In this subsection we proceed with the verification of uncountably many unary languages.

**Theorem 4.6** *PostPFAs can verify uncountably many unary languages with bounded error.*

*Proof.* See [7] for the proof.  $\square$

## Acknowledgements

Dimitrijevs is partially supported by University of Latvia projects AAP2016/B032 “Innovative information technologies” and ZD2018/20546 “For development of scientific activity of Faculty of Computing”. Yakaryılmaz is partially supported by ERC Advanced Grant MQC.

## References

- [1] S. AARONSON, Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A* 461 (2005) 2063, 3473–3482.
- [2] A. CONDON, *Complexity Theory: Current Research*, chapter The complexity of space bounded interactive proof systems. Cambridge University Press, 1993, 147–190.
- [3] A. CONDON, R. J. LIPTON, On the complexity of space bounded interactive proofs (Extended Abstract). In: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*. 1989, 462–467.

- [4] H. G. DEMIRCI, M. HIRVENSALO, K. REINHARDT, A. C. C. SAY, A. YAKARYILMAZ, Classical and quantum realtime alternating automata. In: S. BENSCH, R. FREUND, F. OTTO (eds.), *Sixth Workshop on Non-Classical Models of Automata and Applications (NCMA 2014)*. books@ocg.at 304, Österreichische Computer Gesellschaft, Wien, 2014, 101–114. Preprint: (arXiv:1407.0334).
- [5] M. DIMITRIJEVS, A. YAKARYILMAZ, Uncountable Classical and Quantum Complexity Classes. In: H. BORDIHN, R. FREUND, B. NAGY, GYÖRGY VASZIL (eds.), *Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016)*. books@ocg.at 321, Österreichische Computer Gesellschaft, Wien, 2016, 131–146. Preprint: (arXiv:1608.00417).
- [6] M. DIMITRIJEVS, A. YAKARYILMAZ, Uncountable Realtime Probabilistic Classes. In: G. PIGHIZZINI, C. CÂMPEANU (eds.), *Descriptive Complexity of Formal Systems*. Lecture Notes in Computer Science 10316, Springer, 2017, 102–113. Preprint: (arXiv:1705.01773).
- [7] M. DIMITRIJEVS, A. YAKARYILMAZ, *Postselecting probabilistic finite state recognizers and verifiers*. Technical Report 1807.05169, arXiv, 2018.
- [8] M. DIMITRIJEVS, A. YAKARYILMAZ, *Probabilistic verification of all languages*. Technical Report 1807.04735, arXiv, 2018.
- [9] R. FREIVALDS, Probabilistic two-way machines. In: J. GRUSKA, M. CHYTIL (eds.), *Proceedings of the International Symposium on Mathematical Foundations of Computer Science (MFCS 1981)*. Lecture Notes in Computer Science 118, Springer, 1981, 33–45.
- [10] J. KAŇEPS, Regularity of One-Letter Languages Acceptable by 2-Way Finite Probabilistic Automata. In: L. BUDACH (ed.), *Fundamentals of Computation Theory (FCT'91)*. Lecture Notes in Computer Science 529, Springer, 1991, 287–296.
- [11] Y. I. KUKLIN, Two-way probabilistic automata. *Avtomatika i vychislitel'naja tekhnika* 5 (1973), 36–36. (Russian).
- [12] A. C. C. SAY, A. YAKARYILMAZ, Computation with narrow CTCs. In: C. S. CALUDE, J. KARI, I. PETRE, G. ROZENBERG (eds.), *Unconventional Computation - 10th International Conference, UC 2011, Turku, Finland, June 6-10, 2011. Proceedings*. Lecture Notes in Computer Science 6714, Springer, 2011, 201–211.
- [13] A. C. C. SAY, A. YAKARYILMAZ, Computation with multiple CTCs of fixed length and width. *Natural Computing* 11 (2012) 4, 579–594.
- [14] A. C. C. SAY, A. YAKARYILMAZ, Magic coins are useful for small-space quantum machines. *Quantum Information & Computation* 17 (2017) 11&12, 1027–1043.
- [15] O. SCEGULNAJA-DUBROVSKA, R. FREIVALDS, A context-free language not recognizable by postselection finite quantum automata. In: R. FREIVALDS (ed.), *Randomized and quantum computation*. 2010, 35–48. Satellite workshop of MFCS and CSL 2010.
- [16] O. SCEGULNAJA-DUBROVSKA, L. LĀČE, R. FREIVALDS, Postselection finite quantum automata. In: C. CALUDE, M. HAGIYA, K. MORITA, G. ROZENBERG, J. TIMMIS (eds.), *Unconventional Computation (UC 2010)*. Lecture Notes in Computer Science 6079, Springer, 2010, 115–126.



- [17] A. YAKARYILMAZ, A. C. C. SAY, Succinctness of two-way probabilistic and quantum finite automata. *Discrete Mathematics and Theoretical Computer Science* 12 (2010) 2, 19–40.
- [18] A. YAKARYILMAZ, A. C. C. SAY, *Probabilistic and quantum finite automata with postselection*. Technical Report 1102.0666, arXiv, 2011. (A preliminary version of this paper appeared in the *Proceedings of Randomized and Quantum Computation (satellite workshop of MFCS and CSL 2010)*, pages 14–24, 2010).
- [19] A. YAKARYILMAZ, A. C. C. SAY, Proving the power of postselection. *Fundamenta Informaticae* 123 (2013) 1, 107–134.



# AUTOMATA THAT MAY CHANGE THEIR MIND

Markus Holzer     Martin Kutrib

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

## **Abstract**

*We introduce the concept of mind-changing automata. Basically, the idea is that at the outset the automaton is partially deterministic. Whenever the automaton encounters a situation for which it has an undefined transition, it may choose an appropriate transition out of a set of available transitions. The chosen transition is then added to the transition function. Finally, whenever the automaton is in a situation for which a transition is defined, it can change its mind and interchange the transition by an alternative transition from the set of available transitions. So, the number of transition changes is a natural parameter of the devices considered. We show that mind-changing finite automata (MCFAs) only accept regular languages. Moreover, we prove that, from a descriptive complexity point of view, a single mind-change is already better than nondeterminism, that is, there is a sequence of regular languages  $(L_n)_{n \geq 3}$  accepted by  $n$ -state complete MCFAs with a single alternative transition with at most one mind-change such that any nondeterministic finite automaton accepting  $L_n$  requires at least  $n + \log n - 1$  states. We also consider mind-changing pushdown automata proving that the families of languages induced by the number of mind-changes lie strictly in between the deterministic context-free and context-free language families and form a proper mind-change hierarchy.*

## **1. Introduction**

In linguistics it is generally accepted that language and thought influence each other. As said by Chomsky [1], “[...] language is a mirror of mind in a deep and significant sense.” Although language is a mirror of mind, it is not a model of it. States of mind are used by Turing to model physical processes by abstract machines [5, 6], where the states of mind form a finite control. Thus, simply speaking, an abstract machine or automaton is a device with a finite control and an additional storage such as, for example, a pushdown or Turing tape, that can be manipulated by a finite number of operations. The behavior, that is, the transition function, of the abstract automaton is programmed and cannot be changed. Hence, the states of mind and their relation to each other is fixed. It seems that a re-interpretation of the mind’s opinion on certain states and relations to each other is not possible. This re-interpretation corresponds to a re-programming of the underlying automaton. In fact, re-programming of an automaton is possible, whenever a universal machine of the same type exists, as in case of Turing machines.

But what about other devices such as, for instance, finite automata or pushdown machines, where no universal device exists? In order to allow also these machines a re-programming during the computation, we propose a novel approach on mind-changing automata. It turns out that this approach generalizes the recently introduced concept of one-time nondeterministic computations [2], which is an alternative interpretation of nondeterminism.

A mind-changing automaton is a device which is partially deterministic at the outset. Whenever a situation encounters for which the automaton has an undefined transition, it may choose an appropriate transition out of a set of available transitions. The chosen transition is then added to the transition function. Finally, whenever the automaton is in a situation for which a transition is defined, it can change its mind and interchange the transition by an alternative transition from the set of available transitions. Here an interchanging of transitions is said to be a *mind-change*. The initialization or change of a transition is fixed up to the next time when the automaton decides to change this particular transition again. Then the number of transition changes is a natural parameter of the considered device. We investigate mind-changing finite automata (MCFAs) and mind-changing pushdown automata (MCPDAs). First we show that the mind-changing mechanism for both types of automata can be simulated by nondeterministic machines of the same type. Thus, MCFAs accept only regular languages and MCPDAs only context-free sets. Hence in case of MCFAs the question on the descriptonal complexity of these machines arises, while for MCPDAs the question whether the language families induced by a constant number of mind-changes form a proper hierarchy between the deterministic context-free and context-free languages arises. For MCFAs we obtain an upper bound on the number of states an equivalent nondeterministic or deterministic automaton needs to accept the language under consideration. Since MCFAs without any mind-change operation are shown to characterize the one-time nondeterministic finite automata recently introduced in [2], lower bounds for one-time nondeterministic automata immediately transfer to MCFAs. Roughly speaking, *one-time* nondeterminism means that at the outset the computation is nondeterministic, but whenever the automaton performs a guess, this guess is fixed for the rest of the computation. Moreover, we show that a single mind-change is already better than nondeterminism, that is, there is a sequence of regular languages  $(L_n)_{n \geq 3}$  accepted by  $n$ -state complete MCFAs with a single alternative transition with at most one mind-change such that any deterministic finite automaton requires at least  $2^{n+\log n-1}$  states, which implies that any nondeterministic finite automaton accepting these languages requires at least  $n + \log n - 1$  states. The bound on  $2^{n+\log n-1}$  states for deterministic finite automata is tight for complete MCFAs with a single alternative transition with at most one mind-change. Finally, for MCPDAs we find the following situation: the family of languages accepted by MCPDAs without mind-change characterizes the family of languages accepted by one-time nondeterministic pushdown automata, similarly as in the case of MCFAs, but for MCPDAs the language family under investigation is a proper superset of the family of deterministic context-free languages. Moreover, we prove that, from a descriptonal complexity point of view,  $k + 1$  mind-changes are better than  $k$  mind-changes for MCPDAs. To this end, we first show that the mirror language of  $L_{\text{mi}} = \{ ww^R \mid w \in \{a, b\}^+ \}$  is accepted by an MCPDA with a single mind-change but cannot be accepted by any MCPDA without a mind-change, since otherwise  $L_{\text{mi}}$  belongs to the union closure of deterministic context-free languages, which is known to be *not* the case [4]. Then considering the  $(k + 1)$ -times marked concatenation of the mirror language one can show that this language is a witness for the

strict inclusion of the family of languages accepted by MCPDAs with at most  $k$  mind-changes within the family of languages accepted by MCPDAs with at most  $k + 1$  mind-changes. Thus, MCPDAs form a strict and tight hierarchy dependent on the number of mind-changes within the family of context-free languages. Finally, we show that there is a context-free language that requires a non-constant number of mind-changes to be accepted by any MCPDA.

## 2. Definitions and Preliminaries

Let  $\Sigma^*$  denote the set of all words over the finite alphabet  $\Sigma$ . The *empty word* is denoted by  $\lambda$ , and  $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ . The *reversal* of a word  $w$  is denoted by  $w^R$ . For the *length* of  $w$  we write  $|w|$ . For the number of occurrences of a symbol  $a$  in  $w$  we use the notation  $|w|_a$ . Set inclusion is denoted by  $\subseteq$  and strict set inclusion by  $\subset$ . We write  $2^S$  for the power set and  $|S|$  for the cardinality of a set  $S$ .

We investigate *mind-changing* finite automata. The basic idea is that at the outset the automaton is partially deterministic. In this way, defined transitions constitute situations for which the automaton already has an opinion (on how to proceed), while undefined transitions constitute situations for which the automaton is still irresolute. Whenever the automaton encounters a situation for which it is irresolute, it can form its opinion by choosing an appropriate transition out of a set of transitions. The chosen transition is then added to the transition function. Finally, whenever the automaton is in a situation for which a transition is defined, it can change its mind and replace some transition already defined by an alternative matching transition from the set of transitions. In the sequel we will consider the total number of mind changes as a limited resource.

In order to define *mind-changing* finite automata formally, we recall some classical definitions.

A *nondeterministic finite automaton* (NFA) is a system  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ , where  $Q$  is the finite set of *internal states*,  $\Sigma$  is the finite set of *input symbols*,  $q_0 \in Q$  is the *initial state*,  $F \subseteq Q$  is the set of *accepting states*, and  $\delta: Q \times \Sigma \rightarrow 2^Q$  is the *transition function*. In the forthcoming we sometimes refer to  $\delta$  as a subset of  $Q \times \Sigma \times Q$ . A finite automaton  $M$  is *deterministic* (DFA) if and only if  $|\delta(q, a)| \leq 1$ , for all  $q \in Q$  and  $a \in \Sigma$ . In this case we simply write  $\delta(q, a) = q'$  for  $\delta(q, a) = \{q'\}$  assuming that the transition function is a (partial) mapping  $\delta: Q \times \Sigma \rightarrow Q$ .

Note that here NFAs as well as DFAs may have a partial transition function.

Next, the idea of mind-changing finite automata is implemented as follows: a *mind-changing finite automaton* (MCFA) is a system  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$ , where  $\langle Q, \Sigma, \delta_0, q_0, F \rangle$  is a DFA reflecting the *initial opinion* of  $M$ , and  $T_0 \subseteq Q \times \Sigma \times Q$  with  $T_0 \cap \delta_0 = \emptyset$  is the set of available *alternative transitions*.

A *configuration* of the MCFA  $M$  is a triple  $(q, w, \delta, T)$ , where  $q \in Q$  is the current state,  $w \in \Sigma^*$  is the still unread part of the input,  $\delta$  is the current transition function, and  $T$  is the current set of alternative transitions. The *initial configuration* for input  $w$  is set to  $(q_0, w, \delta_0, T_0)$ . During the course of its computation,  $M$  runs through a sequence of configurations. One step

from a configuration to its *successor configuration*, denoted by  $\vdash_M$ , is defined as follows. Let  $(q, aw, \delta, T)$  be a configuration for  $a \in \Sigma$ . Then we define:

1.  $(q, aw, \delta, T) \vdash_M (q', w, \delta, T)$  if  $\delta(q, a) = q'$  (*ordinary move*),
2.  $(q, aw, \delta, T) \vdash_M (q', w, \delta \cup \{(q, a, q')\}, T \setminus \{(q, a, q')\})$  if  $\delta(q, a)$  is undefined and  $(q, a, q') \in T$  (*form an opinion move*), and
3.  $(q, aw, \delta, T) \vdash_M (q', w, \delta', T')$  with  $\delta' = (\delta \cup \{(q, a, q')\}) \setminus \{(q, a, q'')\}$  and  $T' = (T \cup \{(q, a, q'')\}) \setminus \{(q, a, q')\}$ , if  $\delta(q, a) = q''$  and  $(q, a, q') \in T$  (*mind-changing move*).

As usual, the reflexive transitive closure of  $\vdash_M$  is denoted by  $\vdash_M^*$ . The subscript  $M$  will be dropped from  $\vdash_M$  and  $\vdash_M^*$  if the meaning is clear. Then the *language accepted* by the MCFA  $M$  with up to  $k$  mind changes, for  $k \geq 0$ , is defined as

$$L_k(M) = \{ w \in \Sigma^* \mid (q_0, w, \delta_0, T_0) \vdash_M^* (q, \lambda, \delta, T) \text{ for some } q \in F, \\ \text{using at most } k \text{ mind-changing moves in the computation} \}.$$

Observe, that by definition  $L_k(M) \subseteq L_{k+1}(M)$ , for  $k \geq 0$ .

In order to illustrate the definitions we continue with an example.

**Example 2.1** Consider the MCFA  $M = \langle \{1, 2, 3\}, \{a, b\}, \delta_0, 1, \{3\}, T_0 \rangle$  with the transition function  $\delta_0(1, a) = \delta_0(1, b) = 1$ , and  $\delta_0(2, a) = \delta_0(2, b) = 3$  and the set of alternative transitions  $T_0 = \{(1, a, 2)\}$ . The MCFA  $M$  is depicted in Figure 1. Obviously,  $L_0(M) = \emptyset$ , since the automaton can never change any transition and thus the sole accepting state 3 cannot be reached.

Whenever the MCFA  $M$  decides to make a mind-changing step, that is, exchanging the original transition  $(1, a, 1)$  by  $(1, a, 2)$  from  $T_0$ , then the sole accepting state 3 can be reached from 1 via state 2 by reading either  $aa$  or  $ab$ . Let us see how this works on input  $w = baab$ . To this end let  $\delta' = (\delta_0 \cup \{(1, a, 2)\}) \setminus \{(1, a, 1)\}$  and  $T' = (T_0 \cup \{(1, a, 1)\}) \setminus \{(1, a, 2)\}$ . Then an accepting computation on input  $w$  is

$$(q_0, w, \delta_0, T_0) = (1, baab, \delta_0, T_0) \vdash (1, aab, \delta_0, T_0) \vdash (1, ab, \delta_0, T_0) \vdash (2, b, \delta', T') \vdash (3, \lambda, \delta', T'),$$

where the sole mind-change appeared at the next to last computation step. Yet there is another computation on  $w$  which is not accepting, since the mind-change appeared too early and the computation blocks. This non-accepting computation is

$$(q_0, w, \delta_0, T_0) = (1, baab, \delta_0, T_0) \vdash (1, aab, \delta_0, T_0) \vdash (2, ab, \delta', T') \vdash (3, b, \delta', T').$$

It is worth mentioning, that although the underlying automata induced by  $\delta_0$  and  $\delta'$  are both deterministic, there are more than one computation on  $M$ , due to the mind-changes. By our example it is not hard to see that

$$L_1(M) = \{ w \in \{a, b\}^* \mid \text{the next to last letter of } w \text{ is an } a \}$$

and moreover  $L_k(M) = L_1(M)$ , for  $k \geq 1$ .

In case we consider the MCFA  $M' = \langle \{1, 2, 3\}, \{a, b\}, \delta', q_0, T' \rangle$ , then one observes that

$$L_0(M') = L_1(M') = b^*a(a + b)$$

and  $L_k(M') = \{w \in \{a, b\}^* \mid \text{the next to last letter of } w \text{ is an } a\}$ , for  $k \geq 2$ . ■

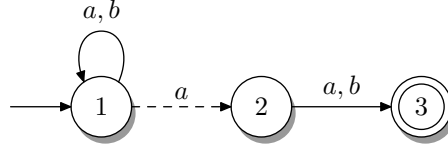


Figure 1: The MCFA  $M = \langle \{1, 2, 3\}, \{a, b\}, \delta_0, q_0, F, T_0 \rangle$ , where the transitions from  $\delta_0$  are drawn with solid arrows and that of  $T_0$  are depicted with dashed arrows.

### 3. A General Upper Bound

Intuitively, it is clear that the family of languages accepted by MCFAs coincides with the regular languages. Although the concept of mind changes does not improve the computational power of ordinary finite automata, the question for the descriptive complexity of such devices arises. Before we consider these costs (in terms of states) for simulations of MCFAs by ordinary finite automata in more detail, we recall and adapt some notation from [2]. Let  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$  be an MCFA. A (non)deterministic finite automaton  $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$  is *compatible* with  $M$  if and only if (i)  $Q' = Q$ , (ii)  $\Sigma' = \Sigma$ , (iii)  $q'_0 = q_0$ , (iv)  $F' = F$ , and (v)  $\delta'(q, a) \subseteq \delta_0(q, a) \cup \{p \mid (q, a, p) \in T_0\}$ , for every  $q \in Q$  and  $a \in \Sigma$ . If  $M'$  is compatible with  $M$ , then we write  $M' \prec M$ . We further define that  $M'$  is *non-empty compatible* with  $M$ , if and only if  $M'$  is compatible with  $M$  and  $\delta_0(q, a) \cup \{p \mid (q, a, p) \in T_0\} \neq \emptyset$  implies  $\delta'(q, a) \neq \emptyset$ , for every  $q \in Q$  and  $a \in \Sigma$ . If  $M'$  is non-empty compatible with  $M$ , then we write  $M' \prec_{ne} M$ . Obviously,  $M' \prec_{ne} M$  implies  $M' \prec M$ , but the converse implication does not hold in general.

It turns out that the upper bound on the costs for the simulations of an MCFA  $M$  by a DFA depends on the number of DFAs that are *non-empty compatible* with  $M$ , that is, on the cardinality of the set  $\mathcal{D}_{ne}(M) = \{M' \mid M' \text{ is a partial DFA with } M' \prec_{ne} M\}$ . This cardinality is equal to the *nondeterministic degree*  $d(M)$  of an MCFA  $M$  with state set  $Q$  and input alphabet  $\Sigma$  that is defined as

$$d(M) = \prod_{\substack{(q,a) \in Q \times \Sigma \\ |\delta_0(q,a) \cup \{p \mid (q,a,p) \in T_0\}| \neq 0}} |\delta_0(q, a) \cup \{p \mid (q, a, p) \in T_0\}|.$$

With this newly introduced notation our next theorem reads as follows:

**Theorem 3.1** *Let  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$  be an  $n$ -state MCFA. Then  $(k + 1) \cdot n \cdot d(M) + 1$  states are sufficient for an NFA to accept the language  $L_k(M)$ , for every  $k \geq 0$ .*

*Proof.* The computations of  $M$  can uniquely be split into up to  $k + 1$  sub-computations, where the sub-computations start at initial time and immediately after mind-changing moves. Every sub-computation is performed by a DFA that is non-empty compatible with  $M$ . The number of such DFAs is  $|\mathcal{D}_{ne}(M)| = d(M)$ .

Now an NFA  $M'$  that accepts  $L_k(M)$  is constructed as follows. For each DFA  $N$  from  $\mathcal{D}_{ne}(M)$  we use  $k + 1$  copies and denote them  $N_0, N_1, \dots, N_k$ . The indexes indicate how many mind-changing moves have been simulated already. So,  $M'$  starts in a new initial state  $q'_0$ , guesses one of the DFAs from  $\mathcal{D}_{ne}(M)$ , say DFA  $N$ , and starts the simulation of  $M$  with the copy  $N_0$ . The copy  $N_0$  is used for the first sub-computation. When the first mind-changing move is to be performed,  $M'$  simulates it and continues the computation with a copy of the DFA from  $\mathcal{D}_{ne}(M)$  that is obtained by exchanging the transition in  $N_0$  according to the mind change. The index of the copy of the new DFA is now 1. Note that the jump into the new DFA is a nondeterministic step of  $M'$  since it is a nondeterministic step in  $M$ . Similarly, the remaining sub-computations are simulated.

It is straightforward to see that  $M'$  accepts  $L_k(M)$ . The number of states of  $M'$  is calculated as one new initial state plus  $(k + 1) \cdot n$  states for the  $k + 1$  copies of each DFA from  $\mathcal{D}_{ne}(M)$ . This makes altogether no more than  $(k + 1) \cdot n \cdot d(M) + 1$  states.  $\square$

From Theorem 3.1 and the powerset construction on NFAs we deduce the following result. Note that we have partial DFAs and the new initial state of  $M'$  is never reached again. So, at least one state can be saved in the exponent.

**Corollary 3.2** *Let  $M$  be an  $n$ -state MCFA with input alphabet  $\Sigma$ . Then  $2^{(k+1) \cdot n \cdot d(M)} + 1$  states are sufficient for a DFA to accept the language  $L_k(M)$ , for every  $k \geq 0$ .*

So, the upper bound on the number of states of a finite automaton accepting  $L_k(M)$ , for an MCFA  $M$ , depends on the cardinality of  $\mathcal{D}_{ne}(M)$ .

**Lemma 3.3** *Let  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$  be an  $n$ -state MCFA. Then  $\mathcal{D}_{ne}(M)$  contains no more than  $n^{|\Sigma|^n}$  DFAs.*

Apart from the cardinality of  $\mathcal{D}_{ne}(M)$  the upper bound on the number of states of a finite automaton accepting  $L_k(M)$ , for an MCFA  $M$ , depends on the number of mind-changing moves allowed. Let us first consider the case  $k = 0$ . Although an MCFA  $M$  cannot do any mind-changing move, if  $k = 0$ , it still can perform moves that form an opinion. With these moves transitions from the current set  $T$  of alternative transitions can be chosen whenever the current transition function  $\delta$  is undefined. The chosen transitions can be used in the computation of  $M$  but cannot be changed afterwards anymore. This is exactly the same idea that underlies the concept of one-time nondeterminism, that was recently introduced in [2], and is formalized as follows. Let  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  be an NFA and

$$(q_0, a_0 a_1 \cdots a_{n-1}) \vdash_M (q_1, a_1 a_2 \cdots a_{n-1}) \vdash_M (q_2, a_2 a_3 \cdots a_{n-1}) \vdash_M \cdots \vdash_M (q_n, \lambda)$$

be a computation of  $M$  on input  $a_0 a_1 \cdots a_{n-1} \in \Sigma^+$ . A computation is *permissible* if and only if  $n = 1$  or  $(q_i, a_i) = (q_j, a_j)$  implies  $q_{i+1} = q_{j+1}$ , for all  $0 \leq i < j \leq n - 1$ . Now,  $M$  is said



to be *one-time nondeterministic* if and only if it may only perform permissible computations. In this case we call  $M$  a *one-time nondeterministic finite automaton* (OTNFA). A word  $w$  is permissible acceptable by  $M$  if there is a permissible computation that ends in an accepting state of  $M$ . The language accepted by an OTNFA  $M$  is

$$L_p(M) = \{ w \in \Sigma^* \mid \text{word } w \text{ is permissible acceptable by } M \}.$$

Then we find the following relation between MCFAs without mind-changes and one-time nondeterministic automata:

**Theorem 3.4** *For every MCFA  $M$  one can effectively construct an OTNFA  $M'$  with the same number of states such that  $L_p(M') = L_0(M)$  and vice versa.*

*Proof.* Assume that  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$  is an MCFA. The OTNFA  $M' = \langle Q, \Sigma, \delta, q_0, F \rangle$  is defined by its transition function

$$\delta(q, a) = \begin{cases} \{ \delta_0(q, a) \} & \text{if } \delta_0(q, a) \text{ is defined} \\ \{ q' \mid (q, a, q') \in T_0 \} & \text{otherwise.} \end{cases}$$

It is not hard to see that a computation of  $M$  without mind-changes induces a permissible computation of  $M'$  and *vice versa*. Thus,  $L_p(M') = L_0(M)$  as desired.

Now, let  $M' = \langle Q, \Sigma, \delta, q_0, F \rangle$  be an OTNFA. An MCFA  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$  is constructed as follows:  $\delta_0 = \delta \setminus \{ (q, a, q') \mid q, q' \in Q \text{ and } a \in \Sigma \text{ such that } |\delta(q, a)| \geq 2 \}$  and  $T_0 = \bigcup_{\substack{(q,a) \in Q \times \Sigma \\ |\delta(q,a)| \geq 2}} \delta(q, a)$ . That is, for all states with two or more  $a$ -transitions in  $M'$ , the transition function  $\delta_0$  in  $M$  on these states and letter  $a$  becomes undefined. Moreover, these  $a$ -transitions form the set  $T_0$  such that they can be used once in a move of  $M$  that forms an opinion. Thus,  $L_0(M) = L_p(M')$ .  $\square$

Now we can deduce upper and lower bounds on MCFA simulations from the corresponding bounds on OTNFAs. The definition of the nondeterministic degree  $d(M)$  of an MCFA  $M$  applies to NFAs as well. For NFAs the set  $T_0$  is always empty.

In [2] it was shown that every  $n$ -state OTNFA  $M$  can be simulated by a DFA with at most  $(n+1)^{d(M)}$  states. Moreover, there is a sequence of regular languages  $(L_n)_{n \geq 1}$  that are accepted by  $n$ -state OTNFAs  $M_n$  with  $d(M_n) = n$  which have a sole nondeterministic state, that is nondeterministic only for one input symbol. On the other hand, any DFA accepting a language  $L_p(M_n)$  requires at least  $(n+1)^n$  states. By the constructions in the proof of Theorem 3.4, these upper and lower bound results translate to MCFAs that do not make any mind-change move at all as follows.

**Corollary 3.5** *Let  $M$  be an  $n$ -state MCFA. Then  $(n+1)^{d(M)}$  states are sufficient for a DFA accepting the language  $L_0(M)$ . There is a sequence of regular languages  $(L_n)_{n \geq 1}$  such that  $L_n$  is accepted by an  $n$ -state MCFA  $M_n$  with  $d(M_n) = n$  that has a sole state on which a mind-change may occur and where all alternative transitions are on the same letter, and any DFA accepting  $L_0(M_n)$  requires at least  $(n+1)^n$  states.*

## 4. One Mind-Change on a Single Transition is Already Better Than Nondeterminism

We consider the case where the MCFA is complete. An MCFA  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$  is *complete* if the underlying DFA  $M' = \langle Q, \Sigma, \delta_0, q_0, F \rangle$  is complete, that is,  $|\delta_0(q, a)| = 1$ , for every  $q \in Q$  and  $a \in \Sigma$ . For  $k = 0$  it is obvious that we have  $L_0(M) = L(M')$ . Thus, in this case we do not save states when comparing MCFAs and DFAs. Next we investigate a non-trivial case on MCFAs, where the underlying DFA is complete,  $k = 1$ , and  $T_0$  is a singleton set. For these parameters we find the following situation.

**Theorem 4.1** *Let  $M = \langle Q, \Sigma, \delta_0, q_0, F, T_0 \rangle$  be an  $n$ -state complete MCFA with  $|T_0| = 1$ . Then  $2^{n+\log n-1}$  states are sufficient and necessary in the worst case for a DFA to accept the language  $L_1(M)$ .*

*Proof.* Let  $T_0 = \{(q_{\text{ndt}}, a_{\text{ndt}}, q'')\}$ . The MCFA  $M$  gives rise to two DFAs  $M_0$  and  $M_1$ . The DFA  $M_0$  is nothing other than the underlying DFA of  $M$ , while the automaton  $M_1$  is  $M_0$  modified by the single transition in  $T_0$ . Let the transition function of  $M_0$  be  $\delta_0$  and that of  $M_1$  is referred to as  $\delta_1$ . By definition,  $\delta_1 = (\delta_0 \cup \{(q_{\text{ndt}}, a_{\text{ndt}}, q'')\}) \setminus \{(q_{\text{ndt}}, a_{\text{ndt}}, q')\}$  and  $T_1 = \{(q_{\text{ndt}}, a_{\text{ndt}}, q')\}$  if  $\delta_0(q_{\text{ndt}}, a_{\text{ndt}}) = q'$ . Thus, the transition functions of  $M_0$  and  $M_1$  differ on state  $q_{\text{ndt}}$  and letter  $a_{\text{ndt}}$ .

We construct a DFA  $M' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$  with the state set  $Q' = Q \times 2^Q$ , the initial state  $q'_0 = (q_0, \emptyset)$ , the final states  $F' = (F \times 2^Q) \cup (Q \times \{P \subseteq Q \mid P \cap F \neq \emptyset\})$ , and the transition function  $\delta'$  defined as

$$\delta'((q, P), a) = \begin{cases} (\delta_0(q, a), \delta_1(\{q\} \cup P, a)) & \text{if } (q, a) = (q_{\text{ndt}}, a_{\text{ndt}}) \\ (\delta_0(q, a), \delta_1(P, a)) & \text{otherwise,} \end{cases}$$

where the transition function  $\delta_1$  is as usual extended to sets of states, and in both cases  $P \subseteq Q$ . Next we argue that  $L(M') = L_1(M)$ .

First we show  $L(M') \subseteq L_1(M)$ . Let  $w \in L(M')$ . If  $w$  is accepted in a state of from  $F \times 2^Q$ , then the computation induced by the first component of the states of  $M'$  gives a computation in the MCFA  $M$  without any mind-change. Thus, this computation in  $M$  is also accepting. Therefore,  $w \in L_1(M)$ . On the other hand, let  $w$  be accepted by  $M'$  by a computation

$$\delta'(q'_0, w) = \delta'((q_0, \emptyset), w) = (q, P)$$

where state  $(q, P)$  is in  $Q \times \{P \subseteq Q \mid P \cap F \neq \emptyset\}$ . Hence there is a  $p \in P$  with  $p \in F$  that can be traced back in the second component of the states in the above mentioned computation of  $M'$  to the state  $\delta_1(q_{\text{ndt}}, a_{\text{ndt}}) = q''$ , that is introduced by the  $a_{\text{ndt}}$ -transition from a state  $(q_{\text{ndt}}, P')$ , for some  $P' \subseteq Q$ . Thus, we have  $\delta_1(q'', v) = p$ , for some suffix  $v$  of  $w$ . In this way we can split the computation of  $M'$  on the word  $w$  into three parts  $w = ua_{\text{ndt}}v$  such that

1.  $\delta'(q'_0, u) = \delta'((q_0, \emptyset), u) = (q_{\text{ndt}}, P')$  with  $\delta_0(q_0, u) = q_{\text{ndt}}$ ,
2.  $\delta'((q_{\text{ndt}}, P'), a_{\text{ndt}}) = \delta_0(q_{\text{ndt}}, a_{\text{ndt}}) \times (\{\delta_1(q_{\text{ndt}}, a_{\text{ndt}})\} \cup \delta_1(P', a_{\text{ndt}})) = (q', \{q''\} \cup P'')$  with  $\delta_1(P', a_{\text{ndt}}) = P''$ , and

$$3. \delta'((q', \{q''\} \cup P''), v) = (q, \{p\} \cup P''') = (q, P) \text{ with } \delta_1(q'', v) = p \text{ and } \delta_1(P'', v) = P''''.$$

This induces a computation in  $M$  with one mind-change as follows:

$$\begin{aligned} (q_0, w, \delta_0, T_0) &= (q_0, ua_{\text{ndt}}v, \delta_0, T_0) \vdash^* (q_{\text{ndt}}, a_{\text{ndt}}v, \delta_0, T_0) \\ &\vdash (\delta_1(q_{\text{ndt}}, a_{\text{ndt}}), v, \delta_1, T_1) = (q'', v, \delta_1, T_1) \vdash^* (p, \lambda, \delta_1, T_1). \end{aligned}$$

Since  $p \in F$ , this computation accepts the word  $w$ . Thus, we have shown  $L(M') \subseteq L_1(M)$ .

The converse inclusion  $L_1(M) \subseteq L(M')$  is shown by similar arguments. Thus,  $L(M') = L_1(M)$  and the number of states of  $M'$  is at most  $n \cdot 2^n$ , which is equal to  $2^{n+\log n}$ .

Next we collapse some states in  $M'$  that are equivalent. Let  $q \in Q$  and  $P \subseteq Q$  with  $q \notin P$ . We show that the states  $(q, P)$  and  $(q, \{q\} \cup P)$  are equivalent and thus can be merged in  $M'$ . To this end we prove by induction that there is no word that distinguishes both states. Observe, that

$$(q, P) \in F' \quad \text{if and only if} \quad (q, \{q\} \cup P) \in F'.$$

Therefore, both states cannot be distinguished by the empty word. Next consider an arbitrary word  $w = av$  with  $a \in \Sigma$  and  $v \in \Sigma^*$ . We distinguish two cases:

1. If  $(q, a) = (q_{\text{ndt}}, a_{\text{ndt}})$ , then

$$\begin{aligned} \delta'((q_{\text{ndt}}, P), a_{\text{ndt}}) &= (\delta_0(q_{\text{ndt}}, a_{\text{ndt}}), \{\delta_1(q_{\text{ndt}}, a_{\text{ndt}})\} \cup \delta_1(P, a_{\text{ndt}})) \\ &= \delta'((q_{\text{ndt}}, \{q_{\text{ndt}}\} \cup P), a_{\text{ndt}}) \end{aligned}$$

and thus both states  $(q, P)$  and  $(q, \{q\} \cup P)$  cannot be distinguished by  $w = a_{\text{ndt}}v$ .

2. Otherwise, that is,  $(q, a) \neq (q_{\text{ndt}}, a_{\text{ndt}})$ , we argue as follows: recall that  $\delta_1(q, a) = \delta_0(q, a)$  in this case. Then we find

$$\delta'((q, P), a) = (\delta_0(q, a), \delta_1(P, a)) = (q', P'),$$

for some  $q' \in Q$  and  $P' \subseteq Q$ , and

$$\delta'((q, \{q\} \cup P), a) = (\delta_0(q, a), \{\delta_1(q, a)\} \cup \delta_1(P, a)) = (q', \{q'\} \cup P').$$

In case  $q' \in P'$ , both states  $(q', P')$  and  $(q', \{q'\} \cup P')$  are identical and therefore equivalent anyway. Otherwise, the induction hypothesis applies to the states  $(q', P')$  and  $(q', \{q'\} \cup P')$ . Thus, these states cannot be distinguished by any word, and in particular not by  $v$ . Hence, the original states  $(q, P)$  and  $(q, \{q\} \cup P)$  cannot be distinguished by  $w = av$  either.

By identifying all the equivalent states  $(q, P)$  and  $(q, \{q\} \cup P)$  in  $M'$  we end up with the state set

$$Q' = \{ (q, P) \mid q \in Q \text{ and } P \subseteq Q \setminus \{q\} \}$$

and an appropriately adapted transition function  $\delta_0$  and  $T_0$ . This reduces the number of states to  $n \cdot 2^{n-1}$ , which is equal to  $2^{n+\log n-1}$  as stated.

It remains to be shown that the bound  $2^{n+\log n-1}$  is tight. Define the  $n$ -state complete MCFA  $M = \langle Q, \{a, b, c\}, \delta_0, q_0, F, T_0 \rangle$  with the state set  $Q = \{0, 1, \dots, n-1\}$ , initial state  $q_0 = 0$ , set of final states  $F = \{n-1\}$ , the transition function  $\delta_0$  defined as

$$\delta_0(i, a) = \begin{cases} i+1 & \text{if } 0 \leq i < n-1 \\ 0 & \text{otherwise,} \end{cases} \quad \delta_0(i, c) = \begin{cases} 0 & \text{if } i = 0 \\ i+1 & \text{if } 1 \leq i < n-1 \\ 1 & \text{otherwise,} \end{cases}$$

$$\delta_0(i, b) = \begin{cases} 0 & \text{if } i = 0 \\ i+1 & \text{if } 1 \leq i < n-1 \\ n-1 & \text{otherwise,} \end{cases}$$

and  $T_0 = \{(0, b, 1)\}$ . The MCFA  $M$  is depicted in Figure 2. The equivalent DFA  $M'$  is constructed as above.

First we show that every state in  $Q' = \{(q, P) \mid q \in Q \text{ and } P \subseteq Q \setminus \{q\}\}$  is accessible from the initial state  $q'_0 = (0, \emptyset)$ . We split this proof into two cases:

1. Consider an arbitrary state of the form  $(0, P)$  in  $Q'$ . The reachability of these states in the DFA  $M'$  is shown by induction of the size of  $P$ . The state  $(0, \emptyset)$  is obviously reachable since it is the initial state of  $M'$ . Assume that all state in  $M'$  of the form  $(0, P)$  with  $|P| < k$  are reachable. Consider  $P$  with  $|P| = k$ . Let  $P = \{i_1, i_2, \dots, i_k\}$  such that  $1 \leq i_1 < i_2 < \dots < i_k \leq n-1$ . Then

$$\begin{aligned} \delta'((0, \{i_2 - i_1, i_3 - i_1, \dots, i_k - i_1\}), bc^{i_1-1}) \\ &= \delta'((0, \{1, i_2 - i_1 + 1, i_3 - i_1 + 1, \dots, i_k - i_1 + 1\}), c^{i_1-1}) \\ &= (0, \{i_1, i_2, i_3, \dots, i_k\}) \\ &= (0, P), \end{aligned}$$

where the state the computation started has  $k-1$  states in the second component. Thus, the induction hypothesis applies.

2. Next let  $(i, P)$  be in  $Q'$ ,  $i > 0$ , and assume  $P = \{i_1, i_2, \dots, i_k\}$ . Then  $\delta'((0, \emptyset), a^i) = (i, \emptyset)$  and in general  $\delta'((0, \{i_1 - i, i_2 - i, \dots, i_k - i\}), a^i) = (i, \{i_1, i_2, \dots, i_k\}) = (i, P)$ , where subtraction is meant with respect to modulo  $n$ , and the computation started with a state which first component is 0. Then the first case applies.

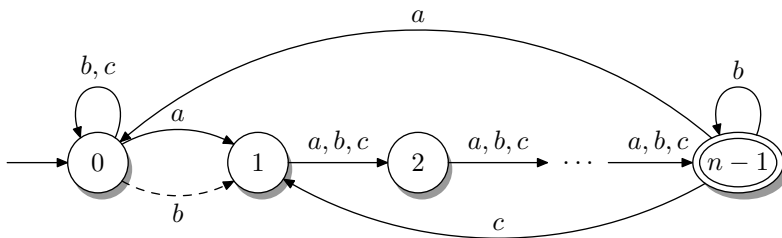


Figure 2: The  $n$ -state complete MCFA  $M = \langle \{0, 1, \dots, n-1\}, \{a, b, c\}, \delta_0, q_0, F, T_0 \rangle$  with a singleton set of alternative transitions  $T_0$  used in the lower bound proof such that any DFA accepting  $L_1(M)$  requires at least  $2^{n+\log n-1}$  states. The transitions from  $\delta_0$  are drawn with solid arrows and that of  $T_0$  are depicted with dashed arrows.

Thus, all states in  $M'$  are reachable.

Next we show that each state in  $M'$  defines a distinct equivalence class. Consider two distinct states  $(q, P)$  and  $(q', P')$  of  $M'$ . We distinguish two cases:

1. In case  $P \cup \{q\} \neq P' \cup \{q'\}$  we may assume without loss of generality that there is an  $i$  in  $(P \cup \{q\}) \setminus (P' \cup \{q'\})$ . Then  $\delta'((q, P), a^{n-i-1}) \in F'$ , while  $\delta'((q', P'), a^{n-i-1}) \notin F'$ .
2. If  $P \cup \{q\} = P' \cup \{q'\}$ , then by assumption we have  $q \neq q'$ . We may safely assume  $q < q'$ . Set  $q = i_0$  and  $q' = i'_0$ . Moreover, let  $P = \{i_1, i_2, \dots, i_k\}$  with  $0 \leq i_1 < i_2 < \dots < i_k \leq n-1$  and  $P' = \{i'_1, i'_2, \dots, i'_k\}$  with  $0 \leq i'_1 < i'_2 < \dots < i'_k \leq n-1$ . Here for  $i_0 = 0$  we find  $i_1 \geq 1$  and moreover  $i'_0 \neq 0$  and  $i'_1 = 0$ , since  $0 \in P \cup \{q\} = P' \cup \{q'\}$ . Then

$$\begin{aligned} \delta'((q, P), b) &= \delta'((0, \{i_1, i_2, \dots, i_k\}), b) \\ &= \begin{cases} (0, \{1\} \cup \{i_1 + 1, i_2 + 1, \dots, i_{k-1} + 1, n-1\}) & \text{if } i_k = n-1 \\ (0, \{1\} \cup \{i_1 + 1, i_2 + 1, \dots, i_k + 1\}) & \text{otherwise} \end{cases} \end{aligned}$$

and

$$\begin{aligned} \delta'((q', P'), b) &= \delta'((i'_0, \{0, i'_2, \dots, i'_k\}), b) \\ &= \begin{cases} (i''_0, \{1, i'_2 + 1, \dots, i'_{k-1} + 1, n-1\}) & \text{if } i_k = n-1 \\ (i''_0, \{1, i'_2 + 1, \dots, i'_{k-1} + 1, i'_k + 1\}) & \text{otherwise,} \end{cases} \end{aligned}$$

where  $i''_0 = i'_0 + 1$  if  $i'_0 \neq n-1$  and  $i''_0 = n-1$  if  $i'_0 = n-1$ . In both cases the reached states are inequivalent, because 0 is an element of the former two states (in the first component of the states), while 0 does not belong to the first or second component of the latter two states. Thus, we are back to the case above. Finally, if  $i_0 \neq 0$ , we have – again addition and subtraction are meant with respect to modulo  $n$  – the computations

$$\begin{aligned} \delta'((q, P), a^{n-i_0}) &= \delta'((i_0, \{i_1, i_2, \dots, i_k\}), a^{n-i_0}) \\ &= (i_0 + n - i_0, \{i_1 + n - i_0, i_2 + n - i_0, \dots, i_k + n - i_0\}) \\ &= (0, \{i_1 - i_0, i_2 - i_0, \dots, i_k - i_0\}) \end{aligned}$$

and

$$\begin{aligned} \delta'((q', P'), a^{n-i_0}) &= \delta'((i'_0, \{i'_1, i'_2, \dots, i'_k\}), a^{n-i_0}) \\ &= (i'_0 + n - i_0, \{i'_1 + n - i_0, i'_2 + n - i_0, \dots, i'_k + n - i_0\}) \\ &= (i'_0 - i_0, \{i'_1 - i_0, i'_2 - i_0, \dots, i'_k - i_0\}), \end{aligned}$$

which reduces this case to the previously analyzed one.

This shows that all states in  $M'$  define distinct equivalence classes, and therefore the DFA  $M'$  is minimal.  $\square$

This immediately gives us that mind-changing is better than nondeterminism from a descriptive complexity point of view.

**Theorem 4.2** *There is a sequence of regular languages  $(L_n)_{n \geq 3}$  such that  $L_n$  is accepted by an  $n$ -state complete MCFA  $M_n$  with a single alternative transition with at most one mind-change such that any NFA accepting  $L_1(M_n)$  requires at least  $n + \log n - 1$  states.*

*Proof.* Consider the sequence of languages  $(L_n)_{n \geq 0}$  induced by the  $n$ -state MCFAs in the lower bound argument from the proof of the previous theorem. Assume to the contrary that there are NFAs with strictly less than  $n + \log n - 1$  states accepting the languages  $L_n$ . By the conversion of these nondeterministic devices to equivalent DFAs we get at most an exponential blow up in the number of states by the powerset construction. If the NFAs we started from have strictly less than  $n + \log n - 1$  states the equivalent DFAs have at most  $2^{n+\log n-2}$  states, which contradicts the lower bound of states  $2^{n+\log n-1}$  on DFAs from the previous theorem. Hence any NFA accepting  $L_n$  requires at least  $n + \log n - 1$  states.  $\square$

## 5. Mind-Changing Pushdown Automata

This section is devoted to generalize the definition of mind-changing finite automata to pushdown automata and to obtain first results on these devices.

Let  $\Sigma$  be an alphabet. For convenience, we use  $\Sigma_\lambda$  for  $\Sigma \cup \{\lambda\}$ . A *nondeterministic pushdown automaton* (NPDA) is a system  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$ , where  $Q$  is a finite set of *internal states*,  $\Sigma$  is the finite set of *input symbols*,  $\Gamma$  is a finite set of *pushdown symbols*,  $\delta$  is a mapping from  $Q \times \Sigma_\lambda \times \Gamma$  to finite subsets of  $Q \times \Gamma^*$  called the *transition function*,  $q_0 \in Q$  is the *initial state*,  $\perp \in \Gamma$  is the so-called *bottom-of-pushdown symbol*, which initially appears on the pushdown store, and  $F \subseteq Q$  is the set of *accepting states*.

An NPDA is a *deterministic pushdown automaton* (DPDA), if there is at most one choice of action for any possible configuration. In particular, there must never be a choice of using an input symbol or of using  $\lambda$  input. Formally, a pushdown automaton  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$  is *deterministic* if (i)  $\delta(q, a, Z)$  contains at most one element, for all  $a$  in  $\Sigma_\lambda$ ,  $q$  in  $Q$ , and  $Z$  in  $\Gamma$ , and (ii) for all  $q$  in  $Q$  and  $Z$  in  $\Gamma$ : if  $\delta(q, \lambda, Z)$  is not empty, then  $\delta(q, a, Z)$  is empty for all  $a$  in  $\Sigma$ .

Now, *mind-changing pushdown automata* (MCPDA) are systems  $M = \langle Q, \Sigma, \Gamma, \delta_0, q_0, \perp, F, T_0 \rangle$ , where  $\langle Q, \Sigma, \Gamma, \delta_0, q_0, \perp, F \rangle$  is a DPDA reflecting the *initial opinion* of  $M$ , and the finite set  $T_0 \subseteq Q \times \Sigma_\lambda \times \Gamma \times Q \times \Gamma^*$  with  $T_0 \cap \delta_0 = \emptyset$  is the set of available *alternative transitions*.

A *configuration* of an MCPDA  $M$  is a quintuple  $(q, w, \gamma, \delta, T)$ , where  $q \in Q$  is the current state,  $w \in \Sigma^*$  is the still unread part of the input,  $\gamma$  the current content of the pushdown store, the leftmost symbol of  $\gamma$  being the top symbol,  $\delta$  is the current transition function, and  $T$  is the current set of alternative transitions. On input  $w$  the initial configuration is defined to be  $(q_0, w, \perp, \delta_0, T_0)$ . One step from a configuration to its *successor configuration*, denoted as before by  $\vdash_M$ , is defined as follows. Let  $(q, aw, Z\gamma, \delta, T)$  be a configuration for  $a \in \Sigma_\lambda$  and  $Z \in \Gamma$ . Then we define:

1.  $(q, aw, Z\gamma, \delta, T) \vdash_M (q', w, \beta\gamma, \delta, T)$  if  $\delta(q, a, Z) = (q', \beta)$  (*ordinary move*),
2.  $(q, aw, Z\gamma, \delta, T) \vdash_M (q', w, \beta\gamma, \delta \cup \{(q, a, Z, q', \beta)\}, T \setminus \{(q, a, Z, q', \beta)\})$  if  $\delta(q, a, Z)$  is undefined and  $(q, a, Z, q', \beta) \in T$  (*form an opinion move*), and
3.  $(q, aw, Z\gamma, \delta, T) \vdash_M (q', w, \beta\gamma, \delta', T')$  with  $\delta' = (\delta \cup \{(q, a, Z, q', \beta)\}) \setminus \{(q, a, Z, q'', \beta')\}$  and

$T' = (T \cup \{(q, a, Z, q'', \beta')\}) \setminus \{(q, a, Z, q', \beta)\}$ , if  $\delta(q, a, Z) = (q'', \beta')$  and  $(q, a, Z, q', \beta) \in T$  (*mind-changing move*).

In order to simplify matters, we require that during any computation the bottom-of-pushdown symbol appears only at the bottom of the pushdown store.

The *language accepted* by the MCPDA  $M$  with accepting states and up to  $k$  mind changes, for  $k \geq 0$ , is defined as

$$L_k(M) = \{w \in \Sigma^* \mid (q_0, w, \perp, \delta_0, T_0) \vdash_M^* (q, \lambda, \gamma, \delta, T) \text{ for some } q \in F, \\ \text{using at most } k \text{ mind-changing moves in the computation}\}.$$

The notion of one-time nondeterminism has been generalized to pushdown automata as well [2]. It turned out that one-time nondeterministic pushdown automata (OTNPDA) are strictly more powerful than DPDA but still strictly less powerful than NPDA. In particular, Theorem 3.4 can be adapted to pushdown automata straightforwardly. Let  $\mathcal{L}(X)$  denote the family of languages accepted by some device of type  $X$ , and moreover let  $\mathcal{L}(\text{MCPDA}_k)$  denote the family of languages accepted by MCPDAs with at most  $k \geq 0$  mind-changing moves. Then we conclude  $\mathcal{L}(\text{OTNPDA}) = \mathcal{L}(\text{MCPDA}_0)$ .

The family of languages accepted by OTNPDA coincides with the (finite) union closure of the deterministic context-free languages [2]. On the other hand, the context-free languages  $\{wcx \mid w, x \in \{a, b\}^*, w \neq x\}$  [7] and the mirror language  $L_{\text{mi}} = \{ww^R \mid w \in \{a, b\}^+\}$  [4] do not belong to the union closure of the deterministic context-free languages. Since Example 5.2 shows that only one mind-changing move is enough to accept the mirror language we obtain the following corollary.

**Corollary 5.1** *The family  $\mathcal{L}(\text{OTNPDA}) = \mathcal{L}(\text{MCPDA}_0)$  is strictly included in the family  $\mathcal{L}(\text{MCPDA}_1)$ .*

So, the first mind-changing move matters. Before we turn to the question whether this is true also for further mind-changing moves, we give the example already mentioned.

**Example 5.2** *The mirror language  $L_{\text{mi}} = \{ww^R \mid w \in \{a, b\}^+\}$  is accepted by some MCPDA with at most one mind-changing move. To this end, the classical construction to accept the language works fine. First all symbols read are pushed onto the pushdown store. A possible mind-changing move now switches the computation to the comparison mode, where the remaining input suffix is compared with the pushdown content. Finally, if prefix and suffix of the input match the computation halts accepting. ■*

The previous example is generalized in a straightforward manner as follows.

**Example 5.3** *Let  $k \geq 1$  be a constant and  $L$  be a language that is accepted by some MCPDA that may perform  $k$  mind-changing moves. Then the language  $L\#L_{\text{mi}}$  is accepted by an MCPDA with at most  $k + 1$  mind-changing moves, where  $\#$  is a new symbol.*

Some MCPDA  $M$  accepting  $L\#L_{mi}$  first simulates an acceptor for  $L$  with at most  $k$  mind-changing moves. If it accepts when the  $\#$  appears in the input, it continues to simulate the MCPDA of Example 5.2 in order to verify that the suffix belongs to  $L_{mi}$ . Altogether,  $M$  performs no more than  $k + 1$  mind-changing moves. ■

Now we are prepared to prove that every mind-changing move matters by showing an infinite and strict mind-changing hierarchy of language families strictly in between the deterministic and general context-free languages. The overall proof is by induction on the number of mind-changing moves. The base of the induction is already available from Corollary 5.1. Assume now that there is a language accepted by some MCPDA that may perform  $k$  mind-changing moves, but that is not accepted by any MCPDA that may perform no more than  $k - 1$  mind-changing moves. The next lemma together with Example 5.3 form the induction step.

**Lemma 5.4** *Let  $k \geq 1$  be a constant and  $L$  be a language that is accepted by some MCPDA that may perform  $k$  mind-changing moves, but that is not accepted by any MCPDA that may perform no more than  $k - 1$  mind-changing moves. Then the language  $L\#L_{mi}$  is not accepted by any MCPDA with at most  $k$  mind-changing moves, where  $\#$  is a new symbol.*

*Proof.* Contrarily, assume that there is an MCPDA  $M = \langle Q, \Sigma, \Gamma, \delta_0, q_0, \perp, F, T_0 \rangle$  such that  $L_k(M) = L\#L_{mi}$ .

The computations of  $M$  can uniquely be split into up to  $k + 1$  sub-computations, where the sub-computations start at initial time and immediately after mind-changing moves. Every sub-computation is performed by a DPDA that is non-empty compatible with  $M$ . The number of such DPDAs is finite. So, each accepting computation of  $M$  is performed with  $k + 1$  DPDAs, and there are only finitely many, say  $\ell \geq 1$ , sequences of  $k + 1$  such DPDAs. We fix these sequences and consider for each sequence  $i$  a separate MCPDA  $M_i$  that simulates  $M$  but blocks and rejects whenever the simulation tries to deviate from the fixed sequence  $i$ . So, we have  $L_k(M) = \bigcup_{1 \leq i \leq \ell} L_k(M_i) = L\#L_{mi}$ . The following modifications are done for all  $M_i$ ,  $1 \leq i \leq \ell$ .

First,  $M_i$  is modified to  $M'_i$  such that  $M'_i$  accepts only inputs that are accepted by  $M_i$  in computations which do not contain any mind-changing move upon and after reading the separating symbol  $\#$ . To this end,  $M'_i$  simulates  $M_i$  until the symbol  $\#$  appears in the input and, afterwards, continues the simulation without mind-changing moves, that is, *deterministically*. Finally,  $M'_i$  accepts if the simulation ends accepting. In this way, we have the inclusion  $L_k(M'_i) \subseteq L_k(M_i)$ .

Let  $\Sigma'$  be the alphabet of  $L$ . Now  $M'_i$  is modified to  $M''_i$  such that  $M''_i$  accepts only inputs that are accepted by  $M'_i$  and which belong to the regular language  $\Sigma'^*\#((ab)^+(ba)^+)^+$ . To this end,  $M''_i$  directly simulates  $M'_i$  and additionally a DFA in its finite control. So, in particular,  $M''_i$  works deterministically on input suffixes of the form  $\#\{a, b\}^*$ , and we have the inclusions  $L_k(M''_i) \subseteq L_k(M'_i) \subseteq L_k(M_i)$ .

Next, we are interested in the suffixes  $\#((ab)^+(ba)^+)^+$  of words accepted by  $M''_i$ . To this end, let  $\pi: L_k(M''_i) \rightarrow \#((ab)^+(ba)^+)^+$  be the projection that extracts these suffixes from accepted words, and let  $S_j = \{ \pi(w) \mid w \in L_k(M''_i) \text{ and } \pi(w) \in \#((ab)^+(ba)^+)^j \}$ , for  $j \geq 1$ . Assume that there are two numbers  $j_1, j_2 \geq 1$  with  $j_2 \geq 2j_1$  such that both sets  $S_{j_1}$  and  $S_{j_2}$  are infinite. Then  $M''_i$  is once more modified to  $M'''_i$  such that  $M'''_i$  accepts only inputs whose suffixes belong



to  $S_{j_2}$  and, moreover, whose suffixes have a prefix that belongs to  $S_{j_1}$ . To this end,  $M_i'''$  directly simulates  $M_i''$  and additionally a finite counter in its finite control. The counter starts with value 1 and is increased whenever  $aa$  appears in the input. Since  $M_i''$  works deterministically on the suffix, it can check whether it runs through an accepting situation while the counter value is  $j_1$  and whether the counter value is  $j_2$  at the end of an accepting computation.

Taking a closer look at the suffixes of words in  $L_k(M_i''')$  shows that they are of the form  $\#vv^Ruu^Rvv^R$ , where  $\#vv^R \in S_{j_1}$  and  $u \in (ba)^*((ab)^+(ba)^+)^{j_2/2-j_1}$ . So, the number of factors  $aa$  and  $bb$  in the suffixes are fixed. Now, a simple application of Ogden's lemma where, for example, the symbols of the last  $v^R$  are marked, shows that  $L_k(M_i''')$  is not even a context-free language. However,  $M_i'''$  is an MCPDA and, thus, accepts a context-free language. From the contradiction we obtain that, for any  $1 \leq i \leq \ell$ , there are no two numbers  $j_1, j_2 \geq 1$  with  $j_2 \geq 2j_1$  such that both sets  $S_{j_1}$  and  $S_{j_2}$  are infinite for  $M_i'''$  and, thus, for  $M_i'$ .

Since, for any  $j \geq 1$ , there are infinitely many suffixes  $\#((ab)^+(ba)^+)^j$  of words from  $L_k(M)$ , and there are infinitely many pairs  $j_1, j_2 \geq 1$  with  $j_2 \geq 2j_1$ , we conclude that there is a  $j_0 \geq 1$  such that  $S_{j_0}$  is finite for any  $L_k(M_i')$ ,  $1 \leq i \leq \ell$ . Therefore, we can choose some fixed suffix  $v_0 \in \#((ab)^+(ba)^+)^{j_0} \cap \#L_{\text{mi}}$  that does *not* belong to  $S_{j_0}$  for any  $L_k(M_i')$ . Since, for any word  $w$  from  $L$ , the concatenation  $wv_0$  belongs to  $L_k(M)$ , we conclude that in any accepting computation on any word  $wv_0$  automaton  $M$  performs a mind-changing move while processing  $v_0$ .

This allows to construct an MCPDA  $\hat{M}$  from  $M$  such that  $L_{k-1}(\hat{M}) = L \cdot v_0$ . To this end,  $\hat{M}$  simulates  $M$  with no more than  $k - 1$  mind-changing moves until the  $\#$  appears in the input. For the construction of the remaining computation a general concept of predicting machines (see [3]) can be used. Basically, the idea is to associate with pushdown symbols information on whether the remaining input (which is fixed) together with the current state and current pushdown content yields an accepting computation. Since the remaining computation of  $\hat{M}$  is performed by no more than two DPDA for which there are only finitely many possibilities, the concept can be applied.

The last step is to construct an MCPDA  $\tilde{M}$  from  $\hat{M}$  such that  $L_{k-1}(\tilde{M}) = L$ . To this end,  $\tilde{M}$  simulates  $\hat{M}$  with no more than  $k - 1$  mind-changing moves on inputs from  $\Sigma'^*$ . Again, since the remaining computation of  $\hat{M}$  is performed by a DPDA, the concept of predicting machines can be used to let  $\tilde{M}$  accept if and only if the input has been read and its extension by  $v_0$  would lead to an accepting computation of  $\hat{M}$ .

We conclude that  $\tilde{M}$  accepts  $L$  with no more than  $k - 1$  mind-changing moves. From the contradiction it follows that the assumption that  $M$  accepts  $L\#L_{\text{mi}}$  with at most  $k$  mind-changing moves is wrong, and the lemma follows.  $\square$

It is worth mentioning that formally we need  $k$  new symbols in order to apply Lemma 5.4  $k$  times. However, it is clear that only one symbol  $\#$  is sufficient. A simple counter maintained deterministically in the finite control uniquely identifies the  $i$ th occurrence of the symbol  $\#$  in the input. So, we have shown the following hierarchy.

**Theorem 5.5** *For all  $k \geq 0$ , the family  $\mathcal{L}(\text{MCPDA}_k)$  is strictly included in  $\mathcal{L}(\text{MCPDA}_{k+1})$ .*

The infinite and tight mind-changing hierarchy bounds the number of mind-changing moves by a constant for every family. This raises the natural question whether a constant number of mind-changing moves is sufficient to accept all context-free languages. In other words, is the family of context-free languages equal to  $\bigcup_{k \geq 0} \mathcal{L}(\text{MCPDA}_k)$ ? The answer is no.

**Theorem 5.6** *There is a context-free language not belonging to  $\bigcup_{k \geq 0} \mathcal{L}(\text{MCPDA}_k)$ .*

*Proof.* As witness we use the context-free language  $L = L_{\text{mi}}(\#L_{\text{mi}})^*$ . Assume that  $L$  belongs to the union  $\bigcup_{k \geq 0} \mathcal{L}(\text{MCPDA}_k)$ . Then there is some  $k_0$  such that  $L$  belongs to  $\mathcal{L}(\text{MCPDA}_{k_0})$ . Now let  $k' \geq k_0$  and consider the intersection of  $L$  and the regular language  $\{a, b\}^*(\#\{a, b\}^*)^{k'}$ . The intersection is  $L_{\text{mi}}(\#L_{\text{mi}})^{k'}$ , which is not accepted by any MCPDA with at most  $k'$  mind-changing moves by Lemma 5.4. Therefore, it is not accepted by any MCPDA with at most  $k_0$  mind-changing moves. Since the intersection trivially belongs to  $L$ , we obtain a contradiction that shows the theorem.  $\square$

In particular, the theorem reveals that there are context-free languages that require infinitely many mind-changing moves. Altogether, we have

$$\begin{aligned} \mathcal{L}(\text{DPDA}) \subset \mathcal{L}(\text{MCPDA}_0) \subset \cdots \subset \mathcal{L}(\text{MCPDA}_k) \subset \mathcal{L}(\text{MCPDA}_{k+1}) \subset \cdots \\ \subset \bigcup_{k \geq 0} \mathcal{L}(\text{MCPDA}_k) \subset \mathcal{L}(\text{NPDA}). \end{aligned}$$

## References

- [1] N. CHOMSKY, *Reflections on Language*. Pantheon, 1975.
- [2] M. HOLZER, M. KUTRIB, One-time nondeterministic computations. In: G. PIGHIZZINI, C. CÂMPEANU (eds.), *Descriptive Complexity of Formal Systems (DCFS 2017)*. Lecture Notes in Computer Science 10316, Springer, 2017, 177–188.
- [3] J. E. HOPCROFT, J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [4] M. KUTRIB, A. MALCHER, Context-dependent nondeterminism for pushdown automata. *Theoretical Computer Science* 376 (2007), 101–111.
- [5] A. M. TURING, On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2* (1937) 42, 230–265.  
<https://doi.org/10.1112/plms/s2-42.1.230>
- [6] A. M. TURING, On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proceedings of the London Mathematical Society, Series 2* (1938) 43, 544–546.  
<https://doi.org/10.1112/plms/s2-43.6.544>
- [7] D. WOTSCHKE, The Boolean closures of the deterministic and nondeterministic context-free languages. In: W. BRAUER (ed.), *GI Jahrestagung*. Lecture Notes in Computer Science 1, Springer, 1973, 113–121.  
[https://doi.org/10.1007/978-3-662-41148-3\\_11](https://doi.org/10.1007/978-3-662-41148-3_11)

# FORBIDDEN PATTERNS FOR ORDERED AUTOMATA

Ondřej Klíma<sup>(A)</sup>      Libor Polák<sup>(A)</sup>

Department of Mathematics and Statistics, Masaryk University,  
Kotlářská 2, 611 37 Brno, Czech Republic  
{klima,polak}@math.muni.cz

## **Abstract**

*The contribution concerns decision procedures in the algebraic theory of regular languages. Among others, various versions of forbidden patterns or configurations in automata are treated in the existing literature. Basically, one looks for certain subgraphs of the minimal automaton of a given language to decide whether this language does not belong to a given significant class of regular languages. We survey numerous known examples and we build a general theory covering the most of familiar ones. The chosen formalism differs from existing ones and the generalization to ordered automata enables us to reformulate some of known examples in a uniform shape. We also describe certain sufficient assumptions on the forbidden pattern which ensure that the corresponding class of languages forms a robust class in the sense of natural closure properties.*

## **1. Introduction**

Certain significant classes of regular languages can be characterized by some kind of forbidden patterns, which cannot occur in an automaton recognizing the language. To recall some examples, we can mention results by Cohen, Perrin and Pin [3] concerning the restriction of linear temporal logic obtained by considering only the operators “next” and “eventually”. The useful characterization obtained in that paper is that a language  $L$  is expressible by this logic, denoted by RTL, if and only if the minimal automaton of  $L$  does not contain the pattern from Figure 1. This characterization gave a polynomial time algorithm for testing whether the language recognized by an  $n$ -state deterministic automaton is RTL-definable (see Theorem 4.2 and its Corollary 4.3 in [3]). The technique of forbidden patterns was also used by Schmitz et al. [4, 14, 15] for the first levels of the Straubing-Thérien hierarchy of the star-free languages.

This paper is focusing on formal theory of forbidden patterns for deterministic finite automata, for which early formalisms were given in [3, 14]. For the purpose of this paper, the basic notion is a *semiautomaton* which is a deterministic automaton without initial and final states being specified. Then a pattern is an (incomplete) semiautomaton over an auxiliary alphabet  $X$  with

---

<sup>(A)</sup>Both authors were supported by Czech Science Foundation under Grant No. GA15-02862S.

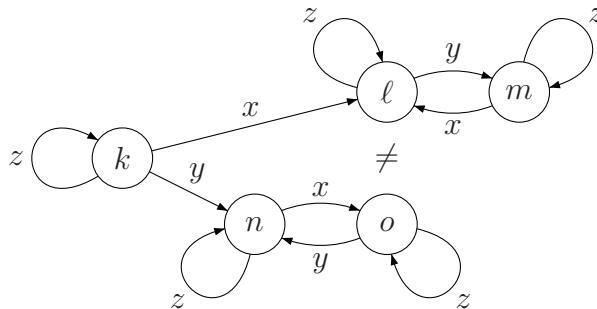


Figure 1: The forbidden pattern for RTL.

a marked pair of states. An example is on Figure 1 where the auxiliary alphabet is  $X = \{x, y, z\}$  and the pair of states is  $(\ell, o)$ . Now, one can consider the class of all regular languages for which there exist DFA recognizing the language, such that the automaton does not contain a pattern where a marked pair of states are different states. In the existing literature, for example, in [5, 7, 10], the approach of forbidden patterns is considered in various modifications: one state of the marked pair being final and the second one non-final, considering complete or incomplete automata, considering the minimal automaton or arbitrary automata, etc. We would like to develop a unified theory of forbidden patterns which would explain some general behaviour of these patterns and the classes defined by them and compare the formalisms with those of numerous known mentioned applications.

Comparing to a notion of forbidden patterns from [3, 14] where patterns are viewed as *subgraphs* of the underlining graph of an automaton under the consideration, in our formalism we map a pattern into that automaton by a homomorphism of semiautomata. Moreover, we consider a certain generalization which reflects one of recent directions of the research in algebraic theory of regular languages devoted to generalizations of the Eilenberg correspondence. Clearly, not all natural classes of regular languages are varieties. In particular, it is the case of some classes studied in papers, where forbidden patterns were applied successfully. Here, we use a combination of three ideas extending Eilenberg correspondence. Pin's modification [11] to positive varieties of languages (classes need not to be closed under complementation) can be combined with Straubing's modification [18] to  $\mathcal{C}$ -varieties (classes are closed only under preimages in homomorphisms from a fixed category  $\mathcal{C}$  of homomorphisms). In the mentioned papers, the corresponding classes of algebraic structures were syntactic ordered monoids and syntactic homomorphisms, respectively. Of course, in our paper, we deal with automata, thus we need to use Eilenberg type correspondence between positive  $\mathcal{C}$ -varieties of regular languages and  $\mathcal{C}$ -varieties of ordered semiautomata studied by the authors in [8]. The last notion is a modification of  $\mathcal{C}$ -varieties of semiautomata introduced in Chaubard et al [2] as  $\mathcal{C}$ -varieties of actions.

In this contribution, we show that every pattern in our formalism satisfying certain assumptions defines a  $\mathcal{C}$ -variety of ordered semiautomata. Then, we explain that many examples of forbidden patterns from the literature are particular instances of our general concept. We also explain how a certain special scheme of patterns working with final states in the minimal automaton of a language can be translated into a pattern for the minimal ordered automaton of the language.

Finally, we show how the well-known characterization of languages from the level 3/2 in the Straubing-Thérien hierarchy of star-free languages can be expressed naturally using our notion.

Due to the space limitations, omitted proofs can be found in the full version of this paper [9].

## 2. $\mathcal{C}$ -Varieties of Ordered Semiautomata

The aim of this section is to overview a necessary minimum concerning Eilenberg type correspondence between positive  $\mathcal{C}$ -varieties of languages and  $\mathcal{C}$ -varieties of ordered semiautomata. This is a framework in which the notion of forbidden patterns is developed. Note that (positive) varieties can be seen as a special case, if  $\mathcal{C}$  is the class of all homomorphisms.

We consider only regular languages in this contribution. The *quotient* of a language  $L \subseteq A^*$  by words  $u, v \in A^*$  is the set  $u^{-1}Lv^{-1} = \{w \in A^* \mid u w v \in L\}$ . In particular, *left quotients* are  $u^{-1}L = \{w \in A^* \mid u w \in L\}$ ,  $u \in A^*$ . The empty word is denoted by  $\lambda$ .

To recall a definition of positive  $\mathcal{C}$ -varieties of languages from [18], we first need to explain a role of a *category of homomorphisms*  $\mathcal{C}$ . From the point of view of category theory, this  $\mathcal{C}$  is a category where objects are free monoids  $A^*$  for a non-empty finite alphabet  $A$  and morphisms are certain monoid homomorphisms among them. We simplify the notation to consider  $\mathcal{C}$  as a class of homomorphisms satisfying the following properties:

- For each finite alphabet  $A$ , the identity mapping  $\text{id}_A : A^* \rightarrow A^*$  belongs to  $\mathcal{C}$ .
- If  $f : B^* \rightarrow A^*$  and  $g : C^* \rightarrow B^*$  belong  $\mathcal{C}$ , then the composition  $g f : C^* \rightarrow A^*$  belongs to  $\mathcal{C}$ .

Examples of categories  $\mathcal{C}$  which are used in this setting are:  $\mathcal{C}_{\text{all}}$  consisting of all homomorphisms between free monoids,  $\mathcal{C}_i$  consisting of all injective homomorphisms,  $\mathcal{C}_{\text{ne}}$  consisting of all non-erasing homomorphisms (here only  $\lambda$  is mapped onto  $\lambda$ ). Furthermore, by the *preimage* in  $f : B^* \rightarrow A^*$  of a given  $L \subseteq A^*$ , the set  $f^{-1}(L) = \{v \in B^* \mid f(v) \in L\}$  is meant.

A *positive  $\mathcal{C}$ -variety of languages*  $\mathcal{V}$  associates to every non-empty finite alphabet  $A$  a class  $\mathcal{V}(A)$  of regular languages over  $A$  in such a way that

- $\mathcal{V}(A)$  is closed under quotients, finite unions and intersections and contains  $\emptyset, A^*$ ,
- $\mathcal{V}$  is closed under preimages in morphisms from  $\mathcal{C}$ .

As we already mentioned, if we take  $\mathcal{C} = \mathcal{C}_{\text{all}}$ , we get exactly the notion of the positive varieties of languages. When adding “each  $\mathcal{V}(A)$  is closed under complements”, we get exactly the notion of the  *$\mathcal{C}$ -varieties* of languages.

To introduce a notion of ordered automata, we explain first that the minimal DFA of a given language is implicitly ordered. Indeed, for a minimal DFA, one can assign to each state  $q$  its *future*  $F_q$  consisting of all words which are acceptable if  $q$  would be the initial state. The minimality implies that different states have different futures. Now, if we identify states with their futures, then the relation  $\subseteq$  is an order on the minimal automaton (which is compatible with every action by a single letter, as we explain latter). We prefer to fix this minimal

(ordered) automaton of a given language, under the name canonical (ordered) automaton, using the construction of Brzozowski [1]: the *canonical automaton* of a regular language  $L$  is  $\mathcal{D}_L = (D_L, A, \cdot, L, F_L)$ , where  $D_L = \{u^{-1}L \mid u \in A^*\}$ ,  $q \cdot a = a^{-1}q$ , for each  $q \in D_L$ ,  $a \in A$ , and  $F_L = \{q \in D_L \mid \lambda \in q\}$ . Then a canonical ordered automaton is  $\mathcal{O}_L = (D_L, A, \cdot, L, F_L, \subseteq)$ .

Before we introduce a notion of ordered semiautomaton, we recall some basic terminology from the theory of ordered sets. By an order  $\leq$  on a set  $M$  we mean a reflexive, antisymmetric and transitive relation. A subset  $X$  of  $M$  is called *upward closed* if for every pair of elements  $x, y \in M$ , we have that  $x \leq y$ ,  $x \in X$  implies  $y \in X$ . A mapping  $f : M \rightarrow N$  between two ordered sets  $(M, \leq)$  and  $(N, \leq)$  is called *isotone* if  $x \leq y$  implies  $f(x) \leq f(y)$  for every pair of elements  $x, y \in M$ . For example, the action by each letter  $a \in A$  in  $\mathcal{O}_L$  is an isotone mapping and the set  $F_L$  of all final states is an upward closed subset with respect to  $\subseteq$ .

A *semiautomaton* is a triple  $\mathcal{A} = (Q, A, \cdot)$ , where  $Q$  is a finite set of states,  $A$  is a finite alphabet and  $\cdot : Q \times A \rightarrow Q$  is a complete transition function. Sometimes we also allow the function  $\cdot$  be a partial function and then we talk about a *partial semiautomaton*. Furthermore, an *ordered semiautomaton* is  $\mathcal{A} = (Q, A, \cdot, \leq)$ , where  $(Q, A, \cdot)$  is a semiautomaton,  $(Q, \leq)$  is an ordered set and for every pair of states  $p \leq q$  and a letter  $a \in A$  we have  $p \cdot a \leq q \cdot a$ . Note that an ordered semiautomaton has, with the exception of Subsection 5.6, a complete transition function in our paper. An example of an ordered semiautomaton can be obtained if we omit the initial and final states in the canonical ordered automaton, i.e., considering  $\overline{\mathcal{O}_L} = (D_L, A, \cdot, \subseteq)$  which is called *canonical ordered semiautomaton*.

In all cases, the transition function can be extended to a mapping  $\cdot : Q \times A^* \rightarrow Q$  in a usual way. Since a composition of isotone mappings is isotone, it follows that the action by every word is an isotone mapping from  $Q$  into itself. The definitions of the recognition of the language is usual, it is enough to assign just an initial state  $i$  and a subset of final states  $F$ . The only difference in the case of ordered automata is that  $F$  must be an upward closed subset.

To get Eilenberg type correspondence between positive  $\mathcal{C}$ -varieties of languages and corresponding classes of ordered semiautomata, we need to introduce required closure properties on classes of ordered semiautomata. Here we give an informal explanation; for missing technical details we refer to [8]. Each quotient of a language  $L$  can be recognized by the same automaton as  $L$  if we change initial and final states in an appropriate way. Therefore this closure property is covered by the fact that we consider semiautomata instead of automata. Since positive  $\mathcal{C}$ -varieties of languages are closed under taking finite unions and intersections, we consider a notion of direct products of ordered semiautomata which is used for recognition of such languages.

Now, we recall a construction on semiautomata which recognizes the preimage of a language in a given homomorphism. Let  $f : B^* \rightarrow A^*$  be an arbitrary homomorphism and  $\mathcal{A} = (Q, A, \cdot, \leq)$  be an ordered semiautomaton. Then *f-renaming* of  $\mathcal{A}$  is  $\mathcal{A}^f = (Q, B, \cdot^f, \leq)$  where  $q \cdot^f b = q \cdot f(b)$ , for every  $q \in Q$  and  $b \in B$ . We use the same notation also for the semiautomaton  $\mathcal{A} = (Q, A, \cdot)$ .

To get an Eilenberg type correspondence, two different classes of semiautomata cannot recognize the same class of languages. Thus the natural idea is to complete a considered class by all possible semiautomata recognizing the same languages. Such semiautomata can be obtained using disjoint union, homomorphic image and subsemiautomata. The first one is easily

understandable. Secondly, a homomorphism of ordered semiautomata and subsemiautomaton are standard notions if semiautomata are viewed as structures with unary operations given by letters and one binary relation. Now we are ready to define the basic notion from [8].

A  $\mathcal{C}$ -variety of ordered semiautomata  $\mathbb{V}$  associates to every non-empty finite alphabet  $A$  a class  $\mathbb{V}(A)$  of ordered semiautomata over alphabet  $A$  in such a way that

- a one-element semiautomaton over  $A$  is in  $\mathbb{V}(A)$  and this class is closed under disjoint unions and direct products of pairs, subsemiautomata and homomorphic images,
- for each non-empty finite alphabet  $B$  and  $f \in \mathcal{C}$ ,  $f : B^* \rightarrow A^*$  and  $\mathcal{A} \in \mathbb{V}(A)$ , we have  $\mathcal{A}^f \in \mathbb{V}(B)$  (we say that  $\mathbb{V}$  is closed under  $\mathcal{C}$ -renaming).

For each  $\mathcal{C}$ -variety of ordered semiautomata  $\mathbb{V}$ , we denote by  $\alpha(\mathbb{V})$  the class of regular languages recognized by these semiautomata, equivalently  $(\alpha(\mathbb{V}))(A) = \{ L \subseteq A^* \mid \overline{\mathcal{O}_L} \in \mathbb{V}(A) \}$ . For each positive  $\mathcal{C}$ -variety of regular languages  $\mathcal{L}$ , we denote by  $\beta(\mathcal{L})$  the  $\mathcal{C}$ -variety of ordered automata generated by all ordered semiautomata  $\overline{\mathcal{O}_L}$ , such that  $L \in \mathcal{L}(A)$  for some alphabet  $A$ .

**Proposition 2.1** ([8]) *The mappings  $\alpha$  and  $\beta$  are mutually inverse isomorphisms between the lattice of all  $\mathcal{C}$ -varieties of ordered semiautomata and the lattice of all positive  $\mathcal{C}$ -varieties of regular languages.*

### 3. Satisfying Configurations

For better understanding of the behaviour of forbidden patterns, we first introduce a notion, in which the results can be formulated in clearer way. At first, by a *family of substitutions*  $\mathcal{E}$  over a set  $X$ , we mean a system of classes  $\mathcal{E}_A$ , where, for each non-empty finite alphabet  $A$ , the class  $\mathcal{E}_A$  is formed by a set of homomorphisms from  $X^*$  to  $A^*$ .

**Definition 3.1** *A configuration  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{E})$  consists of a finite partial semiautomaton  $\mathcal{G} = (V, X, \cdot)$  over a set of variables  $X$ , states  $k, \ell \in V$  and a family of substitutions  $\mathcal{E}$  over  $X$ .*

One can simplify the notation to consider that  $\mathcal{E}$  is a category of homomorphism  $\mathcal{C}$  in the sense of Section 2. In this case,  $\mathcal{C}$  is closed under compositions of homomorphisms. The following definition enables a more general concept of families  $\mathcal{E}$  connected to  $\mathcal{C}$ -varieties. We say that  $\mathcal{E}$  is *closed under extensions* by  $\mathcal{C}$ , if for every  $g \in \mathcal{E}_B$  and  $f \in \mathcal{C}$ ,  $f : B^* \rightarrow A^*$ , we have  $gf \in \mathcal{E}_A$ .

As an example of  $\mathcal{E}$ , we can take homomorphisms with constant content, which means that  $g \in \mathcal{E}_A$  if and only if there is  $C \subseteq A$  such that  $g(x)$ , for each  $x \in X$ , contains exactly the letters from  $C$ . Then this family  $\mathcal{E}$  is closed under extensions by  $\mathcal{C}_{\text{all}}$ .

**Definition 3.2** *We say that an ordered semiautomaton  $\mathcal{A} = (Q, A, \cdot, \leq)$  satisfies the configuration  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{E})$  if, for each homomorphism  $g \in \mathcal{E}_A$  and each homomorphism  $\varphi : \mathcal{G} \rightarrow (Q, A, \cdot)^g$  of the partial semiautomaton  $\mathcal{G}$  into the semiautomaton  $(Q, A, \cdot)^g$ , it holds that  $\varphi(k) \leq \varphi(\ell)$ . We denote by  $\mathbb{K}(A)$  the class of all ordered semiautomata over  $A$  satisfying  $\mathcal{K}$ .*



For  $\mathcal{G} = (V, X, \cdot)$ , the fact that  $\varphi : \mathcal{G} \rightarrow (Q, A, \cdot)^g$  is a homomorphism of semiautomata can be expressed equivalently as  $\varphi(m) \cdot g(x) = \varphi(m \cdot x)$ , where  $m \in V$  and  $x \in X$  are arbitrary such that  $m \cdot x$  is defined in  $\mathcal{G}$ . Alternatively, we say that  $\varphi$  and  $g$  are *compatible*, in this case.

**Proposition 3.3** *Let  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{E})$  be an arbitrary configuration and let  $A$  be a non-empty finite set. Then the following hold.*

- (i)  $\mathbb{K}(A)$  contains the one-element semiautomaton.
- (ii) If  $\mathcal{G}$  is connected, then  $\mathbb{K}(A)$  is closed with respect to disjoint unions.
- (iii)  $\mathbb{K}(A)$  is closed with respect to subsemiautomata.
- (iv)  $\mathbb{K}(A)$  is closed with respect to products of pairs.
- (v) If  $\mathcal{E}$  is closed under extensions by  $\mathcal{C}$ , then  $\mathbb{K}(A)$  is closed with respect to  $\mathcal{C}$ -renaming.

*Proof.* See [9], the full version of this paper. □

Note that  $\mathbb{K}(A)$  is not closed, in general, with respect to homomorphic images as Example 3.4 suggests.

**Example 3.4** *Let  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{C}_{\text{all}})$  be given by Figure 2 (a). We claim that  $\mathbb{K}(A)$  contains the ordered semiautomaton given by Figure 2 (b) ordered by the equality. One can show that identifying  $p$  and  $p'$ ,  $q$  and  $q'$ ,  $r$  and  $r'$ , one gets an ordered semiautomaton outside  $\mathbb{K}(A)$ . The details can be found in the full version of this paper.*

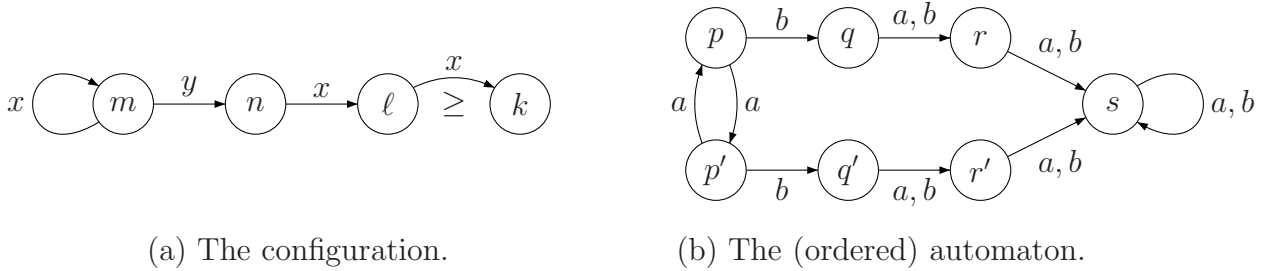


Figure 2: An example where  $\mathbb{K}(A)$  is not closed with respect to homomorphic images.

Let  $\mathbb{V}$  be a *system of ordered semiautomata*, i.e., for every alphabet  $A$ , a class  $\mathbb{V}(A)$  of ordered semiautomata over  $A$  is given. We consider

$$(\mathcal{L}(\mathbb{V}))(A) = \{ L \subseteq A^* \mid \exists \mathcal{A} \in \mathbb{V}(A), \mathcal{A} \text{ recognizes } L \},$$

the classes of languages recognized by the ordered semiautomata from  $\mathbb{V}$ . Since the minimization of an automaton takes a homomorphic image of a subautomaton, for each  $\mathbb{V}$  which is closed on subsemiautomata, we have  $(\mathcal{L}(\mathbb{V}))(A) = \{ L \subseteq A^* \mid \overline{\mathcal{O}_L} \in \mathbf{H}(\mathbb{V}(A)) \}$ , where  $\mathbf{H}(\mathbb{V}(A))$  consists of all homomorphic images of ordered semiautomata from  $\mathbb{V}(A)$ . By Proposition 3.3 (iii), this is the case for each  $\mathbb{K}$  given by a configuration  $\mathcal{K}$ . Moreover, having  $\mathbb{K}(A)$  closed under homomorphic images, it leads to an effective procedure for deciding whether  $L \in (\mathcal{L}(\mathbb{K}))(A)$ .

Using Lemma 17 from [8], which proves expected property concerning semi-commutation of the closure operators with  $\mathbf{H}$ , we get that  $\mathbf{H}(\mathbb{V})$  is a  $\mathcal{C}$ -variety of ordered semiautomata whenever  $\mathbb{V}$



is closed under subsemiautomata, finite products, disjoint unions and  $\mathcal{C}$ -renaming. As Proposition 3.3 and Example 3.4 suggest, the satisfiability of a configuration in the homomorphic image is the most problematic property in our approach.

**Definition 3.5** *The configuration  $\mathcal{K}$  is  $\mathbf{H}$ -invariant if it is satisfied in every homomorphic image of an ordered semiautomaton in which it is satisfied.*

Now we can formulate the central statement which is a direct consequence of Proposition 3.3.

**Theorem 3.6** *Let  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{E})$  be an  $\mathbf{H}$ -invariant configuration such that  $\mathcal{G}$  is connected and  $\mathcal{E}$  is closed under extensions by  $\mathcal{C}$ . Then the class of all ordered semiautomata satisfying  $\mathcal{K}$  forms a  $\mathcal{C}$ -variety of ordered semiautomata.*

To fulfill our program we need some robust class of  $\mathbf{H}$ -invariant configurations.

**Definition 3.7** *A partial semiautomaton  $\mathcal{G} = (V, X, \cdot)$  is acyclic if every cycle in  $\mathcal{G}$  is a loop. Furthermore,  $\mathcal{G}$  is simple if it is acyclic and there is a state  $n_0 \in V$ , which is called a root, such that for every  $n \in V$ ,  $n \neq n_0$ , there is exactly one simple path in  $\mathcal{G}$  from  $n_0$  to  $n$  and no path from  $n$  to  $n_0$ . We say that  $\mathcal{G}$  is balanced if the following two conditions are satisfied: (i) for each  $x, y, z \in X$  and  $n, n', m, m' \in V$  such that  $n \cdot x = n' = n' \cdot y$  and  $m \cdot x = m' = m' \cdot z$  we have  $y = z$ ; (ii) for each  $x, y \in X$  and  $n, n', m, m' \in V$  such that  $n \cdot x = n' = n' \cdot y$  and  $m \cdot x = m'$  we have  $m' \cdot y = m'$ .*

Note that, in a balanced simple semiautomaton, there are not two loops around a single state labeled by different letters. Also we could mention that the semiautomaton on Figure 2 (a) is not balanced, because it does not satisfy the condition (ii) from Definition 3.7.

**Proposition 3.8** *Let  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{C}_{\text{all}})$  be a configuration, where the partial semiautomaton  $\mathcal{G}$  is simple and balanced. Then  $\mathcal{K}$  is  $\mathbf{H}$ -invariant.*

*Proof.* See [9], the full version of this paper. □

## 4. Forbidden Patterns

In the literature, one meets various kinds of the so-called (forbidden) patterns. The main goal of this contribution is to analyze these concepts by the notion from the previous section. In the definition of configuration certain  $\mathcal{E}$  occurs, however for the comparison to the existing examples in the literature this general concept is not needed. So, we put  $\mathcal{E} = \mathcal{C}_{\text{all}}$  for the purpose of this section. Alternatively, when one considers languages without the empty word, one takes also all corresponding homomorphisms, that is  $\mathcal{E} = \mathcal{C}_{\text{ne}}$ .

We start by explaining the basic variant of forbidden pattern which is just the simple reformulation of the notion of configuration.

**Definition 4.1 (Variant 1)** A pattern  $\mathcal{P} = [\mathcal{G}, k \not\leq \ell]$  consists of a partial semiautomaton  $\mathcal{G} = (V, X, \cdot)$  and a pair of states  $k, \ell \in V$ . We say that  $\mathcal{P}$  is present in an ordered semiautomaton  $\mathcal{A} = (Q, A, \cdot, \leq)$  if there exist a homomorphism  $g : X^* \rightarrow A^*$  and a semiautomata homomorphism  $\varphi : \mathcal{G} \rightarrow (Q, A, \cdot, \leq)^g$  such that  $\varphi(k) \not\leq \varphi(\ell)$ . In the opposite case, we say that  $\mathcal{A}$  avoids  $\mathcal{P}$ .

The definition is simply saying that  $\mathcal{A}$  avoids  $\mathcal{P} = [\mathcal{G}, k \not\leq \ell]$  if and only if the configuration  $(\mathcal{G}, k, \ell, \mathcal{C}_{\text{all}})$  is satisfied in  $\mathcal{A}$ . Based on this reformulation, we can define the most common used variant of forbidden patterns from the literature.

**Definition 4.2 (Variant 2)** A pattern  $\mathcal{P} = [\mathcal{G}, k \neq \ell]$  is defined as above. The definition “to be present in  $\mathcal{A}$ ” differs from the previous one having here  $\varphi(k) \neq \varphi(\ell)$  instead of  $\varphi(k) \not\leq \varphi(\ell)$ .

Note that  $\mathcal{A}$  avoids  $\mathcal{P} = [\mathcal{G}, k \neq \ell]$  if and only if both the configurations  $(\mathcal{G}, k, \ell, \mathcal{C}_{\text{all}})$  and  $(\mathcal{G}, \ell, k, \mathcal{C}_{\text{all}})$  are satisfied in  $\mathcal{A}$ .

We demonstrate that such type of patterns is natural by explaining that identities are covered by this notion. Consider an arbitrary identity  $u = v$ ,  $u, v \in X^*$ . Notice that the identity is satisfied in a monoid  $(M, \cdot)$  if, for each homomorphism  $\xi : (X^*, \cdot) \rightarrow (M, \cdot)$ , one has that  $\xi(u) = \xi(v)$ . Recall that the syntactic monoid of a regular language  $L$  is isomorphic to the transition monoid of the minimal automaton of  $L$ .

**Proposition 4.3** Let  $w = x_{i_1} \dots x_{i_m}$ ,  $u' = x_{j_1} \dots x_{j_n}$  and  $v' = x_{k_1} \dots x_{k_o}$  be words over the alphabet  $X$  such that  $j_1 \neq k_1$ . The transition monoid of an semiautomaton  $\mathcal{A} = (Q, A, \cdot)$  satisfies the identity  $wu' = wv'$  if and only if  $\mathcal{A}$  avoids the pattern from Figure 3.

*Proof.* The proof is obvious. □

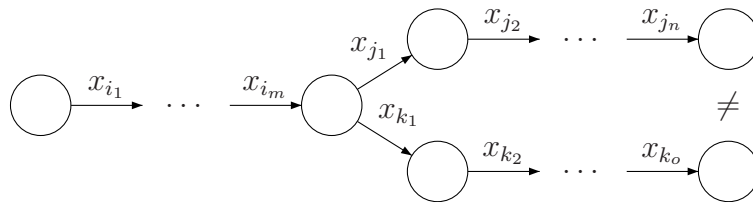


Figure 3: The forbidden pattern for the identity  $x_{i_1} \dots x_{i_m} x_{j_1} \dots x_{j_n} = x_{i_1} \dots x_{i_m} x_{k_1} \dots x_{k_o}$ .

For the inequality  $wu' \leq wv'$ , one uses in the pattern on Figure 3 the sign  $\not\leq$  instead of  $\neq$ . One can also formulate a modification where  $\mathcal{C}$ -identity or  $\mathcal{C}$ -inequality is considered. Since the semiautomaton on Figure 3 is simple and without loops, Proposition 3.8 can be applied here.

Sometimes a pattern is enriched by a condition that a certain state is final and another one is non-final.

**Definition 4.4 (Variant 3)** A pattern  $\mathcal{P} = [\mathcal{G}, m, n]$  consists of a partial semiautomaton  $\mathcal{G} = (V, X, \cdot)$  and a pair of states  $m, n \in V$ . The pattern  $\mathcal{P}$  is present in an automaton  $\mathcal{A} = (Q, A, \cdot, i, F)$  if there exist a homomorphism  $g : X^* \rightarrow A^*$  and a semiautomata homomorphism  $\varphi : \mathcal{G} \rightarrow (Q, A, \cdot)^g$  such that  $\varphi(m) \in F$  and  $\varphi(n) \notin F$ .

Now we show that this variant also fits into our theory of configurations under special assumption. Some applications are presented in Subsections 5.1 and 5.3.

**Proposition 4.5** *Let  $\mathcal{P} = [\mathcal{G}, m, n]$  be a pattern such that states  $m$  and  $n$  are reachable only by a single letter  $z$  from states  $k$  and  $\ell$ , respectively, and that the letter  $z \in X$  is acting just on these two states  $k, \ell$ . More formally, there exist  $\mathcal{G}' = (V', X', \cdot)$ ,  $k, \ell \in V'$ ,  $m, n \notin V'$  and  $z \notin X'$  such that  $\mathcal{G} = (V' \cup \{m, n\}, X' \cup \{z\}, \circ)$  where  $\circ$  is the extension of  $\cdot$  by the rules  $k \circ z = m$  and  $\ell \circ z = n$ . Let  $\mathcal{D} = (Q, A, \cdot, \leq, i, F)$  be an ordered automaton and denote  $\mathcal{A}_{\mathcal{D}} = (Q, A, \cdot, i, F)$  and  $\mathcal{O}_{\mathcal{D}} = (Q, A, \cdot, \leq)$ . Then the following holds:*

- (i) *If  $\mathcal{O}_{\mathcal{D}}$  satisfies the configuration  $\mathcal{K} = (\mathcal{G}', k, \ell, \mathcal{C}_{\text{all}})$  then  $\mathcal{A}_{\mathcal{D}}$  avoids  $\mathcal{P}$ .*
- (ii) *If  $\mathcal{D}$  is the minimal ordered automaton of some regular language such that  $\mathcal{O}_{\mathcal{D}}$  does not satisfy the configuration  $\mathcal{K} = (\mathcal{G}', k, \ell, \mathcal{C}_{\text{all}})$ , then  $\mathcal{P}$  is present in  $\mathcal{A}_{\mathcal{D}}$ .*

*Proof.* See [9], the full version of this paper. □

We finish with a variant which does not fit to our theory of configurations.

**Definition 4.6 (Variant 4)** *A pattern  $\mathcal{P} = [\mathcal{G}, k \neq \ell, m \neq n]$  consists of a partial semiautomaton  $\mathcal{G} = (V, X, \cdot)$  and two pairs of states  $m, n \in V$  and  $k, \ell \in V$ . The definition “to be present in  $\mathcal{A}$ ” differs from Definition 4.2 having both  $\varphi(m) \neq \varphi(n)$  and  $\varphi(k) \neq \varphi(\ell)$ .*

The next example shows that the corresponding class of (ordered) semiautomata need not be closed with respect to the products. This means that avoiding this pattern cannot be equivalent to satisfying an appropriate family of configurations.

**Example 4.7** *Consider the pattern  $\mathcal{P}$  from Figure 4. Let  $A = \{a, b\}$ ,  $K = a^*b$ ,  $L = aA^*$ . One can check that both the canonical ordered automata of  $K$  and  $L$  avoid  $\mathcal{P}$ , but  $\mathcal{P}$  is present in their product. More details can be found in the full version of this paper.*

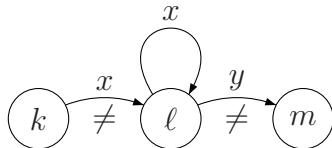


Figure 4: The forbidden pattern giving the class of semiautomata not closed under finite products.

## 5. Known Examples

In this section, we recall some examples of forbidden patterns from the literature (in the chronological order) and we discuss how they fit into our notions. Since some of the examples are results from the algebraic theory of regular languages, they used characterizations based on Eilenberg correspondence. Moreover, they use pseudoidentities for the characterization of certain pseudovarieties of finite monoids. Almost all examples in our contribution are of a very simple setting where pseudoidentities are only products of words and omega power of words.

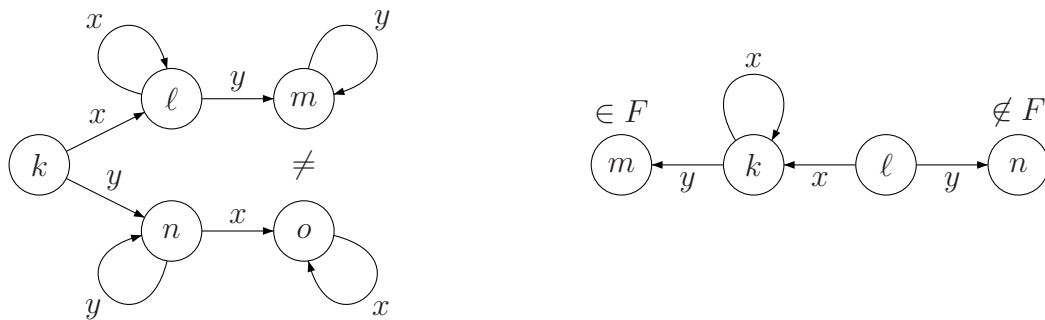
We recall that, for every element  $s$  of a finite semigroup  $S$ , there is the unique idempotent in the subsemigroup of  $S$  generated by the element  $s$ . This element is denoted by  $s^\omega$ . Moreover, for a fixed  $S$ , one has  $s^\omega = s^{|S|!}$ . We mention here formulations which use the fact that the (ordered) syntactic monoid of a language  $L$  is isomorphic to the (ordered) transition monoid of the minimal (ordered) automaton of  $L$ . Anyway, these characterizations are mentioned just for the completeness and the reader can just ignore them.

### 5.1. Reversible Languages

The examples are taken from Pin [10]. A language  $L \subseteq A^*$  is *reversible* if it is recognized by a deterministic and co-deterministic, possibly not complete, finite automaton with a set of initial states.

**Proposition 5.1** *A regular language  $L \subseteq A^*$  is reversible if and only if the ordered transition monoid  $M_L$  of the canonical ordered automaton  $\mathcal{O}_L$  for  $L$  satisfies the identity  $x^\omega y^\omega = y^\omega x^\omega$  and inequality  $x^\omega \leq 1$ . Moreover,*

- (i)  $M_L$  satisfies the first pseudoidentity if and only if  $\mathcal{O}_L$  avoids the pattern on Figure 5 (a).
- (ii)  $M_L$  satisfies the inequality  $x^\omega \leq 1$  if and only if  $\mathcal{O}_L$  avoids the pattern on Figure 5 (b).



(a) The pattern for commuting idempotents. (b) The pattern for the inequality  $x^\omega \leq 1$ .

Figure 5: The forbidden patterns for reversible languages.

Clearly, the pattern on Figure 5 (a) fits into Definition 4.2. For the pattern on Figure 5 (b), we can use Proposition 4.5 and we obtain the equivalent pattern on Figure 6 which fits into Definition 4.1. Moreover, the underlying partial semiautomata on Figure 5 (a) and Figure 6 are simple and balanced. Therefore, by Proposition 3.8 and Theorem 3.6, the class of all semiautomata which avoid one or both of these patterns forms a variety of ordered semiautomata. In particular, in the characterization in Proposition 5.1, one can use the condition that there is an automaton recognizing  $L$  which avoids the corresponding pattern.

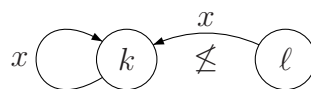


Figure 6: The forbidden pattern for inequality  $x^\omega \leq 1$  for ordered semiautomata.

### 5.2. Locally $\mathcal{R}$ - and $\mathcal{L}$ -trivial Semigroups

Here we refer to the work of Cohen, Perin and Pin [3] mentioned in the introduction. They deal solely with semigroups (not with monoids). Recall that the syntactic semigroup of a language  $L$  is the transition semigroup of the canonical semiautomaton of  $L$ . So, we can reformulate their results using our approach taking for  $\mathcal{E}$  the category  $\mathcal{C}_{ne}$  of all non-erasing homomorphisms.

**Proposition 5.2** *Let  $S$  be the transition semigroup of an automaton  $\mathcal{A}$ . Then*

- (i)  $S$  is  $\mathcal{R}$ -trivial if and only if  $\mathcal{A}$  avoids pattern on Figure 7 (a).
- (ii)  $S$  is  $\mathcal{L}$ -trivial if and only if  $\mathcal{A}$  avoids pattern on Figure 7 (b).
- (iii)  $S$  is a locally  $\mathcal{R}$ -trivial if and only if  $\mathcal{A}$  avoids pattern on Figure 7 (c).
- (iv)  $S$  is a locally  $\mathcal{L}$ -trivial if and only if  $\mathcal{A}$  avoids pattern on Figure 1 (see Section 1).

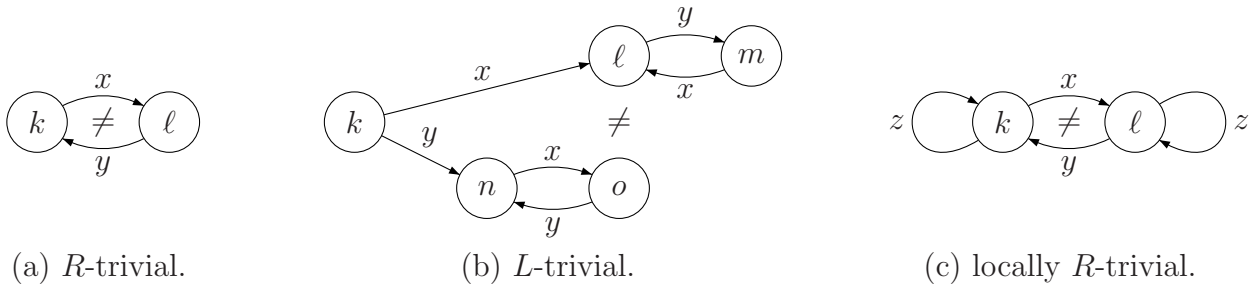


Figure 7: The forbidden patterns for (locally)  $R$ -trivial and  $L$ -trivial semigroups.

The class of all semiautomata avoiding the pattern on Figure 7 (a) is  $\mathbf{H}$ -invariant, although we cannot use Proposition 3.8 for a non-balanced pattern. Nevertheless, we can slightly modify the proof of that proposition and we get that the class of all semiautomata which avoid this pattern form a  $\mathcal{C}_{ne}$ -variety of semiautomata. Similarly, one can treat the remaining three patterns.

### 5.3. Concatenation Hierarchies

Here we follow the work by Pin and Weil [13]. It is mentioned that  $L$  belongs to the level  $1/2$  of the Straubing-Thérien hierarchy if and only if its ordered syntactic monoid satisfies the inequality  $1 \leq x$  and that this is equivalent to the fact that the canonical ordered automaton avoids the pattern from Figure 8 (a). Using Proposition 4.5 one obtains the following reformulation.

**Proposition 5.3** *A regular language  $L$  belongs to the level  $1/2$  of the Straubing-Thérien hierarchy if and only if its canonical ordered automaton avoids the pattern from Figure 8 (b).*

Trivially, the semiautomaton from Figure 8 (b) is simple and balanced. Therefore, one can also use in the characterization the condition that there is an automaton recognizing  $L$  which avoids the pattern.

They also recall that a regular language belongs to the level  $1/2$  of the dot-depth hierarchy if and only if the ordered syntactic semigroup satisfies the inequality  $y^\omega \leq y^\omega x y^\omega$ . Moreover,

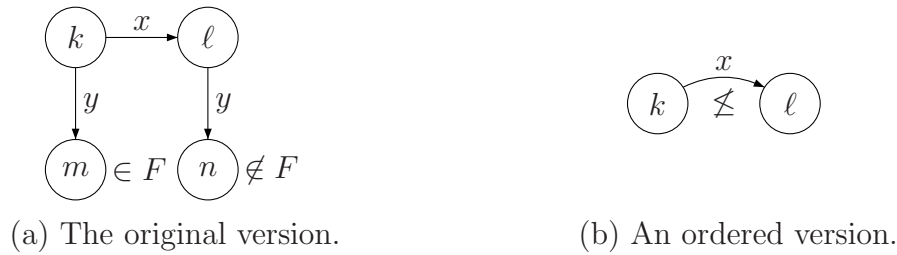


Figure 8: The forbidden patterns for level 1/2 in the Straubing-Thérien hierarchy.

this is equivalent to the fact that the canonical ordered automaton avoids the pattern from Figure 9 (a) substituting for  $x$  and  $y$  non-empty words. In our setting, we get the following statement.

**Proposition 5.4** *A regular language  $L$  belongs to the level 1/2 of the dot-depth hierarchy if and only if the canonical automaton of  $L$  satisfies the configuration  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{C}_{ne})$ , where  $\mathcal{G}$  is a semiautomaton given on Figure 9 (b) and  $\mathcal{C}_{ne}$  consists of all non-erasing homomorphisms.*

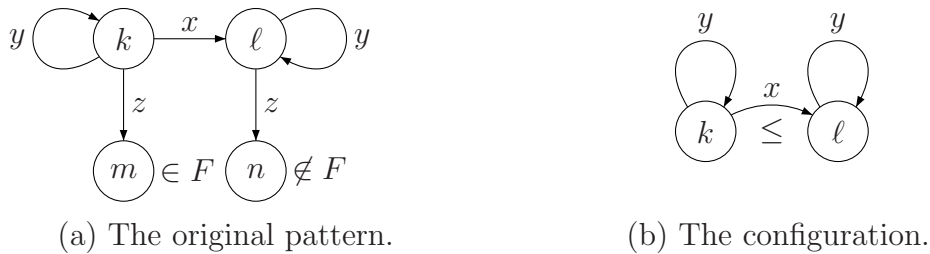


Figure 9: The forbidden pattern and the corresponding configuration for level 3/2 of the ST hierarchy and level 1/2 in the dot-depth hierarchy.

One of the characterizations from [13] says that  $L$  belongs to the level if and only if the syntactic ordered monoid of  $L$  satisfies all inequalities  $y^\omega \leq y^\omega x y^\omega$  where  $x$  and  $y$  are words with the same content. This is equivalent to the fact that the canonical ordered automaton avoids the pattern from Figure 9 (a) substituting for  $x$  and  $y$  the words of the same content. Now we present the formulation of the result using the notion of configuration again. Notice that we use here the full power of Definition 3.2.

**Proposition 5.5** *A regular language  $L$  belongs to the level 3/2 of Straubing-Thérien hierarchy of star-free languages if and only if the canonical ordered semiautomaton of  $L$  satisfies the configuration  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{E})$ , where  $\mathcal{G}$  is a semiautomaton given on Figure 9 (b) and  $\mathcal{E}$  consists of homomorphisms with constant content.*

One can see that the semiautomaton from Figure 9 (b) is simple and balanced. However, we cannot use Proposition 4.5 directly, because we do not work with the category  $\mathcal{C}_{all}$ , but we use categories  $\mathcal{C}_{ne}$  and  $\mathcal{E}$ , respectively. Notice that an appropriate modification of Proposition 4.5 is possible.

To complete the overview concerning characterization of certain levels of the Straubing-Thérien hierarchy, we could mentioned that it is well known that the level 1 corresponds to  $\mathcal{J}$ -trivial

monoids by a famous result established by Simon [16]. Then one can use the characterizations from Proposition 5.2, because  $\mathcal{J}$ -trivial monoids are exactly monoids which are both  $\mathcal{R}$ -trivial and  $\mathcal{L}$ -trivial. This characterization is implicitly contained in Stern [17]. He also considered the level 1 in the dot-depth hierarchy using graph techniques.

## 5.4. A Variant of Reversibility

The results of this subsection are taken from Golovkins and Pin [5]. The authors define a modification of reversible languages, namely they consider the class  $\mathbb{R}$  consisting of all languages recognized by automata with at most two absorbing states, such that each letter  $a \in A$  acts on non-absorbing states injectively. Let  $\overline{\mathbb{R}}$  be the Boolean closure of  $\mathbb{R}$ . They proved, among others, the next result.

**Proposition 5.6** *Let  $L$  be a regular language and  $\mathcal{D}_L$  its canonical automaton.*

- (i) *A language  $L$  is in  $\mathbb{R}$  if and only if the pattern from Figure 4 is not present in  $\mathcal{D}_L$ .*
- (ii) *A language  $L$  is in  $\overline{\mathbb{R}}$  if and only if its syntactic monoid satisfies the identity  $x^\omega y^\omega x^\omega = x^\omega y^\omega$ . Equivalently, the pattern from Figure 10 is not present in  $\mathcal{D}_L$ .*

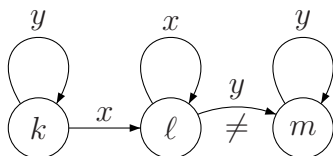


Figure 10: The forbidden pattern for the class  $\overline{\mathbb{R}}$ .

The case of the class  $\mathbb{R}$  does not suit our theory due to Example 4.7 or due to the fact that this class is not closed with respect to unions nor intersections. On the other hand, we can state that the semiautomaton on Figure 10 is simple and balanced. Therefore, this pattern fits to our theory perfectly.

## 5.5. Variants of Definite Languages

Iván and Nagy-György in [7] defined forbidden patterns in general. They looked only for embeddings of a given pattern in a semiautomata. Since all their examples use only the patterns with two states, they fit to our general theory. They recall the notions of cofinite, definitive, codefinite and generalized definite languages. The authors formulated characterizations of those four classes of languages via forbidden patterns. One can easily modify two of four patterns such that everything suits to Proposition 3.8.



### 5.6. Another Variant of Reversibility

Holzer et al. define in [6] another variant of reversibility. They say that a language  $L$  is *reversible* if it is accepted by a partial co-deterministic finite automaton. They prove the following characterization.

**Proposition 5.7** *A language is reversible if and only if its minimal trim automaton avoids the pattern from Figure 11.*

Notice that this concept could not suit to our theory since the class of corresponding languages is not closed with respect to unions – see [6].

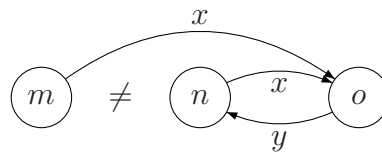
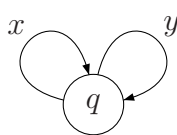


Figure 11: The forbidden pattern for another variant of reversibility.

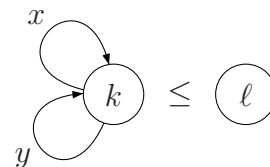
### 5.7. Sparse Languages

The sparse languages are defined by the density function, which counts the number of words of length  $n$  in  $L$ , that is, the function  $d_L : \mathbb{N} \rightarrow \mathbb{N}$  given by  $d_L(n) = |L \cap A^n|$ . A language  $L$  is *sparse* if  $d_L(n) = O(n^k)$  for some  $k > 0$ . Basic results are overviewed in [19, 12], however here we are interested in the characterization from [12].

**Proposition 5.8** *Let  $L \subseteq A^*$  be an arbitrary regular language and  $\mathcal{D}'_L$  be a partial semiautomaton which is the minimal trim automaton of  $L$ . Then the language  $L$  is sparse if and only if the partial semiautomaton  $\mathcal{D}'_L$  does not contain the forbidden pattern from Figure 12 (a) with the additional assumption that the first letter in  $g(x)$  is different from the first letter in  $g(y)$ .*



(a) The forbidden pattern.



(b) The configuration.

Figure 12: The characterization of sparse languages.

Now we introduce a modification of the characterization which fits into our concept of configurations. We could point out that the class of sparse languages is not closed under preimages in all homomorphisms, but just injective homomorphisms. As our final goal is a description of the positive  $\mathcal{C}_1$ -variety of sparse languages, we need to add to each  $\mathcal{V}(A)$  the full language  $A^*$  which are members of each positive  $\mathcal{C}$ -variety of languages.



**Proposition 5.9** *The regular language is sparse or full if and only if the canonical ordered semiautomaton  $\mathcal{O}_L$  of  $L$  satisfies the configuration  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{C}_i)$ , where  $\mathcal{G}$  is given by Figure 12 (b) and  $\mathcal{C}_i$  is the category of injective homomorphisms.*

*Proof.* See [9], the full version of this paper. □

## 6. Conclusion

We have introduced a formalism for forbidden patterns which differs from those in the existing literature. The main difference is that our notion is defined in a more general setting, namely for ordered automata. Among others, this enables us to reformulate some of known examples in a new way in Section 5.3. Furthermore, our forbidden patterns or configurations are mapped into a given automaton in contrast to some mentioned papers, where the pattern is viewed as a subautomaton. Although the basic ideas are similar, in the other formalisms it is not sometimes clear which states need to be really different. The main result of the contribution is the following consequence of Theorem 3.6 and Proposition 3.8.

**Theorem 6.1** *Let  $\mathcal{K} = (\mathcal{G}, k, \ell, \mathcal{E})$  be a configuration such that  $\mathcal{G}$  is connected, balanced and simple partial semiautomaton and  $\mathcal{E}$  is closed under extensions by  $\mathcal{C}$ . Then the class of all ordered semiautomata satisfying  $\mathcal{K}$  forms a  $\mathcal{C}$ -variety of ordered semiautomata.*

The examples of configurations satisfying the assumption of the theorem we saw in Section 5.1.

There are some natural questions concerning future applications of this theory of forbidden patterns. The first goal should be to extend the previous result to broaden the class of configurations because the assumptions in Theorem 6.1 are quite restrictive. We mentioned in Section 5.2 that such extensions are possible.

There is also an important practical question whether the satisfiability of a configuration can be algorithmically decided in a given (ordered) automaton  $\mathcal{A}$ . Surely, it is clear whenever the family of substitutions  $\mathcal{E}$  is formed by all homomorphisms, since (i) there are only finitely many mappings  $\varphi$  from  $\mathcal{K}$  into  $\mathcal{A}$ , (ii) for every such  $\varphi$  and each  $x \in X$ , one only needs to check whether there exists a certain word  $g(x) \in A^*$  which transforms certain states in  $\mathcal{A}$  in a required way. However, the computation of the set of all transitions given by words is the known construction of the transition monoid of the automaton. Thus the question remains to be more interesting in cases when  $\mathcal{E}$  is more exotic family. Note that it is doable in case of exotic families we mentioned in our paper, namely those from Propositions 5.5 and 5.8.

## References

- [1] J. A. BRZOZOWSKI, Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata* 12 (1962), 529–561.
- [2] L. CHAUBARD, J. PIN, H. STRAUBING, Actions, Wreath products of  $C$ -varieties and concatenation product. *Theoretical Computer Science* 356 (2006) 1-2, 73–89.  
<https://doi.org/10.1016/j.tcs.2006.01.039>
- [3] J. COHEN, D. PERRIN, J. PIN, On the expressive power of temporal logic. *Journal of Computer and System Sciences* 46 (1993) 3, 271–294.
- [4] C. GLASSER, H. SCHMITZ, Languages of dot-depth  $3/2$ . In: H. REICHEL, S. TISON (eds.), *STACS 2000*. Lecture Notes in Computer Science 1770, Springer, 2000, 555–566.
- [5] M. GOLOVKINS, J. PIN, Varieties generated by certain models of reversible finite automata. *Chicago Journal of Theoretical Computer Science* 2010 (2010) 2.
- [6] M. HOLZER, S. JAKOBI, M. KUTRIB, Minimal reversible deterministic finite automata. *International Journal of Foundations of Computer Science* 29 (2018) 2, 251–270.
- [7] S. IVÁN, J. NAGY-GYÖRGY, On the structure and syntactic complexity of generalized definite languages. 2013.  
<http://arxiv.org/abs/1304.5714>
- [8] O. KLÍMA, L. POLÁK, On varieties of ordered automata. 2017.  
<https://arxiv.org/abs/1712.08455>
- [9] O. KLÍMA, L. POLÁK, Forbidden patterns for ordered automata. 2018.  
<https://www.muni.cz/~klima/Math/publications.html>
- [10] J. PIN, On reversible automata. In: I. SIMON (ed.), *LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings*. Lecture Notes in Computer Science 583, Springer, 1992, 401–416.
- [11] J. PIN, A variety theorem without complementation. *Russian Mathematics* 39 (1995), 80–90.
- [12] J. PIN, *Mathematical Foundations of Automata Theory*. 2016.  
<https://www.irif.fr/~jep/MPRI/MPRI.html>
- [13] J. PIN, P. WEIL, Polynomial closure and unambiguous product. *Theory of Computing Systems* 30 (1997) 4, 383–422.
- [14] H. SCHMITZ, *The Forbidden Pattern Approach to Concatenation Hierarchies*. Ph.D. thesis, Julius Maximilians University Würzburg, Germany, 2000.  
[urn:nbn:de:bvb:20-opus-2832](https://nbn-resolving.org/urn:nbn:de:bvb:20-opus-2832)
- [15] H. SCHMITZ, K. W. WAGNER, The Boolean hierarchy over level  $1/2$  of the Straubing-Therien hierarchy. *CoRR* cs.CC/9809118 (1998).
- [16] I. SIMON, Piecewise testable events. In: H. BARKHAGE (ed.), *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*. Lecture Notes in Computer Science 33, Springer, 1975, 214–222.

- [17] J. STERN, Characterizations of some classes of regular events. *Theoretical Computer Science* 35 (1985), 17–42.
- [18] H. STRAUBING, On logical descriptions of regular languages. In: S. RAJSBAUM (ed.), *LATIN 2002: Theoretical Informatics*. Lecture Notes in Computer Science 2286, Springer, 2002, 528–538.
- [19] S. YU, Regular Languages. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages, Vol. 1*. Springer, 1997, 41–110.



# A JUMPING $5' \rightarrow 3'$ WATSON-CRICK FINITE AUTOMATA MODEL

Radim Kocman<sup>(A)</sup>      Benedek Nagy<sup>(B)</sup>  
Zbyněk Křivka<sup>(A)</sup>      Alexander Meduna<sup>(A)</sup>

<sup>(A)</sup> Centre of Excellence IT4Innovations, Faculty of Information Technology,  
Brno University of Technology, Božetěchova 2, Brno Czech Republic  
`{ikocman,krivka,meduna}@fit.vutbr.cz`

<sup>(B)</sup> Department of Mathematics, Faculty of Arts and Sciences,  
Eastern Mediterranean University, Famagusta, North Cyprus, Mersin-10, Turkey  
`nbenedek.inf@gmail.com`

## **Abstract**

*This paper introduces and studies a combined model of jumping finite automata and sensing  $5' \rightarrow 3'$  Watson-Crick finite automata. The accepting power of the new model is compared with the original models and also with some well-known language families. Furthermore, the paper investigates changes in the accepting power when restrictions are applied on the model.*

## **1. Introduction**

In recent years, several papers studied finite automata models with multiple heads that process the input string in non-conventional ways (see [1, 2, 7, 8, 9, 10]). Traditionally, when the model utilizes several heads, either each head works on its own tape, or all heads read the same input string in a symbol-by-symbol left-to-right way. In contrast, there are also well-established formal grammars that generate strings in a parallel way, but this process is usually very different than the reading with several heads. In grammars, the sentential form is repeatedly rewritten on several places at once until the process creates the final string. The presented finite automata models with the non-conventional processing have their behavior set somewhere between the mentioned models. They utilize several heads, but these heads cooperate on a single tape to process the single input string. Therefore, every symbol in the input string is read only once, and the heads do not work in the traditional symbol-by-symbol left-to-right way.

The first group of these models is based on *jumping finite automata* (see [5, 6, 1, 2]). This concept is in its core focused on discontinuous information processing. In essence, a jumping finite automaton works just like a classical finite automaton except it does not read the input string in a symbol-by-symbol left-to-right way. After the automaton reads a symbol, the head can jump over (skip) a portion of the tape in either direction. Once an occurrence of a symbol is read on the tape, it cannot be re-read again later. Generally, this model can very easily define

even non-context-free languages if the order of symbols is unimportant for the language. On the other hand, the resulting language families of these models are usually incomparable with the classical families of regular, linear, and context-free languages. When this concept utilizes multiple heads, the heads can naturally jump on specific positions in the tape, and thus they can easily work on different places at once in parallel.

The second group is represented by *sensing*  $5' \rightarrow 3'$  *Watson-Crick (WK) finite automata* (see [7, 8, 9, 10, 11]). This is a biology-inspired concept. In essence, a WK automaton also works just like a classical finite automaton except it uses a WK tape (i.e., double-stranded tape), and it has a separate head for each of the two strands in the tape. This is therefore a concept that always naturally uses two heads. In a  $5' \rightarrow 3'$  WK automaton, both heads read their specific strand in the biochemical  $5'$  to  $3'$  direction. In a computing point of view, however, this means that they read the double strand sequence in opposite directions. Finally, a  $5' \rightarrow 3'$  WK automaton is *sensing* if the heads sense that they are meeting each other, and the processing of the input ends if for all pairs of the sequence one of the letters is read. Sensing  $5' \rightarrow 3'$  WK automata generally accept the family of linear languages.

Even though that these concepts are significantly different, their models sometimes work in a very similar way. Both concepts are also not mutually exclusive in a single formal model. This paper defines *jumping*  $5' \rightarrow 3'$  *WK automata* – a combined model of jumping finite automata and sensing  $5' \rightarrow 3'$  WK automata – and studies their characteristics. We primarily investigate the accepting power of the model and also the effects of restrictions on the model.

## 2. Preliminaries

This paper assumes that the reader is familiar with the theory of automata and formal languages (see [4, 13]). This section recalls only the crucial notions used in this paper.

For a set  $Q$ ,  $\text{card}(Q)$  denotes the cardinality of  $Q$ , and  $2^Q$  denotes the power set of  $Q$ . For an alphabet (finite nonempty set)  $V$ ,  $V^*$  represents the free monoid generated by  $V$  under the operation of concatenation. The unit of  $V^*$  is denoted by  $\varepsilon$ . Members of  $V^*$  are called *strings*;  $V^+ = V^* - \{\varepsilon\}$ ; algebraically,  $V^*$  thus is the free semigroup generated by  $V$  under the operation of concatenation. For  $x \in V^*$ ,  $|x|$  denotes the length of  $x$ , and  $\text{alph}(x)$  denotes the set of all symbols occurring in  $x$ ; for instance,  $\text{alph}(0010) = \{0, 1\}$ . For  $a \in V$ ,  $|x|_a$  denotes the number of occurrences of  $a$  in  $x$ . Let  $X$  and  $Y$  be sets; we call  $X$  and  $Y$  to be *incomparable* if  $X \not\subseteq Y$ ,  $Y \not\subseteq X$ , and  $X \cap Y \neq \emptyset$ .

A *general grammar* or, more simply, a *grammar* is quadruple  $G = (N, T, S, P)$ , where  $N$  and  $T$  are alphabets such that  $N \cap T = \emptyset$ ,  $S \in N$ , and  $P$  is a finite set of rules of the form  $x \rightarrow y$ , where  $x, y \in (N \cup T)^*$  and  $\text{alph}(x) \cap N \neq \emptyset$ . If  $x \rightarrow y \in P$  and  $u, v \in (N \cup T)^*$ , then  $uxv \Rightarrow uyv$  [ $x \rightarrow y$ ], or simply  $uxv \Rightarrow uyv$ . In the standard manner, extend  $\Rightarrow$  to  $\Rightarrow^n$ , where  $n \geq 0$ ; then, based on  $\Rightarrow^n$ , define  $\Rightarrow^+$  and  $\Rightarrow^*$ . The language generated by  $G$ ,  $L(G)$ , is defined as  $L(G) = \{w \in T^* : S \Rightarrow^* w\}$ . We recognize several special cases of grammars:  $G$  is a *context-sensitive grammar* if every  $x \rightarrow y \in P$  satisfies  $x = \alpha A \beta$  and  $y = \alpha y \beta$  such that

$A \in N$ ,  $\alpha, \beta \in (N \cup T)^*$ , and  $y \in (N \cup T)^+$ .  $G$  is a *context-free grammar* if every  $x \rightarrow y \in P$  satisfies  $x \in N$ .  $G$  is a *linear grammar* if every  $x \rightarrow y \in P$  satisfies  $x \in N$  and  $y \in T^*NT^* \cup T^*$ .  $G$  is a *regular grammar* if every  $x \rightarrow y \in P$  satisfies  $x \in N$  and  $y \in TN \cup T$ .

A *finite automaton* is a quintuple  $A = (V, Q, q_0, F, \delta)$ , where  $V$  is an input alphabet,  $Q$  is a finite set of states,  $V \cap Q = \emptyset$ ,  $q_0 \in Q$  is the initial (or start) state, and  $F \subseteq Q$  is a set of final (or accepting) states. The mapping  $\delta$  is a transition function. If  $\delta: Q \times (V \cup \{\varepsilon\}) \rightarrow 2^Q$ , then the device is non-deterministic; if  $\delta: Q \times V \rightarrow Q$ , then the automaton is deterministic. A string  $w$  is accepted by a finite automaton if there is a sequence of transitions starting from  $q_0$ , ending in a state in  $F$ , and the symbols of the sequence yield  $w$ . A language is regular if and only if it can be recognized by a finite automaton. Let **REG**, **LIN**, **CF**, and **CS** denote the families of regular, linear, context-free, and context-sensitive languages, respectively.

## 2.1. Jumping Finite Automata

A *general jumping finite automaton* (see [5, 6]), a *GJFA* for short, is a quintuple  $M = (Q, \Sigma, R, s, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is an input alphabet,  $Q \cap \Sigma = \emptyset$ ,  $R \subseteq Q \times \Sigma^* \times Q$  is finite,  $s \in Q$  is the start state, and  $F \subseteq Q$  is a set of final states. Members of  $R$  are referred to as rules of  $M$ . If  $(p, y, q) \in R$  implies that  $|y| \leq 1$ , then  $M$  is a *jumping finite automaton*, a *JFA* for short. A configuration of  $M$  is any string in  $\Sigma^*Q\Sigma^*$ . The binary jumping relation, symbolically denoted by  $\curvearrowright$ , over  $\Sigma^*Q\Sigma^*$ , is defined as follows. Let  $x, z, x', z' \in \Sigma^*$  such that  $xz = x'z'$  and  $(p, y, q) \in R$ ; then,  $M$  makes a *jump* from  $xpyz$  to  $x'qz'$ , symbolically written as  $xpyz \curvearrowright x'qz'$ . In the standard manner, extend  $\curvearrowright$  to  $\curvearrowright^n$ , where  $n \geq 0$ ; then, based on  $\curvearrowright^n$ , define  $\curvearrowright^+$  and  $\curvearrowright^*$ . The language accepted by  $M$ , denoted by  $L(M)$ , is defined as  $L(M) = \{uv : u, v \in \Sigma^*, usv \curvearrowright^* f, f \in F\}$ . We say that  $M$  accepts  $w$  if and only if  $w \in L(M)$ .  $M$  rejects  $w$  if and only if  $w \in \Sigma^* - L(M)$ .

More recently, *double-jumping modes* for GJFAs were introduced (see [1]), which perform two single jumps simultaneously. Both jumps always follow the same rule, however, they are performed on two different positions on the tape and thus handle different parts of the input string. Additionally, these jumps cannot ever cross each other (i.e., the initial mutual order of reading positions is preserved during the whole accepting process). The specific double-jumping modes then assign one of the three jumping directions to each of the two jumps – (1) to the left, (2) to the right, and (3) in either direction. We omit the precise formal definition.

## 2.2. Watson-Crick Finite Automata

In this part we recall some well-known concepts of DNA computing and related formal language theory. Readers who are not familiar in these topics should read [11].

Let  $V$  be an alphabet and  $\rho \subseteq V \times V$  be its complementary relation. For instance,  $V = \{A, C, G, T\}$  is usually used in DNA computing with the Watson-Crick complementary relation  $\{(T, A), (A, T), (C, G), (G, C)\}$ . The strings built up by complementary pairs of letters are double strands (of DNA).

A *Watson-Crick finite automaton* (or shortly, a *WK automaton*) is a finite automaton working on a Watson-Crick tape, that is, a double-stranded sequence (or molecule) in which the lengths of the strands are equal and the elements of the strands are pairwise complements of each other:  $[\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}][\begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix}] \cdots [\begin{smallmatrix} a_n \\ b_n \end{smallmatrix}] = [\begin{smallmatrix} a_1 a_2 \cdots a_n \\ b_1 b_2 \cdots b_n \end{smallmatrix}]$  with  $a_i, b_i \in V$  and  $(a_i, b_i) \in \rho$  ( $i = 1, \dots, n$ ). The notation  $[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}]$  is used only for strings  $w_1, w_2$  with equal length and satisfying the complementary relation  $\rho$ . The set of all double-stranded strings with this property is denoted by  $\text{WK}_\rho(V)$ . For double-stranded strings for which these conditions are not necessarily satisfied, the notation  $(\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix})$  is used throughout the paper. Formally, a WK automaton is  $M = (V, \rho, Q, q_0, F, \delta)$ , where  $V$ ,  $Q$ ,  $q_0$ , and  $F$  are the same as in finite automata,  $\rho \subseteq V \times V$  is a symmetric relation, and the transition mapping  $\delta: (Q \times (V^*)) \rightarrow 2^Q$  in such a way that  $\delta(q, (\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}))$  ( $q \in Q, w_1, w_2 \in V^*$ ) is non-empty only for finitely many values of  $(q, (\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}))$ .

The elementary difference between finite automata and WK automata, besides the doubled tape, is the number of heads. WK automata scan each of the two strands separately with a unique head. In classical WK automata, the processing of the input sequence ends if all pairs of the sequence are read with both heads. There are also some restricted variations of WK automata which are widely used in the literature (see, e.g., [11]):

- **N** : stateless, i.e., with only one state: if  $Q = F = \{q_0\}$ ;
- **F** : all-final, i.e., with only final states: if  $Q = F$ ;
- **S** : simple (at most one head moves in a step)  $\delta: (Q \times ((\begin{smallmatrix} V^* \\ \{\varepsilon\} \end{smallmatrix}) \cup (\begin{smallmatrix} \{\varepsilon\} \\ V^* \end{smallmatrix})))) \rightarrow 2^Q$ ;
- **1** : 1-limited (exactly one letter is being read in a step)  $\delta: (Q \times ((\begin{smallmatrix} V \\ \{\varepsilon\} \end{smallmatrix}) \cup (\begin{smallmatrix} \{\varepsilon\} \\ V \end{smallmatrix})))) \rightarrow 2^Q$ .

Further variations such as **NS**, **FS**, **N1**, and **F1** WK automata can be identified in a straightforward way by using multiple constraints.

In  $5' \rightarrow 3'$  *WK automata* (see [7, 8, 9, 10]), both heads start from the 5' end of the appropriate strand. Physically/mathematically and from a computing point of view they read the double-stranded sequence in opposite directions, while biochemically they go to the same direction. A  $5' \rightarrow 3'$  WK automaton is *sensing* if the heads sense that they are meeting (i.e., they are close enough to meet in the next step or there is a possibility to read strings at overlapping positions). In sensing  $5' \rightarrow 3'$  WK automata, the processing of the input sequence ends if for all pairs of the sequence one of the letters is read. Due to the complementary relation, the sequence is fully processed; thus, the automaton makes a decision on the acceptance.

In the usual WK automata, the state transition is a mapping of the form  $(Q \times (V^*)) \rightarrow 2^Q$ . In a transition  $q' \in \delta(q, (\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}))$ , we call  $r_l = |w_1|$  and  $r_r = |w_2|$  the left and right *radius* of the transition (they are the lengths of the strings that the heads read from *left to right* and from *right to left* in this step, respectively). The value  $r = r_l + r_r$  is the radius of the transition. Since  $\delta(q, (\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}))$  is non-empty only for finitely many triplets of  $(q, w_1, w_2)$ , there is a transition (maybe more) with the maximal radius for a given automaton. Let  $\delta$  be extended by the sensing condition in the following way: Let  $r$  be the maximum of the values  $r_l + r_r$  for the values given in the transition function of the original WK automaton. Then, let  $\delta': (Q \times (V^*) \times D) \rightarrow 2^Q$ , where  $D$  is the *sensing distance set*  $\{-\infty, 0, 1, \dots, r, +\infty\}$ . This set gives the distance of the two heads between 0 and  $r$ ,  $+\infty$  when the heads are further than  $r$ , or  $-\infty$  when the heads are after their meeting point. Trivially, this automaton is finite, and  $D$  can be used only to control



the sensing (i.e., the appropriate meeting of the heads). To describe the work of the automata, we use the concept of configuration. A configuration  $(\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix})(q, s)(\begin{smallmatrix} w'_1 \\ w'_2 \end{smallmatrix})$  consists of the state  $q$ , the actual sensing distance  $s$ , and the input  $[\begin{smallmatrix} w_1 w'_1 \\ w_2 w'_2 \end{smallmatrix}] \in \text{WK}_\rho(V)$  in such a way that the first head (upper strand) has already processed the part  $w_1$ , while the second head (lower strand) has already processed  $w'_2$ . A step of the automaton, according to the state transition function, can be of the following two types:

- (1) Normal steps :  $(\begin{smallmatrix} w_1 \\ w_2 y \end{smallmatrix})(q, +\infty)(\begin{smallmatrix} x w'_1 \\ w'_2 \end{smallmatrix}) \Rightarrow (\begin{smallmatrix} w_1 x \\ w_2 \end{smallmatrix})(q', s)(\begin{smallmatrix} w'_1 \\ y w'_2 \end{smallmatrix})$ , for  $w_1, w_2, w'_1, w'_2, x, y \in V^*$  with  $|w_2 y| - |w_1| > r$ ,  $q, q' \in Q$ , if and only if  $[\begin{smallmatrix} w_1 x w'_1 \\ w_2 y w'_2 \end{smallmatrix}] \in \text{WK}_\rho(V)$  and  $q' \in \delta(q, (\begin{smallmatrix} x \\ y \end{smallmatrix}), +\infty)$ , and  $s = \begin{cases} |w_2| - |w_1 x| & \text{if } |w_2| - |w_1 x| \leq r; \\ +\infty & \text{in other cases.} \end{cases}$
- (2) Sensing steps :  $(\begin{smallmatrix} w_1 \\ w_2 y \end{smallmatrix})(q, s)(\begin{smallmatrix} x w'_1 \\ w'_2 \end{smallmatrix}) \Rightarrow (\begin{smallmatrix} w_1 x \\ w_2 \end{smallmatrix})(q', s')(\begin{smallmatrix} w'_1 \\ y w'_2 \end{smallmatrix})$ , for  $w_1, w_2, w'_1, w'_2, x, y \in V^*$ , if and only if  $[\begin{smallmatrix} w_1 x w'_1 \\ w_2 y w'_2 \end{smallmatrix}] \in \text{WK}_\rho(V)$  and  $q' \in \delta(q, (\begin{smallmatrix} x \\ y \end{smallmatrix}), s)$ , and  $s' = \begin{cases} s - |x| - |y| & \text{if } s - |x| - |y| \geq 0; \\ -\infty & \text{in other cases.} \end{cases}$

In the standard manner, extend  $\Rightarrow$  to  $\Rightarrow^n$ , where  $n \geq 0$ ; then, based on  $\Rightarrow^n$ , define  $\Rightarrow^+$  and  $\Rightarrow^*$ . The accepted language, denoted by  $L(M)$ , can be defined by the final accepting configurations that can be reached from the initial one: A double strand  $[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}]$  is accepted by a sensing 5' → 3' WK automaton  $M$  if and only if  $(\begin{smallmatrix} \varepsilon \\ w_2 \end{smallmatrix})(q_0, s_0)(\begin{smallmatrix} w_1 \\ \varepsilon \end{smallmatrix}) \Rightarrow^* [\begin{smallmatrix} w'_1 \\ w'_2 \end{smallmatrix}](q_f, 0)[\begin{smallmatrix} w''_1 \\ w''_2 \end{smallmatrix}]$ , for  $q_f \in F$ , where  $[\begin{smallmatrix} w'_1 \\ w'_2 \end{smallmatrix}][\begin{smallmatrix} w''_1 \\ w''_2 \end{smallmatrix}]] = [\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}]$  with the proper value of  $s_0$  (it is  $+\infty$  if  $|w_1| > r$ , elsewhere it is  $|w_1|$ ); since the full input is processed by the time the heads meet.

From a biochemical point of view, a double-stranded sequence has no distinguishable start and end. Consequently, each word that is accepted by a WK automaton has a complement-symmetric pair which is also in the language. This fact does not cause any problem in connection to formal language theory. For instance, double strands having only  $A$  and  $C$  in a strand (and thus having  $T$  and  $G$  in the other) can represent languages over a binary alphabet: considering the pair  $[\begin{smallmatrix} A \\ T \end{smallmatrix}]$  as letter  $a$  and  $[\begin{smallmatrix} C \\ G \end{smallmatrix}]$  as letter  $b$  in the new alphabet  $V'$ .

At the end, we briefly mention other closely related 5' → 3' WK automata models. Besides the sensing version, the papers [7, 8, 9] also define the *full-reading sensing version*. The formal definition remains practically identical, however, the automaton continues with the reading after the meeting point, and both heads have to read the whole strand from the 5' end to the 3' end. The resulting behavior therefore combines some properties of classical WK automata and sensing 5' → 3' WK automata. It can be easily seen that the full-reading sensing version is generally stronger than the sensing version. Lastly, the paper [10] introduces a version of sensing 5' → 3' WK automata without the sensing distance. It shows that it is not strictly necessary to know the precise sensing distance and that we can obtain the same power even if we are able to recognize only the actual meeting event. Nonetheless, this result does not hold in general if we consider the restricted variations of these models.

### 3. Definitions

Considering sensing  $5' \rightarrow 3'$  WK automata and full-reading sensing  $5' \rightarrow 3'$  WK automata, there is quite a large gap between their behaviors. The definition of sensing  $5' \rightarrow 3'$  WK automata states that we need to read only one of the letters from all pairs of the input sequence before it is fully processed. However, this also limits the positioning of the heads because they can read letters only until they meet. On the other hand, the definition of full-reading sensing  $5' \rightarrow 3'$  WK automata allows the heads to traverse the whole input. Nonetheless, this also means that all pairs of the input sequence will be read twice. Considering other models, jumping finite automata offer a mechanism that allows heads to skip (jump over) some symbols. Moreover, in some of the recently introduced double-jumping modes, these automata behave very similarly to  $5' \rightarrow 3'$  WK automata. It is therefore our goal to fill the gap by introducing the jumping mechanism into sensing  $5' \rightarrow 3'$  WK automata. We want the heads to be able to traverse the whole input, but we also want to read all pairs of the input sequence only once.

It is possible to fit the jumping mechanism straightforwardly into the original definition of sensing  $5' \rightarrow 3'$  WK automata. Observe that we are also newly tracking only the meeting event of the heads and not the precise sensing distance.

**Definition 3.1** A sensing  $5' \rightarrow 3'$  WK automaton with jumping feature is a 6-tuple  $M = (V, \rho, Q, q_0, F, \delta)$ , where  $V$ ,  $\rho$ ,  $Q$ ,  $q_0$ , and  $F$  are the same as in WK automata,  $V \cap \{\#\} = \emptyset$ ,  $\delta: (Q \times (V^*) \times D) \rightarrow 2^Q$ , where  $D = \{\oplus, \ominus\}$  indicates the mutual position of heads, and the transition function assigns a nonempty set only for finitely many triplets of  $(Q \times (V^*) \times D)$ . We denote the head as  $\blacktriangleright$ -head or  $\blacktriangleleft$ -head if it reads from left to right or from right to left, respectively. We use symbol  $\oplus$  if the  $\blacktriangleright$ -head is on the input tape positioned before the  $\blacktriangleleft$ -head; otherwise, we use symbol  $\ominus$ . A configuration  $\binom{w_1}{w_2}(q, s)\binom{w'_1}{w'_2}$  has the same structure as in sensing  $5' \rightarrow 3'$  WK automata; however,  $s$  indicates only the mutual position of heads, and a partially processed input  $\binom{w_1 w'_1}{w_2 w'_2}$  may not satisfy the complementary relation  $\rho$ . A step of the automaton can be of the following two types: Let  $w'_1, w_2, x, y \in V^*$  and  $w_1, w'_2 \in (V \cup \{\#\})^*$ .

- (1) Reading steps:  $\binom{w_1}{w_2 y}(q, s)\binom{x w'_1}{w'_2} \rightsquigarrow \binom{w_1 \{\#\}^{|x|}}{w'_2}(q', s')\binom{w'_1}{\{\#\}^{|y|} w'_2}$ , where  $q' \in \delta(q, \binom{x}{y}, s)$ , and  $s'$  is either  $\oplus$  if  $|w_2| > |w_1 x|$  or  $\ominus$  in other cases.
- (2) Jumping steps:  $\binom{w_1}{w_2 v}(q, s)\binom{u w'_1}{w'_2} \rightsquigarrow \binom{w_1 u}{w_2}(q, s')\binom{w'_1}{v w'_2}$ , where  $s'$  is either  $\oplus$  if  $|w_2| > |w_1 u|$  or  $\ominus$  in other cases.

Note that the jumping steps are an integral and inseparable part of the behavior of the automaton, and thus they are not affected by the state transition function. In the standard manner, extend  $\rightsquigarrow$  to  $\rightsquigarrow^n$ , where  $n \geq 0$ ; then, based on  $\rightsquigarrow^n$ , define  $\rightsquigarrow^+$  and  $\rightsquigarrow^*$ . The accepted language, denoted by  $L(M)$ , can be defined by the final accepting configurations that can be reached from the initial one: A double strand  $\binom{w_1}{w_2}$  is accepted by a sensing  $5' \rightarrow 3'$  WK automaton with jumping feature  $M$  if and only if  $\binom{\varepsilon}{w_2}(q_0, \oplus)\binom{w_1}{\varepsilon} \rightsquigarrow^* \binom{w'_1}{\varepsilon}(q_f, \ominus)\binom{\varepsilon}{w'_2}$ , for  $q_f \in F$ , where  $w'_1 = a_1 a_2 \dots a_n$ ,  $w'_2 = b_1 b_2 \dots b_n$ ,  $a_i, b_i \in (V \cup \{\#\})$ , and either  $a_i = \#$  or  $b_i = \#$ , for all  $i = 1, \dots, n$ , for some  $n \geq 0$ .

From a practical point of view, however, this definition is not ideal. The automaton can easily end up in a configuration that cannot yield accepting results, and the correct positions of auxiliary symbols  $\#$  need to be checked separately at the end of the process. Therefore, we present a modified definition that has the jumping mechanism integrated more into its structure. We are also using a simplification for complementary pairs and treat them as single letters. Such a change has no effect on the accepting power, and this form of input is more natural for formal language theory.

**Definition 3.2** A jumping  $5' \rightarrow 3'$  WK automaton is a quintuple  $M = (V, Q, q_0, F, \delta)$ , where  $V$ ,  $Q$ ,  $q_0$ , and  $F$  are the same as in WK automata,  $V \cap \{\#\} = \emptyset$ , the state transition function  $\delta: (Q \times V^* \times V^* \times D) \rightarrow 2^Q$ , where  $D = \{\oplus, \ominus\}$  indicates the mutual position of heads, and  $\delta$  assigns a nonempty set only for finitely many quadruples of  $(Q \times V^* \times V^* \times D)$ . A configuration  $(q, s, w_1, w_2, w_3)$  consists of the state  $q$ , the position of heads  $s \in D$ , and the three unprocessed portions of the input tape: (a) before the first head ( $w_1$ ), (b) between the heads ( $w_2$ ), and (c) after the second head ( $w_3$ ). A step of the automaton can be of the following four types: Let  $x, y, u, v, w_2 \in V^*$  and  $w_1, w_3 \in (V \cup \{\#\})^*$ .

- (1)  $\oplus$ -reading:  $(q, \oplus, w_1, xw_2y, w_3) \rightsquigarrow (q', s, w_1\{\#\}^{|x|}, w_2, \{\#\}^{|y|}w_3)$ , where  $q' \in \delta(q, x, y, \oplus)$ , and  $s$  is either  $\oplus$  if  $|w_2| > 0$  or  $\ominus$  in other cases.
- (2)  $\ominus$ -reading:  $(q, \ominus, w_1y, \varepsilon, xw_3) \rightsquigarrow (q', \ominus, w_1, \varepsilon, w_3)$ , where  $q' \in \delta(q, x, y, \ominus)$ . ]
- (3)  $\oplus$ -jumping:  $(q, \oplus, w_1, uw_2v, w_3) \rightsquigarrow (q, s, w_1u, w_2, vw_3)$ , where  $s$  is either  $\oplus$  if  $|w_2| > 0$  or  $\ominus$  in other cases.
- (4)  $\ominus$ -jumping:  $(q, \ominus, w_1\{\#\}^*, \varepsilon, \{\#\}^*w_3) \rightsquigarrow (q, \ominus, w_1, \varepsilon, w_3)$ .

In the standard manner, extend  $\rightsquigarrow$  to  $\rightsquigarrow^n$ , where  $n \geq 0$ ; then, based on  $\rightsquigarrow^n$ , define  $\rightsquigarrow^+$  and  $\rightsquigarrow^*$ . The accepted language, denoted by  $L(M)$ , can be defined by the final accepting configurations that can be reached from the initial one: A string  $w$  is accepted by a jumping  $5' \rightarrow 3'$  WK automaton  $M$  if and only if  $(q_0, \oplus, \varepsilon, w, \varepsilon) \rightsquigarrow^* (q_f, \ominus, \varepsilon, \varepsilon, \varepsilon)$ , for  $q_f \in F$ .

Even though the structure of this definition is considerably different from Definition 3.1, it is not hard to show that both models accept the same family of languages.

**Proposition 3.3** The models of Definitions 3.1 and 3.2 accept the same family of languages.

*Proof. (sketch).* This proposition can be proved by construction (from both sides). Let  $M_1 = (V_1, \rho, Q, q_0, F, \delta_1)$  from Definition 3.1 and  $M_2 = (V_2, Q, q_0, F, \delta_2)$  from Definition 3.2. The states can clearly remain identical. We can define bijection  $\varphi: \rho \rightarrow V_2$ . Let  $\varphi(a_i, a'_i) = x_i$  and  $\varphi(b_i, b'_i) = y_i$ , where  $a_i, a'_i, b_i, b'_i \in V_1$ ,  $(a_i, a'_i), (b_i, b'_i) \in \rho$ ,  $x_i, y_i \in V_2$ , for all  $i = 1, \dots, n$ ,  $n$  is a positive integer. Any  $\delta_1(q, \binom{a_1 \dots a_n}{b'_1 \dots b'_m}, s)$  can be converted into  $\delta_2(q, x_1 \dots x_n, y_1 \dots y_m, s)$ , for some  $n, m \geq 0$ , and vice versa. With this transformation, we reason that both models accept the same inputs. Observe that the reading in the first model marks processed positions in the configuration with the auxiliary symbol  $\#$ , and, at the end, the model checks whether each pair of symbols was read precisely once. On the other hand, the second model allows only correct transitions that do not violate this reading condition. Furthermore, it keeps only unprocessed parts of the input in the configuration. Consequently, the second model requires more types of steps to handle the different stages of the process. Before the heads meet, either the  $\oplus$ -reading

reads some symbols and marks them (with #) for the other head, or the  $\oplus$ -jumping skips some symbols and leaves them for the other head. After the meeting point, either the  $\ominus$ -reading reads remaining symbols that were previously skipped, or the  $\ominus$ -jumping erases marked symbols from the configuration. Besides that, the reading and jumping behave analogically in both models and thus give the same resulting accepting power. A rigorous version of this proof is left to the reader.  $\square$

Hereafter, we primarily use Definition 3.2.

## 4. Examples

To demonstrate the behavior of the automata, we present a few simple examples.

**Example 4.1** *Let us recall that  $L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$  is a well-known non-linear context-free language. We show that, even though the jumping directions in the model are quite restricted, we are able to accept such a language. Consider the following jumping  $5' \rightarrow 3'$  WK automaton*

$$M = (\{a, b\}, \{s\}, s, \{s\}, \delta)$$

with the state transition function  $\delta$ :  $\delta(s, a, b, \oplus) = \{s\}$  and  $\delta(s, a, b, \ominus) = \{s\}$ . Starting from  $s$ ,  $M$  can either utilize the jumping or read simultaneously with both heads (the  $\blacktriangleright$ -head reads  $a$  and the  $\blacktriangleleft$ -head reads  $b$ ), and it always stays in the sole state  $s$ . Now, consider the inputs  $aaabbb$  and  $baabba$ . The former can be accepted by using three  $\oplus$ -readings and one  $\ominus$ -jumping:

$$(s, \oplus, \varepsilon, aaabbb, \varepsilon) \rightsquigarrow (s, \oplus, \#, aabb, \#) \rightsquigarrow (s, \oplus, \#\#, ab, \#\#) \rightsquigarrow (s, \ominus, \#\#\#, \varepsilon, \#\#\#) \rightsquigarrow (s, \ominus, \varepsilon, \varepsilon, \varepsilon).$$

The latter input is more complex and can be accepted by using one  $\oplus$ -jumping, two  $\oplus$ -readings, one  $\ominus$ -jumping, and one  $\ominus$ -reading:

$$(s, \oplus, \varepsilon, baabba, \varepsilon) \rightsquigarrow (s, \oplus, b, aabb, a) \rightsquigarrow (s, \oplus, b\#, ab, \#a) \rightsquigarrow (s, \ominus, b\#\#, \varepsilon, \#\#a) \rightsquigarrow (s, \ominus, b, \varepsilon, a) \rightsquigarrow (s, \ominus, \varepsilon, \varepsilon, \varepsilon).$$

It is not hard to see that, by combining different types of steps, we can accept any input containing the same number of  $a$ 's and  $b$ 's, and thus  $L(M) = L$ .

**Example 4.2** *Consider the following jumping  $5' \rightarrow 3'$  WK automaton*

$$M = (\{a, b\}, \{s\}, s, \{s\}, \delta)$$

with the state transition function  $\delta$ :  $\delta(s, a, b, \oplus) = \{s\}$ . Observe that this is almost identical to Example 4.1, however, we cannot use the  $\ominus$ -reading anymore. Consequently, we also cannot effectively use the  $\oplus$ -jumping because there is no way how to process remaining symbols afterwards. As a result, the accepted language changes to  $L(M) = \{a^n b^n : n \geq 0\}$ .

Lastly, we give a more complex example that uses all parts of the model.

**Example 4.3** Consider the following jumping  $5' \rightarrow 3'$  WK automaton

$$M = (\{a, b, c\}, \{s_0, s_1, s_2\}, s_0, \{s_0\}, \delta)$$

with  $\delta$ :  $\delta(s_0, a, b, \oplus) = \{s_1\}$ ,  $\delta(s_1, \varepsilon, b, \oplus) = \{s_0\}$ ,  $\delta(s_0, c, c, \ominus) = \{s_2\}$ , and  $\delta(s_2, \varepsilon, c, \ominus) = \{s_0\}$ . We can divide the accepting process of  $M$  into two stages. First, before the heads meet, the automaton ensures that for every  $a$  on the left side there are two  $b$ 's on the right side; other symbols are skipped with the jumps. Second, after the heads meet, the automaton checks if the part before the meeting point has double the number of  $c$ 's as the part after the meeting point. Thus,  $L(M) = \{w_1w_2 : w_1 \in \{a, c\}^*, w_2 \in \{b, c\}^*, 2 \cdot |w_1|_a = |w_2|_b, |w_1|_c = 2 \cdot |w_2|_c\}$ .

## 5. General Results

These results cover the general behavior of jumping  $5' \rightarrow 3'$  WK automata without any further restrictions. Let **SWK**, **JWK**, **GJFA**, and **JFA** denote the language families accepted by sensing  $5' \rightarrow 3'$  WK automata, jumping  $5' \rightarrow 3'$  WK automata, general jumping finite automata, and jumping finite automata, respectively.

Due to space constraints, some of our proofs are only sketched.

Considering the previous results on other models that use the jumping mechanism (see [5, 6, 3, 1]), it is a common characteristic that they define language families that are incomparable with the classical families of regular, linear, and context-free languages. On the other hand, sensing  $5' \rightarrow 3'$  WK automata (see [7, 8, 9, 10]) are closely related to the family of linear languages. First, we show that the new model is able to accept all regular and linear languages. Furthermore, the accepting power goes beyond the family of linear languages.

**Lemma 5.1** For every regular language  $L$ , there is a jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L = L(M)$ .

*Proof.* Consider a finite automaton  $N = (V, Q, q_0, F, \delta_1)$  such that  $L(N) = L$ . We can construct the jumping  $5' \rightarrow 3'$  WK automaton  $M = (V, Q, q_0, F, \delta_2)$  where  $\delta_2(q, a, \varepsilon, \oplus) = \delta_1(q, a)$  for all  $q \in Q$ ,  $a \in (V \cup \{\varepsilon\})$ . Observe that with such a state transition function the  $\oplus$ -reading steps always look like this:  $(q, \oplus, w_1, aw_2, w_3) \rightsquigarrow (q', s, w_1\{\#\}^{|a|}, w_2, w_3)$ , where  $q' \in \delta_2(q, a, \varepsilon, \oplus)$ ,  $w_2 \in V^*$ ,  $w_1, w_3 \in (V \cup \{\#\})^*$ , and  $s$  is either  $\oplus$  if  $|w_2| > 0$  or  $\ominus$  in other cases. There are no possible  $\ominus$ -reading steps. The  $\oplus$ -jumping can be potentially used to skip some symbols before the heads meet; nonetheless, the resulting configuration will be in the form  $(q, s, w_1, w_2, w_3)$  where  $\text{alph}(w_1w_3) \cap V \neq \emptyset$ . Since there is no way how to read such symbols in  $w_1$  and  $w_3$ , the configuration cannot yield an accepting result. Consequently, any input string will be read in  $M$  the same way as in  $N$  (the remaining  $\#$ 's will be erased with the  $\ominus$ -jumping afterwards). Thus,  $L(M) = L(N) = L$ .  $\square$

**Lemma 5.2** For every sensing  $5' \rightarrow 3'$  WK automaton  $M_1$ , there is a jumping  $5' \rightarrow 3'$  WK automaton  $M_2$  such that  $L(M_1) = L(M_2)$ .

*Proof.* This can be proved by construction. Consider any sensing  $5' \rightarrow 3'$  WK automaton  $M_1$ . A direct conversion would be complicated, however, let us recall that **LIN** = **SWK** (see Theorem 2 in [9]). Consider a linear grammar  $G = (N, T, S, P)$  such that  $L(G) = L(M_1)$ . We can construct the jumping  $5' \rightarrow 3'$  WK automaton  $M_2$  such that  $L(M_2) = L(G)$ . Assume that  $q_f \notin (N \cup T)$ . Define  $M_2 = (T, N \cup \{q_f\}, S, \{q_f\}, \delta)$ , where  $B \in \delta(A, u, v, \oplus)$  if  $A \rightarrow uBv \in P$  and  $q_f \in \delta(A, u, \varepsilon, \oplus)$  if  $A \rightarrow u \in P$  ( $A, B \in N, u, v \in T^*$ ). By the same reasoning as in the proof of Lemma 5.1, only the  $\oplus$ -reading can be effectively used before the heads meet. Consequently, it can be easily seen that  $M_2$  reads all symbols in the same fashion as  $G$  generates them. Moreover, the heads of  $M_2$  can meet each other with the accepting state  $q_f$  if and only if  $G$  can finish the generation process with a rule  $A \rightarrow u$ . Thus,  $L(M_2) = L(G) = L(M_1)$ .  $\square$

**Theorem 5.3**  $LIN = SWK \subset JWK$ .

*Proof.* **SWK**  $\subseteq$  **JWK** follows from Lemma 5.2. **LIN** = **SWK** was proved in [9]. **JWK**  $\not\subseteq$  **LIN** follows from Example 4.1.  $\square$

The next two characteristics follow from the previous results.

**Theorem 5.4** *Jumping  $5' \rightarrow 3'$  WK automata without  $\ominus$ -reading steps accept linear languages.*

*Proof.* Consider any jumping  $5' \rightarrow 3'$  WK automaton  $M = (V, Q, q_0, F, \delta)$  that has no possible  $\ominus$ -reading steps. Expanding the reasoning in the proof of Lemma 5.2, if there are no possible  $\ominus$ -reading steps, the  $\oplus$ -jumping cannot be effectively used, and we can construct a linear grammar that generates strings in the same fashion as  $M$  reads them. Define the linear grammar  $G = (Q, V, q_0, R)$ , where  $R$  is constructed in the following way: (1) For each  $p \in \delta(q, u, v, \oplus)$ , add  $q \rightarrow upv$  to  $R$ . (2) For each  $f \in F$ , add  $f \rightarrow \varepsilon$  to  $R$ . Clearly,  $L(G) = L(M)$ .  $\square$

**Proposition 5.5** *The language family accepted by double-jumping finite automata that perform right-left and left-right jumps (see [1]) is strictly included in **JWK**.*

*Proof.* First, Theorem 3.18 in [1] shows that jumping finite automata that perform right-left and left-right jumps accept the same family of languages. Second, Theorem 3.7 in [1] shows that this family is strictly included in **LIN**. Finally, Theorem 5.3 shows that **LIN** is strictly included in **JWK**.  $\square$

Even though the model is able to accept some non-linear languages, the jumping directions of the heads are quite restricted compared to general jumping finite automata. Consequently, there are some languages accepted by jumping  $5' \rightarrow 3'$  WK automata and general jumping finite automata that cannot be accepted with the other model.

**Lemma 5.6** *There is no jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(M) = \{a^n b^n c^n : n \geq 0\}$ .*

*Proof. (sketch).* It is clear that with a finite memory the automaton can remember only a finite amount of symbols that were already processed. Intuitively, therefore, for a long enough input the automaton must be able to repeatedly perform some sequence of steps in some phase that reads a correlated number of  $a$ 's,  $b$ 's, and  $c$ 's. Nonetheless, in any such a sequence, some head

has to jump over a portion of the unprocessed input to read  $b$ 's, and this head can no longer read the symbols it skipped. The skipped portion has to contain either all remaining  $a$ 's or  $c$ 's. These  $a$ 's or  $c$ 's can still be read with the other head, but, in order to reach them, this head would have to skip the other remaining portion of the input. This portion would have to contain all remaining  $b$ 's and also either all remaining  $c$ 's or  $a$ 's. Consequently, there cannot be a repeatable sequence of steps that reads distinct symbols from three different places.  $\square$

**Lemma 5.7** *There is no jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(M) = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$ .*

*Proof. (sketch).* Assume that there is a jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(M) = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$ . Intuitively, such an automaton must be able to properly check the number of symbols in any input  $w = a^n b^n c^n$ , where  $n$  is a positive integer. However, the argument in the sketch of the proof of Lemma 5.6 shows that it is not possible.  $\square$

**Proposition 5.8**  *$JWK$  is incomparable with  $GJFA$  and  $JFA$ .*

*Proof.* The language  $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$  from Example 4.1 and the language  $\{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$  from Lemma 5.7 are accepted with (general) jumping finite automata (see Example 5 in [5]). The language  $\{a^n b^n : n \geq 0\}$  from Example 4.2 is not accepted with (general) jumping finite automata (see Lemma 19 in [5]).  $\square$

The last group of results compares the accepting power of the model with the families of context-sensitive and context-free languages.

**Theorem 5.9**  $JWK \subseteq CS$ .

*Proof.* Clearly, the use of two heads and the jumping behavior can be simulated by linear bounded automata, so  $JWK \subseteq CS$ . From Lemma 5.6,  $CS - JWK \neq \emptyset$ .  $\square$

**Lemma 5.10** *There are some non-context-free languages accepted by jumping  $5' \rightarrow 3'$  WK automata.*

*Proof.* Consider the following jumping  $5' \rightarrow 3'$  WK automaton

$$M = (\{a, b, c, d\}, \{s\}, s, \{s\}, \delta)$$

with the state transition function  $\delta$ :  $\delta(s, a, c, \oplus) = \{s\}$  and  $\delta(s, d, b, \ominus) = \{s\}$ . The accepting process has two stages. First, before the heads meet, the automaton reads the same number of  $a$ 's and  $c$ 's. Second, after the heads meet, the automaton reads the same number of  $d$ 's and  $b$ 's. Thus,  $L(M) = \{w_1 w_2 : w_1 \in \{a, b\}^*, w_2 \in \{c, d\}^*, |w_1|_a = |w_2|_c, |w_1|_b = |w_2|_d\}$ .

Proof by contradiction. Assume that  $L(M)$  is a context-free language. The family of context-free languages is closed under intersection with regular sets. Let  $K = L(M) \cap \{a\}^* \{b\}^* \{c\}^* \{d\}^*$ . Clearly, there are some strings in  $L(M)$  that satisfy this forced order of symbols. Furthermore, they all have the proper correlated numbers of these symbols. Consequently,  $K = \{a^n b^m c^n d^m : n, m \geq 0\}$ . However,  $K$  is a well-known non-context-free language (see Chapter 3.1 in [12]). That is a contradiction with the assumption that  $L(M)$  is a context-free language. Therefore,  $L(M)$  is a non-context-free language.  $\square$

**Lemma 5.11** *There is no jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(M) = \{a^n b^n c^m d^m : n, m \geq 0\}$ .*

*Proof. (sketch).* It is clear that with a finite memory the automaton can remember only a finite amount of symbols that were already processed. As it is known from finite automata, a single head alone cannot recognize strings  $a^n b^n$  or  $c^m d^m$ . Intuitively, therefore, the automaton has to use both heads to process such a string. But in order to do so, some head has to jump over the other part ( $a^n b^n$  or  $c^m d^m$ ). However, since the heads cannot travel back on the tape, there is no way how to use both heads to process both parts.  $\square$

**Theorem 5.12**  *$\mathbf{JWK}$  and  $\mathbf{CF}$  are incomparable.*

*Proof.*  $\mathbf{JWK} \not\subseteq \mathbf{CF}$  follows from Lemma 5.10.  $\mathbf{CF} \not\subseteq \mathbf{JWK}$  follows from Lemma 5.11. Lastly,  $\mathbf{LIN} \subset \mathbf{JWK}$  and  $\mathbf{LIN} \subset \mathbf{CF}$ .  $\square$

## 6. Results on Restricted Variations

In this section, we compare the accepting power of unrestricted and restricted variations of jumping  $5' \rightarrow 3'$  WK automata. This paper considers the same standard restrictions as they are defined for Watson-Crick finite automata. Since these restrictions regulate only the state control and reading steps of the automaton, the jumping is not affected in any way.

Let  $\mathbf{JWK}$  denote the language family accepted by jumping  $5' \rightarrow 3'$  WK automata. We are using prefixes  $\mathbf{N}$ ,  $\mathbf{F}$ ,  $\mathbf{S}$ ,  $\mathbf{1}$ ,  $\mathbf{NS}$ ,  $\mathbf{FS}$ ,  $\mathbf{N1}$ , and  $\mathbf{F1}$  to specify the restricted variations of jumping  $5' \rightarrow 3'$  WK automata and appropriate language families.

In the field of DNA computing, the empty string/empty sequence usually does not belong to any language because it does not refer to a molecule. This paper is not so strict and thus considers the empty string as a possible valid input. Nonetheless, the following proofs are deliberately based on more complex inputs to mitigate the impact of the empty string on the results.

Note that there are some inherent inclusions between language families based on the application of restrictions on the model. Additionally, several other basic relations can be established directly from the restriction definitions:

**Lemma 6.1** *The following relations hold: (i)  $\mathbf{N JWK} \subseteq \mathbf{F JWK}$ ; (ii)  $\mathbf{1 JWK} \subseteq \mathbf{S JWK}$ ; (iii)  $\mathbf{F1 JWK} \subseteq \mathbf{FS JWK}$ ; (iv)  $\mathbf{N1 JWK} \subseteq \mathbf{NS JWK}$ ; (v)  $\mathbf{NS JWK} \subseteq \mathbf{FS JWK}$ ; (vi)  $\mathbf{N1 JWK} \subseteq \mathbf{F1 JWK}$ .*

*Proof.* These results follow directly from the definitions since the stateless restriction ( $\mathbf{N}$ ) is a special case of the all-final restriction ( $\mathbf{F}$ ) and the 1-limited restriction ( $\mathbf{1}$ ) is a special case of the simple restriction ( $\mathbf{S}$ ).  $\square$

Due to space constraints, we present only a quick overview of the results.



**Theorem 6.2**  $S JWK = JWK$ .

*Proof. (idea).* Any general reading step can be replaced with two simple reading steps and a new auxiliary state that together accomplish the same action.  $\square$

**Example 6.3** Consider the following jumping  $5' \rightarrow 3'$  WK automaton  $M = (\{a, b, c\}, \{s, f\}, s, \{f\}, \delta)$  with the state transition function  $\delta$ :

$$\begin{aligned} \delta(s, a, b, \oplus) &= \{s\}, & \delta(f, a, b, \oplus) &= \{f\}, & \delta(f, a, b, \ominus) &= \{f\}, \\ \delta(s, cc, \varepsilon, \oplus) &= \{f\}, & \delta(s, \varepsilon, cc, \oplus) &= \{f\}. \end{aligned}$$

It is clear that the first three transitions mimic the behavior of Example 4.1. The other two transitions ensure that the input is accepted only if it also contains precisely one substring  $cc$ . Therefore,  $L(M) = \{w_1ccw_2 : w_1, w_2 \in \{a, b\}^*, |w_1w_2|_a = |w_1w_2|_b\}$ .

**Theorem 6.4**  $1 JWK \subset JWK$ .

*Proof. (idea).* There is no  $1$  jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(M) = \{w_1ccw_2 : w_1, w_2 \in \{a, b\}^*, |w_1w_2|_a = |w_1w_2|_b\}$ .  $\square$

**Example 6.5** Consider the following  $1$  jumping  $5' \rightarrow 3'$  WK automaton  $M = (\{a, b\}, \{s, p\}, s, \{s\}, \delta)$  with the state transition function  $\delta$ :

$$\begin{aligned} \delta(s, a, \varepsilon, \oplus) &= \{p\}, & \delta(p, \varepsilon, b, \oplus) &= \{s\}, \\ \delta(s, a, \varepsilon, \ominus) &= \{p\}, & \delta(p, \varepsilon, b, \ominus) &= \{s\}. \end{aligned}$$

It is not hard to see that the resulting behavior is similar to Example 4.1. The automaton now reads  $a$ 's and  $b$ 's with separate steps and uses one auxiliary state that is not final. Consequently,  $L(M) = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$ .

**Theorem 6.6**  $LIN \subset 1 JWK$ .

*Proof. (idea).* For every linear grammar  $G$ , there is a  $1$  jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(G) = L(M)$ . Example 6.5 shows that  $1 JWK \not\subset LIN$ .  $\square$

**Theorem 6.7**  $F JWK \subset JWK$ .

*Proof. (idea).* There is no  $F$  jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(M) = \{ca^n cb^n c : n \geq 0\} \cup \{\varepsilon\}$ .  $\square$

**Example 6.8** Consider the following  $F$  (in fact, even  $N$ ) jumping  $5' \rightarrow 3'$  WK automaton  $M = (\{a, b, c\}, \{s\}, s, \{s\}, \delta)$  with the state transition function  $\delta$ :

$$\begin{aligned} \delta(s, a, b, \oplus) &= \{s\}, & \delta(s, a, b, \ominus) &= \{s\}, \\ \delta(s, cc, \varepsilon, \oplus) &= \{s\}, & \delta(s, \varepsilon, cc, \oplus) &= \{s\}. \end{aligned}$$

This is a slightly modified version of Example 6.3 where the substring  $cc$  can occur arbitrarily many times. Therefore,  $L(M) = \{w \in \{a, b, cc\}^* : |w|_a = |w|_b\}$ .  $L(M) \notin 1 JWK$ .

**Theorem 6.9**  $NJWK \subset FJWK$ .

*Proof. (idea).* From Lemma 6.1,  $NJWK \subseteq FJWK$ . There is no **N** jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that  $L(M) = \{\varepsilon, a\}$ .  $\square$

**Proposition 6.10**  $FSJWK \subset FJWK$ .

*Proof. (idea).* There is no **FS** jumping  $5' \rightarrow 3'$  WK automaton  $M$  such that that  $L(M) = \{cca^ncc : n \geq 0\} \cup \{\varepsilon\}$ .  $\square$

**Example 6.11** Consider the following **FS** jumping  $5' \rightarrow 3'$  WK automaton  $M = (\{a, b, c\}, \{s, p\}, s, \{s, p\}, \delta)$  with the state transition function  $\delta$ :

$$\begin{aligned} \delta(s, a, \varepsilon, \oplus) &= \{p\}, & \delta(p, \varepsilon, b, \oplus) &= \{s\}, \\ \delta(s, a, \varepsilon, \ominus) &= \{p\}, & \delta(p, \varepsilon, b, \ominus) &= \{s\}, \\ \delta(s, cc, \varepsilon, \oplus) &= \{s\}, & \delta(s, \varepsilon, cc, \oplus) &= \{s\}, \\ \delta(p, cc, \varepsilon, \oplus) &= \{p\}, & \delta(p, \varepsilon, cc, \oplus) &= \{p\}. \end{aligned}$$

As a result,  $L(M) = \{w \in \{a, b, cc\}^* : |w|_a = |w|_b \text{ or } |w|_a = |w|_b + 1\}$ .

This automaton is just a combination of previous approaches from Examples 6.5 and 6.8. Note that  $L(M)$  resembles the resulting language of Example 6.8.

**Proposition 6.12**  $F1JWK \subset FSJWK$ .

*Proof. (idea).* There is no **F1** jumping  $5' \rightarrow 3'$  WK automaton that accepts  $\{aa\}^*$ .  $\square$

**Corollary 6.13**  $F1JWK \subset 1JWK$ .**Theorem 6.14**  $NSJWK \subset REG$ .

*Proof. (idea).* For any **NS** jumping  $5' \rightarrow 3'$  WK automaton we can construct a finite automaton that accepts the same language.  $\square$

**Proposition 6.15**  $N1JWK \subset NSJWK$ .

*Proof.* This proof is analogous to that of Proposition 6.12.  $\square$

**Corollary 6.16** The following relations hold: (i)  $NSJWK \subset NJWK$ ; (ii)  $NSJWK \subset FSJWK$ ; (iii)  $N1JWK \subset F1JWK$ .

All the obtained results comparing the accepting power of unrestricted and restricted variations of jumping  $5' \rightarrow 3'$  WK automata are summarized in Figure 1.

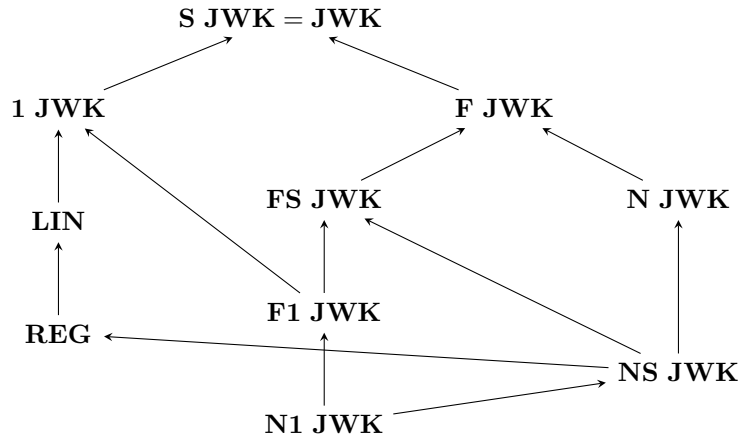


Figure 1: A hierarchy of language families closely related to the unrestricted and restricted variations of jumping  $5' \rightarrow 3'$  WK automata is shown. If there is an arrow from family  $X$  to family  $Y$  in the figure, then  $X \subset Y$ . Furthermore, if there is no path (following the arrows) between families  $X$  and  $Y$ , then  $X$  and  $Y$  are incomparable.

## 7. Conclusion

The results clearly show that, with the addition of the jumping mechanism into the model, the accepting power was increased above sensing  $5' \rightarrow 3'$  WK automata. The model is now able to accept some non-linear and even some non-context-free languages. On the other hand, the jumping movement of the heads is restricted compared to jumping finite automata, and this limits its capabilities to accept languages that require discontinuous information processing. Considering the comparison with full-reading sensing  $5' \rightarrow 3'$  WK automata, the results are not yet clear. However, we know that there are some languages, like  $\{a^n b^n c^n : n \geq 0\}$ , that cannot be accepted by jumping  $5' \rightarrow 3'$  WK automata and that are accepted by full-reading sensing  $5' \rightarrow 3'$  WK automata (see [7, 8, 9]).

If we compare the hierarchies of language families related to the restricted variations of jumping  $5' \rightarrow 3'$  WK automata and sensing  $5' \rightarrow 3'$  WK automata (see [8, 9, 10]), there are several noticeable remarks. Most importantly, the 1-limited restriction (**1**) has a negative impact on the accepting power, which is usually not the case in sensing  $5' \rightarrow 3'$  WK automata. In parts where several restrictions are combined together, the hierarchy structure resembles sensing  $5' \rightarrow 3'$  WK automata without the sensing distance. Nonetheless, almost all restricted variations, with the exception of **NS** and **N1**, are still able to accept some non-linear languages.

Lastly, the reader may notice that the  $\ominus$ -jumping can be used only in situations where it is forced by the current configuration. Furthermore, jumping finite automata usually immediately erase symbols from the configuration and do not use the auxiliary symbol  $\#$ . It is therefore a question whether this part could be safely removed from the model. Without it, the conversion from Definition 3.1 cannot be straightforward, and it is not clear whether the accepting power remains identical. Observe that, if we remove  $\#$ 's, the configuration can create new connected strings that were not in the original input and for which there can be a possible  $\ominus$ -reading step.

## Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602; the TAČR grant TE01020415; and the BUT grant FIT-S-17-3964. The authors thank all the anonymous referees for their useful comments and suggestions.

## References

- [1] R. KOCMAN, Z. KŘIVKA, A. MEDUNA, On double-jumping finite automata. In: H. BORDIHN, R. FREUND, B. NAGY, GY. VASZIL (eds.), *Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016)*. books@ocg.at 321, Österreichische Computer Gesellschaft, Wien, 2016, 195–210.
- [2] R. KOCMAN, A. MEDUNA, On parallel versions of jumping finite automata. In: J. JANECH, J. KOSTOLNY, T. GRATKOWSKI (eds.), *Proceedings of the 2015 Federated Conference on Software Development and Object Technologies (SDOT 2015)*. Advances in Intelligent Systems and Computing 511, Springer, 2017, 142–149.
- [3] Z. KŘIVKA, A. MEDUNA, Jumping grammars. *International Journal of Foundations of Computer Science* 26 (2015) 6, 709–731.
- [4] A. MEDUNA, *Automata and Languages: Theory and Applications*. Springer, 2000.
- [5] A. MEDUNA, P. ZEMEK, Jumping finite automata. *International Journal of Foundations of Computer Science* 23 (2012) 7, 1555–1578.
- [6] A. MEDUNA, P. ZEMEK, *Regulated Grammars and Automata*. Springer, 2014.
- [7] B. NAGY, On  $5' \rightarrow 3'$  sensing Watson-Crick finite automata. In: M. H. GARZON, H. YAN (eds.), *13th International Meeting on DNA Computing, DNA13, Memphis, TN, USA, June 4-8, 2007, Revised Selected Papers*. Lecture Notes in Computer Science 4848, Springer, 2008, 256–262.
- [8] B. NAGY,  $5' \rightarrow 3'$  sensing Watson-Crick finite automata. In: G. FUNG (ed.), *Sequence and Genome Analysis II – Methods and Applications*. iConcept Press, 2010, 39–56.
- [9] B. NAGY, On a hierarchy of  $5' \rightarrow 3'$  sensing Watson-Crick finite automata languages. *Journal of Logic and Computation* 23 (2013) 4, 855–872.
- [10] B. NAGY, S. PARCHAMI, H. MIR-MOHAMMAD-SADEGHI, A new sensing  $5' \rightarrow 3'$  Watson-Crick automata concept. In: E. CSUHAI-VARJÚ, P. DÖMÖSI, GY. VASZIL (eds.), *Proceedings 15th International Conference on Automata and Formal Languages (AFL 2017)*. EPTCS 252, Open Publishing Association, 2017, 195–204.
- [11] GH. PĂUN, G. ROZENBERG, A. SALOMAA, *DNA Computing: New Computing Paradigms*. Springer-Verlag Berlin Heidelberg, 1998.
- [12] G. ROZENBERG, A. SALOMAA, *Handbook of Formal Languages, Vol. 2: Linear Modeling: Background and Application*. Springer-Verlag, 1997.
- [13] D. WOOD, *Theory of Computation: A Primer*. Addison-Wesley, Boston, 1987.

# TWO-SIDED LOCALLY TESTABLE LANGUAGES

Martin Kutrib<sup>(A)</sup>      Friedrich Otto<sup>(B)</sup>

<sup>(A)</sup>Institut für Informatik, Universität Giessen,  
Arndtstr. 2, 35392 Giessen, Germany  
kutrib@informatik.uni-giessen.de

<sup>(B)</sup>Department of Computer Science,  
Faculty of Mathematics and Physics,  
Charles University, 118 00 Praha 1  
otto@ktiml.mff.cuni.cz

## **Abstract**

*We extend the two-sided strictly testable languages to the two-sided testable languages, showing that for each integer  $k \geq 1$  and each symmetric binary relation  $R$  on  $\Sigma^k$ , the family  $2LT_R(k)$  of  $k$ - $R$ -testable languages is obtained as a special kind of Boolean closure of the family of two-sided strictly  $k$ -testable languages. We further study closure and non-closure properties and prove that all two-sided locally testable languages are even linear languages.*

## **1. Introduction**

In 1971 McNaughton and Papert introduced the strictly locally testable and the locally testable languages [5]. These languages have received much attention in the literature because of their elegance and simplicity. A language  $L$  over  $\Sigma$  is called strictly locally testable, if it is strictly  $k$ -testable for some integer  $k \geq 1$ , which means that there are sets  $A, B, C$  of words of length  $k$  over  $\Sigma$  such that a word  $w$  of length at least  $k$  belongs to  $L$  if and only if its prefix  $P_k(w)$  of length  $k$  belongs to  $A$ , its suffix  $S_k(w)$  of length  $k$  belongs to  $B$ , and the set  $I_k(w)$  of all its inner factors of length  $k$  is a subset of  $C$ . Thus, in order to check that a word  $w$  belongs to  $L$ , an automaton with a window of size  $k$  can be used that simply moves its window from left to right across  $w$ . For a language  $L$  on  $\Sigma$  to be locally testable it is required that it is  $k$ -testable for some  $k \geq 1$ , which means that membership of a word  $w$  of length at least  $k$  only depends on its prefix  $P_k(w)$ , its suffix  $S_k(w)$  and its set of inner factors  $I_k(w)$ . In fact, the set of  $k$ -testable languages is just the Boolean closure of the set of strictly  $k$ -testable languages [5].

The automaton for a (strictly) locally testable language scans its input simply from left to right, just as classical models of automata, like finite-state automata or pushdown automata. But already early on researchers were also interested in devices with the ability to scan their inputs in a more flexible way. This has been achieved in several ways, for example, by two-way head motion, more than one input head, or a combination thereof. One of the easiest

combination are two heads, one scanning the input from left to right and the other scanning it from right to left. If this two-head extension is applied to finite-state automata, a machine model is obtained that characterizes the linear context-free languages [7]. Pushdown automata with this two-head extension accept linguistically important languages, describing a family of mildly context-sensitive languages [6]. What happens if the two-head extension is applied to the machine model for the (strictly) locally testable languages that memorizes factors of a certain length? Here it makes no sense that the heads move independently of each other, as then we would just get the intersection of two (strictly) locally testable languages, which is itself a (strictly) locally testable language. Thus, the movements of the two heads must be synchronized and the factors read concurrently must be correlated in some way. This idea has been formalized in [2], leading to the notion of two-sided strictly locally testable languages.

Here we extend this notion to the two-sided locally testable languages, just as the strictly locally testable languages were extended to the locally testable languages in [5]. A language  $L$  over  $\Sigma$  is called two-sided locally testable, if there exist an integer  $k \geq 1$  and a symmetric binary relation  $R$  on  $\Sigma^k$  such that  $L$  is  $k$ - $R$ -testable. This means that all words  $w \in L$  of length at least  $k$  are  $R$ -symmetric (see Section 2 for the definition), and that it only depends on the prefix  $P_k(w)$ , the suffix  $S_k(w)$ , and the set  $I_k(w)$  of inner factors of length  $k$  of the word  $w$  whether  $w$  belongs to  $L$ . We will see that the family of  $k$ - $R$ -testable languages is the closure of the strictly  $k$ - $R$ -testable languages under the operations of union, intersection, and  $R$ -complementation, where the latter only considers  $R$ -symmetric words of length at least  $k$ . Further, the family of all two-sided locally testable languages is closed under intersection and  $R$ -complementation, but it is not closed under union. Also we consider closure under the operations of reversal, concatenation, Kleene star, length-preserving homomorphisms, length-preserving inverse homomorphisms, and non-erasing inverse homomorphisms.

Concerning the expressive capacity of two-sided locally testable languages, we prove that they are contained in the even linear languages introduced in [1], extending the corresponding result on two-sided strictly locally testable languages from [2]. On the other hand, this language family is incomparable under inclusion to the regular, the deterministic linear, the deterministic context-free, and the Church-Rosser languages (see [4] for the latter). Finally, we even separate the two-sided  $k$ -testable languages from the  $k$ -testable languages by regular example languages.

The paper is structured as follows. After giving the necessary definitions in Section 2, we present the aforementioned presentation of  $k$ - $R$ -testable languages in Section 3, where we also derive the various closure and non-closure properties described above. In Section 4 we investigate the expressive capacity of two-sided locally testable languages and discuss decidability results. The paper closes with a short summary and some open problems for future work.

## 2. Definitions and Preliminaries

For a finite alphabet  $\Sigma$ , we use  $\Sigma^*$  ( $\Sigma^+$ ) to denote the set of all (nonempty) words, and we use  $\lambda$  to denote the empty word. For  $k \geq 0$  we write  $\Sigma^{\leq k}$  for the set of all words of lengths at most  $k$ ,  $\Sigma^k$  for the set of all words of length  $k$ , and  $\Sigma^{\geq k}$  for the set of all words of length at

least  $k$ . For a word  $w \in \Sigma^*$ ,  $|w|$  is used to denote its length. If  $w = a_1 a_2 \cdots a_n$  with  $n \geq 1$  and  $a_1, a_2, \dots, a_n \in \Sigma$ , then for  $1 \leq i \leq n$  and  $1 \leq k \leq n - i$ ,  $w[i, k]$  denotes the factor of length  $k$  of  $w$  starting at position  $i$ , that is,  $w[i, k] = a_i a_{i+1} \cdots a_{i+k-1}$ . Also we denote  $w[i, 1]$  by  $w[i]$ .

Let  $P_k(w)$  and  $S_k(w)$  be the prefix and the suffix of length  $k$  of a word  $w$ , respectively, that is,  $P_k(w) = w[1, k]$  and  $S_k(w) = w[|w| - k + 1, |w|]$ . Further, let  $I_k(w)$  be the set of all factors of length  $k$  of  $w$  except the prefix and the suffix, that is,

$$I_k(w) = \{ u \mid |u| = k \text{ and } \exists x, y \in \Sigma^+ : w = xuy \} = \{ w[i, k] \mid 2 \leq i \leq |w| - k \}.$$

These are defined only for  $|w| \geq k$ . If  $|w| = k$ , then  $P_k(w) = S_k(w) = w$ , while  $I_k(w)$  is empty, whenever  $|w| \leq k + 1$ .

Let  $k$  be a positive integer, and let  $A, B, C \subseteq \Sigma^k$ . By  $L(A, B, C)$  we denote the language  $L(A, B, C) = \{ w \in \Sigma^* \mid |w| \geq k, P_k(w) \in A, S_k(w) \in B, \text{ and } I_k(w) \subseteq C \}$ . A language  $L \subseteq \Sigma^*$  is said to be *strictly  $k$ -testable* if there exist finite sets  $A, B, C \subseteq \Sigma^k$  such that  $L \cap \Sigma^{\geq k} = L(A, B, C)$ . We will say that a language  $L$  is *strictly locally testable* if it is strictly  $k$ -testable for some  $k > 0$ . As a strictly  $k$ -testable language is obtained from the language  $L(A, B, C)$  by possibly adding some words of length at most  $k - 1$ , a triple  $(A, B, C)$  can be used as a basis for several different strictly  $k$ -testable languages.

A natural extension of strictly locally testable languages are the *two-sided strictly locally testable* languages studied in [2]. Let  $k$  be a positive integer. A *two-sided strictly  $k$ -testable* language  $L \subseteq \Sigma^*$  is given through a strictly  $k$ -testable language  $H$  presented by a triple  $(A, B, C)$  and a (finite) symmetric binary relation  $R \subseteq \Sigma^k \times \Sigma^k$ . Now a word  $w \in \Sigma^{\geq k}$  belongs to  $L$  if  $w \in H$  and if, for all indices  $i \in \{1, 2, \dots, |w| - k + 1\}$ ,  $(w[i, k], w[|w| + 2 - k - i, k]) \in R$ . We write  $L(A, B, C, R)$  for the language  $L \cap \Sigma^{\geq k}$  of all words of  $L$  that have length at least  $k$ . Again, a two-sided strictly  $k$ -testable language is obtained from the language  $L(A, B, C, R)$  by possibly adding some words of length at most  $k - 1$ . Thus, a four-tuple  $(A, B, C, R)$  can be used as a basis for several different two-sided strictly  $k$ -testable languages. Finally, a language is *two-sided strictly locally testable* if it is two-sided strictly  $k$ -testable for some  $k \geq 1$ .

**Example 2.1** Let  $\Sigma = \{a, b\}$ , and let  $A = \{a, b\} = B = C$ . Then the triple  $(A, B, C)$  defines the strictly 1-testable language  $L = L(A, B, C) = \Sigma^+$ . Now let  $R \subset \Sigma \times \Sigma$  be defined as  $R = \{(a, b), (b, a)\}$ , and let  $L' = L(A, B, C, R)$  be the resulting two-sided strictly 1-testable language. Then a word  $w \in \Sigma^+$  belongs to  $L'$  if and only if, for all  $i = 1, 2, \dots, |w|$ , the  $i$ -th letter from the left differs from the  $i$ -th letter from the right. Thus,  $L'$  is not even regular, as  $L' \cap (a^* \cdot b^*) = \{ a^n b^n \mid n \geq 1 \}$  holds. Hence,  $L'$  is in particular not strictly locally testable. ■

So far, the membership of a word  $w$  is tested by checking whether its prefix, suffix, and its factors of length  $k$  do belong to sets of allowed prefixes, suffixes, and factors. In addition, the occurring prefix and suffix as well as the occurring factors have to be in relation. In particular, the test does not depend on whether a factor appears or does not appear. Nor is it possible to raise conditions like ‘if factor  $x$  appears, then factor  $y$  has to appear as well.’ A generalization that addresses such issues is the family of locally testable languages.

A language  $L \subseteq \Sigma^*$  is  $k$ -testable for some  $k \geq 1$  if the following conditions are met [5]:

$$\forall x, y \in \Sigma^{\geq k}: \text{ if } P_k(x) = P_k(y) \wedge S_k(x) = S_k(y) \wedge I_k(x) = I_k(y), \text{ then } (x \in L \iff y \in L).$$

Observe that also here the definition says nothing about the words of length at most  $k-1$  in  $L$ . A language is called *locally testable* if it is  $k$ -testable for some  $k \geq 1$ .

Here we propose a generalization of these notions to the two-sided context.

**Definition 2.2** Let  $k \geq 1$ , let  $\Sigma$  be an alphabet, and let  $R \subseteq \Sigma^k \times \Sigma^k$  be a symmetric relation.

1. By  $\Sigma_R^{\geq k}$  we denote the set of all words  $w \in \Sigma^*$  that satisfy the following condition:

$$|w| \geq k \text{ and } \forall i \in \{1, 2, \dots, |w| - k + 1\}, (w[i, k], w[|w| + 2 - k - i, k]) \in R,$$

that is,  $w$  is of length at least  $k$  and the factor  $w[i, k]$  of  $w$  of length  $k$  starting at position  $i$  and the factor  $w[|w| + 2 - k - i, k]$  of  $w$  of length  $k$  starting at position  $|w| + 2 - k - i$  are in relation  $R$ , for all  $i = 1, 2, \dots, |w| - k + 1$ . We call these words the  $R$ -symmetric words.

2. A language  $L \subseteq \Sigma^*$  is  $k$ - $R$ -testable, if  $L \cap \Sigma^{\geq k} \subseteq \Sigma_R^{\geq k}$ , that is, all words in  $L$  that are of length at least  $k$  are  $R$ -symmetric, and the following conditions are met:

$$\forall x, y \in \Sigma_R^{\geq k}: \text{ if } P_k(x) = P_k(y) \wedge S_k(x) = S_k(y) \wedge I_k(x) = I_k(y), \text{ then } (x \in L \iff y \in L).$$

3. A language  $L \subseteq \Sigma^*$  is called two-sided  $k$ -testable if there exists a symmetric binary relation  $R \subseteq \Sigma^k \times \Sigma^k$  such that  $L$  is  $k$ - $R$ -testable.

4. A language  $L \subseteq \Sigma^*$  is called two-sided locally testable if it is two-sided  $k$ -testable for some  $k \geq 1$ .

**Example 2.3** The language  $L = \{ad^n b, ae^n c \mid n \geq 1\}$  is two-sided strictly 2-testable, but it is not two-sided strictly 1-testable. However, this language is two-sided 1-testable. For the symmetric binary relation  $R = \{(a, b), (b, a), (a, c), (c, a), (d, d), (e, e)\}$ , we have

$$\Sigma_R^{\geq 1} = \{w \in \{a, b, c, d, e\}^{\geq 1} \mid \forall i = 1, 2, \dots, |w|: (w[i] = a \text{ iff } w[|w| + 1 - i] \in \{b, c\}) \\ \text{and if } w[i] \in \{d, e\}, \text{ then } w[i] = w[|w| + i - i]\}.$$

Now  $L = \{w \in \Sigma_R^{\geq 1} \mid P_1(w) = a \wedge (I_1(w) = \{d\} \wedge S_1(w) = b) \vee (I_1(w) = \{e\} \wedge S_1(w) = c)\}$ . Actually,  $L$  is the union of the two-sided strictly 1-testable languages  $L_1 = \{ad^n b \mid n \geq 1\}$  and  $L_2 = \{ae^n c \mid n \geq 1\}$ . ■

We will denote the family of strictly  $k$ -testable languages by  $\text{SLT}(k)$  and the class of strictly locally testable languages by  $\text{SLT}$ . For the family of two-sided strictly  $k$ -testable (two-sided strictly locally testable) languages we write  $2\text{SLT}(k)$  ( $2\text{SLT}$ ).

Similarly, we omit the infix  $S$  when we denote the families of non-strictly locally testable languages, that is, we write  $\text{LT}(k)$  for the family of  $k$ -testable languages,  $\text{LT}$  for the family of locally testable languages,  $2\text{LT}_R(k)$  for the family of two-sided  $k$ - $R$ -testable languages,  $2\text{LT}(k)$  for the family of two-sided  $k$ -testable languages, and by  $2\text{LT}$  we denote the family of all two-sided



locally testable languages. If the relation  $R$  is fixed, then we also use the notations  $\text{LT}_R(k)$  for one-sided  $k$ - $R$ -testable languages. Note that a given relation  $R \subseteq \Sigma^k \times \Sigma^k$  implies the constant  $k$ . Therefore, the family  $2\text{LT}_R$  is equal to  $2\text{LT}_R(k)$  and the family  $\text{LT}_R$  is equal to  $\text{LT}_R(k)$ .

In [2] it is shown that  $\text{SLT}(k) \subsetneq 2\text{SLT}(k)$  for all  $k \geq 2$  and that  $2\text{SLT}(k) \subsetneq 2\text{SLT}(k+1)$  for all  $k \geq 1$ . Also it is known that  $\text{LT}(k) \subsetneq \text{LT}(k+1)$  [5]. In Section 4 below we will establish the corresponding result for two-sided  $k$ -testability, but here we already observe the following inclusions.

**Lemma 2.4** *For all  $k \geq 1$ ,  $2\text{LT}(k) \subseteq 2\text{LT}(k+1)$ .*

### 3. Closure Properties

Before we turn to explore the expressive capacity of two-sided locally testable languages in more detail, we study their basic closure properties in order to provide some tools for further proofs. So, in this section we consider the closure properties of the families  $2\text{LT}$ ,  $2\text{LT}(k)$ , and  $2\text{LT}_R(k)$ , for  $k \geq 1$ . We start with technical considerations.

Let  $k \geq 1$ , let  $\Sigma$  be an alphabet, and let  $R \subseteq \Sigma^k \times \Sigma^k$  be a symmetric relation. For all words  $u, v \in \Sigma^k$  and all subsets  $C \subseteq \Sigma^k$ , we define

$$L_R(u, v, C) = \{ w \in \Sigma_R^{\geq k} \mid P_k(w) = u, S_k(w) = v, \text{ and } I_k(w) = C \},$$

and for a  $k$ - $R$ -testable language  $L \subseteq \Sigma^*$ , we define

$$\text{triple}(L) = \{ (u, v, C) \mid u, v \in \Sigma^k \text{ and } C \subseteq \Sigma^k \text{ such that } L_R(u, v, C) \cap L \neq \emptyset \}.$$

If  $(u, v, C) \in \text{triple}(L)$ , then by definition  $L_R(u, v, C) \cap L \neq \emptyset$ . So, there is a word  $w \in \Sigma_R^{\geq k}$  with  $P_k(w) = u$ ,  $S_k(w) = v$ ,  $I_k(w) = C$  that belongs to  $L$  and to  $L_R(u, v, C)$ . Since  $L$  is  $k$ - $R$ -testable, all words  $w' \in \Sigma_R^{\geq k}$  with  $P_k(w') = u$ ,  $S_k(w') = v$ ,  $I_k(w') = C$  belong to  $L$  as well. We conclude that  $L_R(u, v, C) \subseteq L$  if  $(u, v, C) \in \text{triple}(L)$ . Thus, we have the representation  $L = F_L \cup \bigcup_{(u,v,C) \in \text{triple}(L)} L_R(u, v, C)$ , where  $F_L = \{ w \in L \mid |w| \leq k-1 \}$ .

On the other hand, it is easily seen that the union of any finite number of languages of the form  $L_R(u, v, C)$  is a  $k$ - $R$ -testable language.

**Lemma 3.1** *Let  $k \geq 1$ , let  $\Sigma$  be an alphabet, and let  $R \subseteq \Sigma^k \times \Sigma^k$  be a symmetric relation. If  $L = \bigcup_{i=1}^m L_R(u_i, v_i, C_i)$ , then  $L$  belongs to  $2\text{LT}_R(k)$ .*

If  $L \subseteq \Sigma^*$  is a two-sided strictly  $k$ -testable language, then there exist a symmetric binary relation  $R$  on  $\Sigma^k$ , sets  $A, B, C \subseteq \Sigma^k$ , and a finite subset  $F \subseteq \Sigma^{\leq k-1}$  such that

$$L = F \cup \{ w \in \Sigma_R^{\geq k} \mid P_k(w) \in A, S_k(w) \in B, \text{ and } I_k(w) \subseteq C \}.$$

Hence,  $L$  can be written as  $L = F \cup \bigcup_{u \in A, v \in B, C' \subseteq C} L_R(u, v, C')$ , which implies that  $L$  is  $k$ - $R$ -testable. Thus, we have the following inclusions.

**Corollary 3.2** For all  $k \geq 1$ ,  $2SLT(k) \subseteq 2LT(k)$  and  $2SLT \subseteq 2LT$ .

Now we turn to the Boolean operations union and intersection and a variant of the operation of complementation, called  $R$ -complementation. Let  $k \geq 1$ , let  $\Sigma$  be an alphabet, and let  $R \subseteq \Sigma^k \times \Sigma^k$  be a symmetric binary relation. For a  $k$ - $R$ -testable language  $L \subseteq \Sigma^*$ , the  $R$ -complement is the set  $L_R^c = (\Sigma^{\leq k-1} \cap L^c) \cup (\Sigma_R^{\geq k} \cap L^c)$ , that is, it contains all words of length at most  $k-1$  that do not belong to  $L$ , and it contains all  $R$ -symmetric words that do not belong to  $L$ . It is easily seen that  $L^c = L_R^c \cup (\Sigma^{\geq k} \setminus \Sigma_R^{\geq k})$ .

**Proposition 3.3** Let  $k \geq 1$ , let  $\Sigma$  be an alphabet, and let  $R \subseteq \Sigma^k \times \Sigma^k$  be a symmetric relation. Then the family  $2LT_R(k)$  is closed under the Boolean operations intersection and union and under the operation of  $R$ -complementation.

*Proof.* Let  $L, L_1, L_2 \subseteq \Sigma^*$  be  $k$ - $R$ -testable languages.

The union  $L_1 \cup L_2$  is represented by  $F_{L_1} \cup F_{L_2} \cup \bigcup_{(u,v,C) \in (\text{triple}(L_1) \cup \text{triple}(L_2))} L_R(u, v, C)$ . By Lemma 3.1 it follows that  $L_1 \cup L_2 \in 2LT_R(k)$ . Similarly, the intersection  $L_1 \cap L_2$  is represented by  $(F_{L_1} \cap F_{L_2}) \cup \bigcup_{(u,v,C) \in (\text{triple}(L_1) \cap \text{triple}(L_2))} L_R(u, v, C)$ , which shows that  $L_1 \cap L_2 \in 2LT_R(k)$ . Finally, for the  $R$ -complement  $L_R^c$  we have

$$L_R^c = \left( \Sigma^{\leq k-1} \cup \Sigma_R^{\geq k} \right) \setminus L = (\Sigma^{\leq k-1} \setminus F_L) \cup \left( \bigcup_{(u,v,C) \notin \text{triple}(L)} L_R(u, v, C) \right).$$

Since the number of triples  $(u, v, C)$  that do not belong to  $\text{triple}(L)$  is finite, Lemma 3.1 shows that  $L_R^c \in 2LT_R(k)$ .  $\square$

**Proposition 3.4** For all  $k \geq 1$ , the families  $2LT(k)$  and  $2LT$  are closed under intersection and  $R$ -complementation.

*Proof.* Let  $R_1, R_2 \subseteq \Sigma^k \times \Sigma^k$  be two symmetric binary relations. Then  $R_\cap = R_1 \cap R_2$  is also a symmetric binary relation on  $\Sigma^k$ . Moreover,  $\Sigma_{R_\cap}^{\geq k} = \Sigma_{R_1}^{\geq k} \cap \Sigma_{R_2}^{\geq k}$ . Therefore, if  $L_1 \in 2LT_{R_1}(k)$  and  $L_2 \in 2LT_{R_2}(k)$ , then  $L_1 \cap L_2 \in 2LT_{R_\cap}(k)$ , which gives the closure of  $2LT(k)$  under intersection.

By Proposition 3.3, the family  $2LT_R(k)$  is closed under  $R$ -complementation for any symmetric binary relation  $R \subseteq \Sigma^k \times \Sigma^k$ , and so the family  $2LT(k)$  is closed under  $R$ -complementation.

Finally, let  $k_1, k_2 \geq 1$  and let  $L_1 \in 2LT(k_1)$  and  $L_2 \in 2LT(k_2)$ . For  $k = \max\{k_1, k_2\}$  we obtain  $L_1 \in 2LT(k)$  and  $L_2 \in 2LT(k)$  by Lemma 2.4. In this way, the closures of the family  $2LT$  follow from the closures of  $2LT(k)$ .  $\square$

To establish closure under intersection, we used the fact that  $\Sigma_{R_1 \cap R_2}^{\geq k} = \Sigma_{R_1}^{\geq k} \cap \Sigma_{R_2}^{\geq k}$  for any  $k \geq 1$  and any two symmetric binary relations  $R_1, R_2$  on  $\Sigma^k$ . A similar argument does not hold for union, as  $\Sigma_{R_1}^{\geq k} \cup \Sigma_{R_2}^{\geq k}$  is in general a proper subset of  $\Sigma_{R_1 \cup R_2}^{\geq k}$ .

**Example 3.5** Let  $\Sigma = \{a, b\}$ ,  $k = 1$ ,  $R_1 = \{(a, a)\}$  and  $R_2 = \{(b, b)\}$ . Then  $\Sigma_{R_1}^{\geq 1} = a^+$  and  $\Sigma_{R_2}^{\geq 1} = b^+$ , but  $\Sigma_{R_1 \cup R_2}^{\geq 1} = \{w \in \Sigma^+ \mid \forall i = 1, 2, \dots, |w|: w[i] = w[|w| + 1 - i]\}$ , which also contains the word  $w = abba \notin \Sigma_{R_1}^{\geq 1} \cup \Sigma_{R_2}^{\geq 1}$ .  $\blacksquare$

In fact, the families  $2LT$  and  $2LT(k)$  are not closed under union.

**Proposition 3.6** *For all  $k \geq 1$ , the families  $2LT(k)$  and  $2LT$  are not closed under union.*

*Proof.* Let  $k = 1$  and  $\Sigma = \{a, b, c\}$ . We take  $R_1 = \{(a, a), (b, c), (c, b)\}$  and define the language  $L_1$  as follows:

$$\begin{aligned} L_1 &= \{w \in \Sigma_{R_1}^{\geq 1} \mid P_1(w), S_1(w) \in \{a, b, c\}, I_1(w) \subseteq \{a, b, c\}\} \\ &= \{w \mid w \in \{a, b, c\}^*, \forall i = 1, 2, \dots, |w|: w[i] = a \Rightarrow w[|w| + 1 - i] = a \text{ and} \\ &\quad w[i] = b \Rightarrow w[|w| + 1 - i] = c \text{ and } w[i] = c \Rightarrow w[|w| + 1 - i] = b\}. \end{aligned}$$

Similarly, we take  $R_2 = \{(a, a), (b, b), (c, c)\}$  and define the language  $L_2$  as follows:

$$\begin{aligned} L_2 &= \{w \in \Sigma_{R_2}^{\geq 1} \mid P_1(w), S_1(w) \in \{a, b, c\}, I_1(w) \subseteq \{a, b, c\}\} \\ &= \{w \mid w \in \{a, b, c\}^*, \forall i = 1, 2, \dots, |w|: w[i] = w[|w| + 1 - i]\}. \end{aligned}$$

Clearly,  $L_1 \in 2LT_{R_1}(1)$  and  $L_2 \in 2LT_{R_2}(1)$  and, hence,  $L_1, L_2 \in 2LT(1) \subseteq 2LT$ .

Now assume that the union  $L_1 \cup L_2$  belongs to the family  $2LT$ . Then there are an integer  $k \geq 1$  and a symmetric binary relation  $R$  on  $\{a, b, c\}^k$  such that  $L_1 \cup L_2$  belongs to  $2LT_R(k)$ .

We consider the word  $v_1 = a^k b^k a^k c^k a^k$ . Since  $v_1 \in L_1$ , the relation  $R$  necessarily contains the pairs  $(a^{k-i} b^i, c^i a^{k-i})$ ,  $(c^i a^{k-i}, a^{k-i} b^i)$ ,  $(b^{k-i} a^i, a^i c^{k-i})$ , and  $(a^i c^{k-i}, b^{k-i} a^i)$  for all  $0 \leq i \leq k$ .

Since  $v_2 = a^k c^k a^k c^k a^k$  belongs to  $L_2$ , the relation  $R$  also contains the pairs  $(a^{k-i} c^i, c^i a^{k-i})$  and  $(c^i a^{k-i}, a^{k-i} c^i)$  for all  $0 \leq i \leq k$ .

This implies that also the word  $w = a^k b^k a^k c^k a^k c^k a^k c^k a^k$  belongs to  $\{a, b, c\}_R^{\geq k}$ . Moreover, since  $P_k(w) = P_k(v_1)$ ,  $S_k(w) = S_k(v_1)$ , and  $I_k(w) = I_k(v_1)$ , the word  $w$  belongs to the  $k$ - $R$ -testable language  $L_1 \cup L_2$ . However, as  $w \notin L_1$  and  $w \notin L_2$ , this is a contradiction. Hence, we conclude that neither  $2LT(k)$  nor  $2LT$  are closed under union.  $\square$

The closure under reversal is easily seen by reversing all factors and the components of  $R$ . Here a pair  $(u, v) \in R$  is called *non-redundant* for  $L$ , if there exist a word  $w \in L$  and an index  $i$  such that  $w[i, k] = u$  and  $w[|w| + 2 - k - i, k] = v$ .

**Proposition 3.7** *For all  $k \geq 1$ , the families  $2LT(k)$  and  $2LT$  are closed under reversal. For a language  $L \in 2LT_R(k)$ , its reversal  $L^R$  belongs to  $2LT_R(k)$  if and only if, for all pairs  $(u, v) \in R$  that are non-redundant for  $L$ , the pair  $(u^R, v^R)$  belongs to  $R$ .*

*Proof.* Consider a  $k$ - $R$ -testable language  $L$ . For any factor  $u$  of length  $k$  from some word  $w \in L$ , the factor  $u^R$  appears in  $w^R$  and vice versa. Moreover, for  $1 \leq i \leq |w| - k + 1$ , whenever the factors  $w[i, k]$  and  $w[|w| + 2 - k - i, k]$  of  $w \in L$  are in relation  $R$ , the factors  $(w[|w| + 2 - k - i, k])^R$  and  $(w[i, k])^R$  of  $w^R$  are in relation  $R_r = \{(u^R, v^R) \mid (u, v) \in R\}$ . So,  $L^R$  is  $k$ - $R_r$ -testable. This shows the assertion for  $2LT(k)$  and  $2LT$ .

Finally, let  $L \in 2LT_R(k)$ . Then, in order to have  $L^R \in 2LT_R(k)$ , it is necessary and sufficient that  $(u^R, v^R) \in R$  for all pairs  $(u, v) \in R$  that are non-redundant for  $L$ .  $\square$

Next, we turn to the operations of concatenation and Kleene star.

**Proposition 3.8** *For all  $k \geq 1$ , the families  $2LT(k)$  and  $2LT$  are neither closed under concatenation nor under Kleene star. There exist an alphabet  $\Sigma$  and a symmetric relation  $R \subseteq \Sigma^k \times \Sigma^k$  such that the family  $2LT_R(k)$  is neither closed under concatenation nor under Kleene star.*

*Proof.* Let  $k = 1$  and  $\Sigma = \{a, b, c\}$ . We choose  $R = \{(a, a), (b, c), (c, b)\}$  and consider the language

$$\begin{aligned} L &= \{ w \in \Sigma_R^{\geq 1} \mid P_1(w), S_1(w) \in \{a, b, c\}, I_1(w) \subseteq \{a, b, c\} \} \\ &= \{ w \mid w \in \{a, b, c\}^*, \forall i = 1, 2, \dots, |w|: w[i] = a \Rightarrow w[|w| + 1 - i] = a \text{ and} \\ &\quad w[i] = b \Rightarrow w[|w| + 1 - i] = c \text{ and } w[i] = c \Rightarrow w[|w| + 1 - i] = b \}. \end{aligned}$$

The language  $L$  is 1- $R$ -testable and belongs to all families in question. Let  $\ell \geq 1$  be a positive integer and let  $R' \subseteq \{a, b, c\}^\ell \times \{a, b, c\}^\ell$  be a symmetric relation, and assume that the concatenation  $L \cdot L$  or the iteration  $L^*$  belongs to the family  $2LT_{R'}(\ell) \subseteq 2LT(\ell) \subseteq 2LT$ .

Since  $a^+ \in L$  and  $b^\ell c^\ell \in L$ , the word  $w = b^\ell c^\ell a^{2\ell}$  belongs to the concatenation  $L \cdot L$  as well as to the iteration  $L^*$ . Therefore, the relation  $R'$  necessarily contains the pairs  $(b^{\ell-i} c^i, a^\ell)$ ,  $(a^\ell, b^{\ell-i} c^i)$ ,  $(c^{\ell-i} a^i, c^i a^{\ell-i})$  for all  $0 \leq i \leq \ell$ .

Now consider the word  $w' = b^\ell c^{\ell+1} a^{2\ell+1} \in \Sigma_{R'}^{\geq \ell}$ . Since  $P_\ell(w') = P_\ell(w)$ ,  $S_\ell(w') = S_\ell(w)$ , and  $I_\ell(w') = I_\ell(w)$ , we conclude that  $w' \in L \cdot L$  or  $w' \in L^*$ . However, since in any word from  $L$  the number of occurrences of the letter  $b$  is equal to the number of occurrences of  $c$ , the same is true for  $L \cdot L$  and  $L^*$ . Since  $w'$  includes  $\ell$  occurrences of the letter  $b$  but  $(\ell + 1)$  occurrences of the letter  $c$ , the word  $w'$  can neither belong to  $L \cdot L$  nor to  $L^*$ , a contradiction.  $\square$

Turning to the operation of applying homomorphisms, we see that even the restriction to length-preserving homomorphisms does not yield a positive result.

**Proposition 3.9** *For all  $k \geq 1$ , the families  $2LT(k)$  and  $2LT$  are not closed under length-preserving homomorphisms. There exist a binary alphabet  $\Sigma$  and a symmetric relation  $R \subseteq \Sigma^k \times \Sigma^k$  such that the family  $2LT_R(k)$  is not closed under length-preserving homomorphisms.*

*Proof.* The non-closure result is shown by using the two-sided strictly 1-testable language  $L'$  over the alphabet  $\{a, b\}$  from Example 2.1, whose non-empty words have the property that the  $i$ -th letter from the left differs from the  $i$ -th letter from the right. Let  $h : \{a, b\}^* \rightarrow \{a\}^*$  be the length-preserving homomorphism that is defined by  $h(a) = h(b) = a$ . Since all words from  $L'$  have even length,  $h(L') = \{a^{2n} \mid n \geq 1\}$ , which is not even two-sided locally testable. So,  $h(L')$  does not belong to  $2LT$ .  $\square$

Finally, we consider inverse homomorphisms. In this case, the edge between closure and non-closure is sharp. While the families  $2LT(k)$  as well as  $2LT$  are closed under length-preserving inverse homomorphisms, this closure property is lost when the restriction of the homomorphisms is relaxed to being only  $\lambda$ -free, just as it happens for the two-sided strict testability (cf. [2]).

**Proposition 3.10** *For all  $k \geq 1$ , the families  $2LT(k)$  and  $2LT$  are closed under length-preserving inverse homomorphisms. For a language  $L \in 2LT_R(k)$ , its preimage  $h^{-1}(L)$*

with respect to a length-preserving homomorphism  $h$  belongs to  $2LT_R(k)$  if and only if  $\{(x, y) \mid (h(x), h(y)) \in R \text{ is non-redundant for } L\}$  is contained in  $R$ .

*Proof.* Consider a two-sided locally testable language  $L$  over an alphabet  $\Sigma$  and let  $h : \Delta^* \rightarrow \Sigma^*$  be a length-preserving homomorphism mapping from some alphabet  $\Delta$  to  $\Sigma$ .

In order to represent  $h^{-1}(L)$ , it suffices to consider prefixes, suffixes, and factors whose homomorphic images are factors that appear in the representation of  $L$ . More precisely,

$$\forall w \in \Sigma^{\geq k} \forall w' \in \Delta^{\geq k}: \quad \text{if } P_k(w) = h(P_k(w')) \wedge S_k(w) = h(S_k(w')) \wedge I_k(w) = h(I_k(w')), \\ \text{then } (w \in L \iff w' \in h^{-1}(L)).$$

In other words, if language  $L$  is  $k$ - $R$ -testable, then its preimage  $h^{-1}(L)$  is  $k$ - $R_h$ -testable, where  $R_h = \{(u, v) \mid (h(u), h(v)) \in R\}$ .

Finally, if  $h$  is a length-preserving homomorphism from  $\Sigma^*$  to  $\Sigma^*$ , and if  $L \subseteq \Sigma^*$  belongs to  $2LT_R(k)$ , then the language  $h^{-1}(L)$  also belongs to  $2LT_R(k)$  if and only if, for all pairs  $(u, v) \in R$  that are non-redundant for  $L$ , each pair  $(x, y) \in (h^{-1}(u), h^{-1}(v))$  belongs to  $R$ .  $\square$

As mentioned before, the restriction to non-erasing homomorphisms is not sufficient to obtain a positive closure result.

**Proposition 3.11** *For all  $k \geq 1$ , the families  $2LT(k)$  and  $2LT$  are not closed under  $\lambda$ -free inverse homomorphisms. There exist a binary alphabet  $\Sigma$  and a symmetric relation  $R \subseteq \Sigma^k \times \Sigma^k$  such that the family  $2LT_R(k)$  is not closed under  $\lambda$ -free inverse homomorphisms.*

*Proof.* Once more, we utilize the language  $L'$  over the alphabet  $\{a, b\}$  from Example 2.1, whose non-empty words have the property that the  $i$ -th letter from the left differs from the  $i$ -th letter from the right. It is a two-sided strictly 1-testable language and all of its words have even lengths.

Let  $h : \{a, b\}^* \rightarrow \{a, b\}^*$  be the  $\lambda$ -free homomorphism defined by  $h(a) = aa$  and  $h(b) = b$ , and assume that the language  $h^{-1}(L')$  belongs to some family  $2LT_R(\ell)$ . Since the word  $a^{2\ell+2}b^{2\ell+2}$  belongs to  $L'$ , the word  $w = a^{\ell+1}b^{2\ell+2}$  belongs to  $h^{-1}(L')$ . Therefore, the relation  $R$  necessarily contains the pairs  $(a^{\ell-i}b^i, b^\ell)$  and  $(b^\ell, a^{\ell-i}b^i)$  for all  $0 \leq i \leq \ell$ .

Now consider the word  $w' = a^{\ell+1}bb^{2\ell+2}$  that is from  $\Sigma_R^{\geq \ell}$ . Since  $P_\ell(w') = P_\ell(w)$ ,  $S_\ell(w') = S_\ell(w)$ , and  $I_\ell(w') = I_\ell(w)$ , we conclude  $w' \in h^{-1}(L)$ . However, the word  $h(w') = a^{2\ell+2}bb^{2\ell+2}$  of odd length does not belong to  $L'$ . This contradiction shows that  $h^{-1}(L')$  does not belong to  $2LT$ .  $\square$

## 4. Expressive Capacity

Here we turn to the study of the expressive capacity of two-sided locally testable languages with respect to other important language families. It is not hard to see that even the strongest family under consideration does not contain the regular languages, as, for example, the regular

language  $(aa)^*$  is not two-sided locally testable [2]. On the other hand, Example 2.1 reveals that there is even a two-sided strictly 1-testable language which is not regular. Before we continue with such relationships we show an upper bound for the expressive capacity. To this end, we improve the result from [2] that all two-sided strictly locally testable languages are even linear context-free by showing that the family  $2LT$  is a proper subfamily of the even linear languages. Recall that a language  $L \subseteq \Sigma^*$  is called *even linear* if it is generated by a grammar  $G = (V, \Sigma, S, P)$  such that all productions in  $P$  are of the form  $(A \rightarrow uBv)$ , where  $A, B \in V$  and  $u, v \in \Sigma^*$  satisfying  $|u| = |v|$ , or of the form  $(A \rightarrow w)$ , where  $A \in V$  and  $w \in \Sigma^*$  [1]. The family of even linear languages is denoted by  $ELIN$ .

**Theorem 4.1** *The family  $2LT$  is properly included in  $ELIN$ .*

*Proof.* Since the family of even linear languages includes the regular languages, but not all regular languages are two-sided locally testable, we can immediately conclude that not all even linear languages are two-sided locally testable.

So, it remains to be shown that each two-sided locally testable language is even linear. For each  $L \in 2LT$ , there are an integer  $k \geq 1$  and a symmetric binary relation  $R$  on  $\Sigma^k$ , where  $\Sigma$  is the alphabet of  $L$ , such that  $L$  belongs to  $2LT_R(k)$ . From above we know that each two-sided locally testable language  $L$  can be represented as  $L = F \cup \bigcup_{(u,v,C) \in \text{triple}(L)} L_R(u, v, C)$ , where  $F \subseteq \Sigma^{\leq k-1}$  and  $L_R(u, v, C) = \{w \in \Sigma^{\geq k} \mid P_k(w) = u, S_k(w) = v, \text{ and } I_k(w) = C\}$ . Since the set  $\text{triple}(L)$  is finite and the family  $ELIN$  is closed under union, it is sufficient to construct even linear grammars for the sets  $L_R(u, v, C)$ . This, however, can be done similarly to the construction of a linear grammar for a two-sided strictly  $k$ -testable language (see [2]).  $\square$

We continue by collecting some more or less immediate relationships of the family of two-sided locally testable languages. The next example from [2] is a slight modification of Example 2.1, where  $L_{pal}$  is defined to be the language of *palindromes*  $L_{pal} = \{w \in \Sigma^* \mid w = w^R\}$ .

**Example 4.2** *Let  $k = 1$ , let  $\Sigma = \{a, b\}$ , and let  $R \subset \Sigma \times \Sigma$  be defined as  $R = \{(a, a), (b, b)\}$ . Then  $\Sigma_R^{\geq 1}$  consists of all those non-empty words for which the  $i$ -th letter from the left coincides with the  $i$ -th letter from the right, that is,  $\Sigma_R^{\geq 1} = L_{pal} \cap \Sigma^+$ . So,  $L_{pal}$  is (strictly) 1-testable.  $\blacksquare$*

Recall that the language  $L_{pal}$  is not a Church-Rosser language [3]. On the other hand, the regular language  $(aa)^*$  is not two-sided locally testable. If it was, then there would be  $k \geq 1$  and a relation  $R$  such that  $(aa)^*$  is  $k$ - $R$ -testable. In this case  $R = \{(a^k, a^k)\}$ , and hence, also unary words of odd lengths would belong to the language considered, a contradiction. In fact, for a unary language  $L \subseteq a^*$ , it is easily seen that it is two-sided  $k$ -testable if and only if it is strictly  $k$ -testable.

**Corollary 4.3** *For all  $k \geq 1$ , the families  $2LT$  and  $2LT(k)$  are incomparable to the families of regular, deterministic linear, deterministic context-free, and Church-Rosser languages. There exist a binary alphabet  $\Sigma$  and a symmetric relation  $R \subseteq \Sigma^k \times \Sigma^k$  such that the family  $2LT_R(k)$  is incomparable to the families of regular, deterministic linear, deterministic context-free, and Church-Rosser languages.*

Before we discuss the inclusions depicted in Figure 1 below we present the following relationships to two-sided strictly locally testable languages, which corresponds to the situation in the one-sided setting [5]. Here  $2SLT_R(k)$  is used to denote the family of two-sided strictly  $k$ -testable languages that are based on the fixed symmetric relation  $R$ .

**Theorem 4.4** *For each  $k \geq 1$  and each symmetric relation  $R \subseteq \Sigma^k \times \Sigma^k$ , the family  $2LT_R(k)$  is the closure of the family  $2SLT_R(k)$  under union, intersection, and  $R$ -complementation.*

*Proof.* By Proposition 3.3, the family  $2LT_R(k)$  is closed under the operations considered. Since  $2SLT_R(k) \subseteq 2LT_R(k)$ , it is sufficient to show that every language from  $2LT_R(k)$  can be represented as a combination of languages from  $2SLT_R(k)$  using the operations of union, intersection, and  $R$ -complementation.

Recall from above that each language  $L \in 2LT_R(k)$  has a representation of the form  $L = F_L \cup \bigcup_{(u,v,C) \in \text{triple}(L)} L_R(u, v, C)$ , where  $F_L$  is a set of strings of lengths less than  $k$  and  $L_R(u, v, C) = \{w \in \Sigma_R^{\geq k} \mid P_k(w) = u, S_k(w) = v, \text{ and } I_k(w) = C\}$ . Since  $\text{triple}(L)$  is a finite set, it remains to be shown that any language  $L_R(u, v, C)$  with  $(u, v, C) \in \text{triple}(L)$  can be represented as a combination of languages from  $2SLT_R(k)$  using the above operations.

Starting with the two-sided strictly  $k$ -testable language  $L(\{u\}, \{v\}, C, R)$  we obtain the inclusion  $L_R(u, v, C) \subseteq L(\{u\}, \{v\}, C, R)$ . The problem to cope with is that there may be words  $w$  in the latter language such that  $I_k(w) \subset C$ . These words do not belong to  $L_R(u, v, C)$ . However, the set of these words can be filtered out by building the intersection of the  $R$ -complements of all languages  $L(\{u\}, \{v\}, C', R)$ , where the intersection is taken over all proper subsets  $C'$  of  $C$ , that is,  $\bigcap_{C' \subset C} L_R^c(\{u\}, \{v\}, C', R)$ .

So, we have the representation  $L_R(u, v, C) = L(\{u\}, \{v\}, C, R) \cap \bigcap_{C' \subset C} L_R^c(\{u\}, \{v\}, C', R)$ . Thus, each language  $L \in 2LT_R(k)$  is a finite combination of two-sided strictly  $k$ -testable languages with respect to the relation  $R$ .  $\square$

Since the families  $2LT(k)$  and  $2LT$  are not closed under union, a similar characterization in terms of two-sided strictly locally testable languages does not exist. However, for every language  $L$  from these families, there are an integer  $k \geq 1$  and a symmetric relation  $R \subseteq \Sigma^k \times \Sigma^k$  such that  $L \in 2LT_R(k)$ . This implies that  $L$  has a representation as a combination of two-sided strictly  $k$ -testable languages and, thus, belongs to the closure of  $2SLT_R(k)$  with respect to the operations of union, intersection, and  $R$ -complementation. So, we have the following corollary, where by  $R$ -Boolean closure we mean the closure under the operations of union, intersection, and  $R$ -complementation.

**Corollary 4.5** *Let  $k \geq 1$ .*

1. *The family  $2LT$  is properly included in the  $R$ -Boolean closure of  $2SLT$ .*
2. *The family  $2LT(k)$  is properly included in the  $R$ -Boolean closure of  $2SLT(k)$ .*
3. *The  $R$ -Boolean closure of  $2LT$  coincides with the  $R$ -Boolean closure of  $2SLT$ .*
4. *The  $R$ -Boolean closure of  $2LT(k)$  coincides with the  $R$ -Boolean closure of  $2SLT(k)$ .*

In order to give evidence for the inclusions of Figure 1, we note that all inclusions depicted follow for structural reasons. Moreover, recall that all one-sided locally testable families depicted are subsets of the regular languages, while even the two-sided strictly 1-testable languages include a non-regular language. This shows the properness of the inclusions between the variants of one-sided and two-sided testable families. The properness of the inclusions of  $2LT_R(k)$  in  $2LT(k)$  and of  $LT_R(k)$  in  $LT(k)$  is trivial, since the latter contain languages based on different relations. So, the hierarchical inclusions remain to be shown. Here we can apply witness languages that also separate the levels of the hierarchy of strictly locally testable languages.

For all  $k \geq 1$ , let  $L_k$  be the finite language  $L_k = \{a^k, a^{k+1}\}$  and let  $R_k$  be the symmetric binary relation  $R_k = \{(a^k, a^k)\}$ .

**Lemma 4.6** *For all  $k \geq 1$ ,  $L_{k+1} \in LT_{R_{k+1}}(k+1) \setminus 2LT(k)$ .*

*Proof.* The language  $L_{k+1}$  belongs to  $LT_{R_{k+1}}(k+1)$ , since

$$P_{k+1}(a^{k+1}) = S_{k+1}(a^{k+1}) = a^{k+1} = P_{k+1}(a^{k+2}) = S_{k+1}(a^{k+2})$$

and  $I_{k+1}(a^{k+1}) = I_{k+1}(a^{k+2}) = \emptyset$ . Any other word  $w$  from  $\{a\}^{\geq k+2}$  has a non-empty set  $I_{k+1}(w)$ .

On the other hand, assume that  $L_{k+1}$  is a two-sided  $k$ -testable language. Then the underlying relation  $R$  necessarily contains the pair  $(a^k, a^k)$ . However, since

$$P_k(a^{k+2}) = S_k(a^{k+2}) = a^k = P_k(a^{k+3}) = S_k(a^{k+3})$$

and  $I_k(a^{k+2}) = I_k(a^{k+3}) = \{a^k\}$ , the word  $a^{k+3}$  belongs to the language as well, a contradiction.  $\square$

Thus, we can draw the following conclusions.

**Corollary 4.7** *1. The (two-sided)  $k$ -testable languages form an infinite ascending hierarchy with respect to the parameter  $k$ .*

*2. For all  $k \geq 1$ , there exist symmetric relations  $R_k = \{(a^k, a^k)\}$  such that the families  $2LT_{R_k}(k)$  ( $LT_{R_k}(k)$ ) form an infinite ascending hierarchy with respect to the parameter  $k$ .*

*3. For all  $k \geq 1$ ,  $LT(k) \subsetneq 2LT(k)$ , but  $2LT(k) = SLT(k)$  for unary languages.*

*4. All families depicted in Figure 1 that are not connected by a path are incomparable.*

Above we have argued that the locally testable languages are properly contained in the two-sided locally testable languages, as the former only contain regular languages, while the latter contain some non-regular languages. In the case of strict testability, it has been observed in [2] that there are even regular languages that are two-sided strictly testable, but not (one-sided) strictly testable. Here we show a corresponding result for  $k$ -testability.

**Theorem 4.8** *For all  $k \geq 2$ ,  $LT(k) \subsetneq 2LT(k) \cap \text{REG}$ .*

*Proof.* Let us first consider the case of  $k = 2$ . We take  $\Sigma = \{a, b, c, d\}$  and

$$R_2 = \{(ab, aa), (aa, ab), (bd, da), (da, bd), (db, bd), (bd, db), \\ (bc, cb), (cb, bc), (bc, db), (db, bc), (cb, bd), (bd, cb)\}.$$



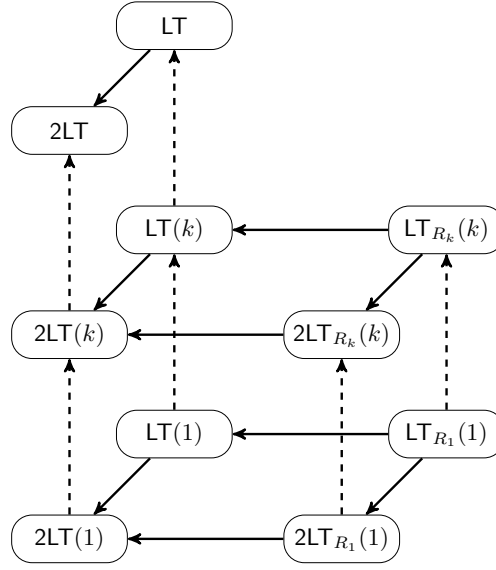


Figure 1: Inclusion structure of two-sided and one-sided locally testable language families. The arrows indicate strict inclusions. The dashed lines are used to express the infinite hierarchies dependent on the window size  $k$ . The families  $2LT_{R_k}(k)$  and  $LT_{R_k}(k)$  depend on fixed binary symmetric relations  $R_k \subseteq \Sigma^k \times \Sigma^k$ . There are such relations that witness the inclusions depicted. All families not connected by a path are incomparable.

Now we choose  $u = ab$ ,  $v = aa$ , and  $C = \{bc, cb, bd, db, da\}$ , and define the language  $L_2$  as

$$L_2 = L_{R_2}(u, v, C) = \{w \in \Sigma_{R_2}^{\geq 2} \mid P_2(w) = u, S_2(w) = v, \text{ and } I_2(w) = C\}.$$

Then

$$L_2 = \{abdbx_1bx_2b \cdots bx_nbdaa \mid n \geq 1, x_1, x_2, \dots, x_n \in \{c, d\}, \exists i : x_i = c\},$$

as the only allowed inner factor that ends in  $a$  is the factor  $da$ , the only other factor of length two that is in relation with  $da$  is  $bd$ , and as the inner factors  $bc$  and  $cb$  must occur. By definition  $L_2$  is two-sided  $k$ -testable, and it is obviously a regular language. Now  $w_1 = abdbcbdaa \in L_2$ , while  $w_2 = abcbdbdaa \notin L_2$ . However,  $P_2(w_2) = ab = P_2(w_1)$ ,  $S_2(w_2) = aa = S_2(w_1)$ , and  $I_2(w_2) = C = I_2(w_1)$ , which implies that  $L_2$  is not 2-testable, that is,  $L_2 \in (2LT(2) \cap \text{REG}) \setminus LT(2)$ .

On the other hand, it is easily seen that  $L_2$  is 3-testable, as it consists of all words  $w$  satisfying  $P_3(w) = abd$ ,  $S_3(w) = daa$ , and  $I_3(w) \in \{I, I \cup \{dbd\}, I \cup \{cbc\}, I \cup \{cbc, dbd\}\}$ , where  $I$  is the minimal set of required infixes  $I = \{bdb, dbc, cbd, bda\}$ .

For  $k \geq 3$ , we choose the language

$$L_k = \{a^{k-1}b^{k-1}db^{k-1}x_1b^{k-1} \cdots b^{k-1}x_nb^{k-1}da^k \mid n \geq 1, x_1, x_2, \dots, x_n \in \{c, d\}, \exists i : x_i = c\}.$$

Then  $w_1 = a^{k-1}b^{k-1}db^{k-1}cb^{k-1}da^k \in L_k$ , but  $w_2 = a^{k-1}b^{k-1}cb^{k-1}db^{k-1}da^k \notin L_k$ , although  $P_k(w_1) = a^{k-1}b = P_k(w_2)$ ,  $S_k(w_1) = a^k = S_k(w_2)$ , and

$$I_k(w_1) = \{a^i b^{k-1-i} \mid 1 \leq i \leq k-2\} \cup \{b^i db^{k-1-i}, b^i cb^{k-1-i}, b^i da^{k-1-i} \mid 0 \leq i \leq k-1\} = I_k(w_2).$$

Thus,  $L_k$  is not  $k$ -testable. By taking  $u = a^{k-1}b$ ,  $v = a^k$ ,  $C = I_k(w_1)$  and by choosing a corresponding symmetric binary relation  $R_k$  on  $\Sigma^k$ , it can be shown that  $L_k = L_{R_k}(u, v, C)$ , that is,  $L_k \in (2LT(k) \cap \text{REG}) \setminus LT(k)$ . However, it can be shown that the language  $L_k$  is locally  $(2k - 1)$ -testable.  $\square$

A corresponding result does not hold for  $k = 1$ .

**Theorem 4.9**  $2LT(1) \cap \text{REG} = LT(1)$ .

Currently, we do not yet know whether  $2LT \cap \text{REG}$  contains any languages that are not locally testable.

Let  $L \subseteq \Sigma^*$  be a two-sided locally testable language that is given through a symmetric binary relation  $R \subseteq \Sigma^k \times \Sigma^k$ , a finite set  $F_L \subseteq \Sigma^{\leq k-1}$ , and the set

$$\text{triple}(L) = \{ (u, v, C) \mid u, v \in \Sigma^k \text{ and } C \subseteq \Sigma^k \text{ such that } L_R(u, v, C) \cap L \neq \emptyset \},$$

that is,  $L = F_L \cup \bigcup_{(u,v,C) \in \text{triple}(L)} L_R(u, v, C)$ . Then the membership problem for  $L$ , that is, the question of whether a given word  $w \in \Sigma^*$  belongs to  $L$ , is obviously decidable in linear time. As emptiness and finiteness are decidable for linear languages, Theorem 4.1 implies that these problems are also decidable for two-sided locally testable languages.

The language  $L$  is universal, that is,  $L = \Sigma^*$ , if and only if  $\bigcup_{(u,v,C) \in \text{triple}(L)} L_R(u, v, C) = \Sigma^{\geq k}$  and  $L_F = \Sigma^{\leq k-1}$ . The latter is easily tested, and the former implies in particular that all words of length at least  $k$  are  $R$ -symmetric, that is,  $R = \Sigma^k \times \Sigma^k$ , which is easily tested, too. If  $R = \Sigma^k \times \Sigma^k$ , then  $L$  is actually a locally testable language, and therewith regular, which means that the question of whether  $L = \Sigma^*$  is decidable.

**Theorem 4.10** *Universality is decidable for two-sided locally testable languages.*

The universality problem can be seen as a special case of the following *Regular Inclusion Problem*:

- INSTANCE:** A regular language  $S \subseteq \Sigma^*$  and a two-sided locally testable language  $L \subseteq \Sigma^*$ .  
**QUESTION:** Is  $S$  contained in  $L$ ?

In fact, this problem is decidable.

**Theorem 4.11** *The Regular Inclusion Problem is decidable for two-sided locally testable languages.*

Given a regular language  $S$  and a two-sided locally testable language  $L$ , we can decide whether  $L$  is contained in  $S$ . In fact, we can construct an even linear grammar for the language  $L$  (as the proof of Theorem 4.1 is constructive), and from a DFA for  $S$ , we easily obtain a DFA for the complement  $S^c = \Sigma^* \setminus S$  of  $S$ . Hence, we can construct a linear grammar  $G$  for the language  $L \cap S^c$ . Now  $L \subseteq S$  if and only if  $L(G) = L \cap S^c$  is empty, which is decidable. From this observation and from Theorem 4.11 we get the following decidability result.

**Corollary 4.12** *The following Regular Equality Problem is decidable:*

**INSTANCE:** *A regular language  $S \subseteq \Sigma^*$  and a two-sided locally testable language  $L$ .*

**QUESTION:** *Is  $S = L$ ?*

It remains currently open whether we can decide regularity for two-sided locally testable languages. Finally we turn to the inclusion and equivalence problems for two-sided locally testable languages.

**Theorem 4.13** *Let  $k \geq 1$ , let  $\Sigma$  be a finite alphabet, and let  $R$  be a symmetric binary relation on  $\Sigma^k$ . Then the inclusion problem is decidable for  $k$ - $R$ -testable languages.*

*Proof.* Let  $L_1 = F_{L_1} \cup \bigcup_{(u,v,C) \in \text{triple}(L_1)} L_R(u, v, C)$  and  $L_2 = F_{L_2} \cup \bigcup_{(x,y,D) \in \text{triple}(L_2)} L_R(x, y, D)$  be two  $k$ - $R$ -testable languages over  $\Sigma$ . Then  $L_1 \subseteq L_2$  if and only if the following conditions are satisfied:

1.  $F_{L_1} \subseteq F_{L_2}$ , and
2.  $\text{triple}(L_1) \subseteq \text{triple}(L_2)$ .

Obviously, these conditions are checked easily. □

Hence, it follows that for  $k$ - $R$ -testable languages equivalence is decidable. Actually, we can extend the above result slightly as follows.

**Theorem 4.14** *Let  $k \geq 1$ , let  $\Sigma$  be a finite alphabet, and let  $R_1$  and  $R_2$  be two symmetric binary relations on  $\Sigma^k$  such that  $R_1 \subseteq R_2$ . Then it is decidable whether a  $k$ - $R_1$ -testable language  $L_1$  is contained in a  $k$ - $R_2$ -testable language  $L_2$ .*

If  $L_1$  is a  $k$ - $R_1$ -testable and  $L_2$  is a  $k$ - $R_2$ -testable language on the same alphabet  $\Sigma$ , but  $R_1 \subseteq R_2$  does not hold, then in order to check  $L_1 \subseteq L_2$ , also  $L_{R_1}(u, v, C) \subseteq \Sigma_{R_2}^{\geq k}$  must be checked for each  $(u, v, C) \in \text{triple}(L_1)$ . We currently do not see how this can be done algorithmically. Thus, it remains open whether inclusion (or equivalence) is decidable for the classes  $2\text{LT}(k)$  and  $2\text{LT}$ .

## 5. Conclusions, Open and Untouched Questions

We have extended the two-sided strictly locally testable languages of [2] to the two-sided locally testable languages. We have shown that the latter are obtained as the  $R$ -Boolean closure of the former, and we have established some closure and non-closure properties. Further, extending the results of [2], it can be shown that two-sided  $k$ -testable languages are learnable in the limit from positive data. Some problems that remain open for future work include the following.

**Inside the Boolean closure:** How about the union closure of  $2\text{SLT}(k)$ , etc? For example, the language over alphabet  $\{a, b\}$  whose words have to have both factors  $a^k$  and  $b^k$  belongs to  $\text{LT}(k)$ , but does not have a representation as union of languages from  $2\text{SLT}(k)$ . So, the union closure is properly contained in the Boolean closure.

**Separation by regular languages:** For each fixed integer  $k \geq 2$ , we have seen that there exists a regular language  $L_k$  that is two-sided  $k$ -testable, but not  $k$ -testable. However, as  $L_k$  is  $k'$ -testable for some integer  $k' > k$ , it remains open whether we can separate the family of (one-sided) locally testable languages from the family of two-sided locally testable languages by a regular language.

**Decidability:** We have seen that emptiness, finiteness, containment of a given regular set, and equality to a given regular set are decidable for two-sided locally testable languages. Further, inclusion and equivalence are decidable for  $k$ - $R$ -testable languages, but it remains open whether these problems are decidable for two-sided locally testable languages in general.

## References

- [1] V. AMAR, G. R. PUTZOLU, On a family of linear grammars. *Information and Control* 7 (1964), 283–291.
- [2] M. HOLZER, M. KUTRIB, F. OTTO, Two-sided strictly locally testable languages. In: R. FREUND, F. MRÁZ, D. PRŮŠA (eds.), *Non-Classical Models of Automata and Applications (NCMA 2017)*. books@ocg.at 329, Österreichische Computer Gesellschaft, Wien, 2017, 135–150.
- [3] T. JURDZIŃSKI, K. LORYŚ, Lower bound technique for length-reducing automata. *Information and Computation* 205 (2007), 1387–1412.
- [4] R. MC NAUGHTON, P. NARENDRAN, F. OTTO, Church-Rosser Thue systems and formal languages. *Journal of the ACM (JACM)* 35 (1988), 324–344.
- [5] R. MC NAUGHTON, S. PAPERT, *Counter-Free Automata*. Number 65 in Research monographs, MIT Press, 1971.
- [6] B. NAGY, A family of two-head pushdown automata. In: R. FREUND, M. HOLZER, N. MOREIRA, R. REIS (eds.), *Non-Classical Models of Automata and Applications (NCMA 2015)*. books@ocg.at 318, Österreichische Computer Gesellschaft, Wien, 2015, 177–191.
- [7] A. L. ROSENBERG, A machine realization of the linear context-free languages. *Information and Control* 10 (1967), 175–188.

# CHARACTERIZATIONS OF LRR-LANGUAGES BY CORRECTNESS-PRESERVING COMPUTATIONS

František Mráz      Friedrich Otto      Martin Plátek

Charles University, Faculty of Mathematics and Physics  
Department of Computer Science, Malostranské nám. 25  
118 00 Praha 1, Czech Republic

frantisek.mraz@mff.cuni.cz    otto@ktiml.mff.cuni.cz    martin.platek@mff.cuni.cz

## ***Abstract***

*We present a transformation from several types of monotone deterministic two-way restarting list automata (RLAs) that do not have any type of correctness preserving property into several types of monotone deterministic RLAs that satisfy the complete strong correctness preserving property. These types of automata provide new characterizations for the class LRR of left-to-right regular languages, which are suitable for the (lexical) disambiguation of the syntactic analysis of individual LRR-languages and for the localization of syntactical errors.*

## **1. Introduction**

The motivation for this paper and some other related papers is to give a theoretical background for an environment that supports lexicalized syntax of natural languages based on constraints. The first goal of these papers are techniques for lexical disambiguation. Here we show that the class LRR of left-to-right regular languages of [17] can be characterized by several types of constrained two-way restarting list automata (RLAs), and that some of them yield fully (lexically) disambiguated syntactic analysis with almost continuous constituents (reductions in contextual form, see below) for LRR-languages. This result can be used to present a technique for lexical disambiguation and syntactic analysis for all context-free languages (see [14]).

We present a transformation from several types of monotone deterministic RLAs that do not have any type of correctness preserving property into several types of monotone deterministic RLAs that satisfy the complete strong correctness preserving property. These types of automata provide new characterizations for the class LRR, which are suitable for the (lexical) disambiguation of the syntactic analysis of individual LRR-languages and for the localization of syntactical errors. A first such transformation, which was also mentioned in [14], was described in detail in the technical report [15]. As we have found a serious gap in the description of this transformation, we present a different, much simplified, transformation in the current paper.

In [14] it is shown that restarting automata with the complete (weak) correctness preserving

property are sensitive to the size of their windows, which provides us with a tool for measuring the complexity of analysis by reduction (see below). For natural languages as well as for programming languages, this type of complexity is fairly low. On the other hand, even within the three lower classes of the Chomsky hierarchy, this type of complexity is not bounded [14]. Let us note that similar correctness preserving properties hold for (Marcus) contextual grammars (see, e.g., [9]) and for pure grammars (see, e.g., [10]). Contextual and pure grammars are motivated by linguistic considerations similar to our considerations.

In addition to presenting the new transformation, we also enhance the result by requiring the automata obtained to be in the so-called strong cyclic form (see, e.g., [5]). The strong cyclic form is a useful notion which, together with the complete strong correctness preserving property, supports the localization of syntactical errors and error recovery (see, e.g., [16]).

In order to formulate our results in quite a general form, we consider the two-way restarting list automaton that combines the features of a restarting automaton (see, e.g., [12]) with those of a list automaton [7]. A *two-way restarting list automaton* (RLA)  $M$  is a one-tape automaton with a finite-state control and a read/write window of a fixed finite size. This window can move in both directions along the tape (that is, a list of items) containing a word delimited by sentinels. The RLA  $M$  uses an input alphabet and a working (basic) alphabet that includes the input alphabet. It can decide (in general non-deterministically) to rewrite the contents of its window: it may delete some items from the list and/or replace some items. In addition,  $M$  can perform *restart operations*. A *restart* causes  $M$  to move its window to the left end of the tape, so that the first symbol it contains is the left sentinel, and to reenter its initial state.

We recall some constraints that are typical for restarting automata, and we outline ways for new combinations of constraints. These constraints include the use of deletions instead of rewritings, the restriction to contextual deletions, the *complete strong correctness preserving property* (CSCPP), and the requirement that the automata are in strong cyclic form. The CSCPP is an important property of analysis by reduction. It guarantees that, in a sequence of transformations  $w_1 \rightarrow w_2 \rightarrow \dots \rightarrow w_n$ , all words  $w_1, w_2, \dots, w_n$  belong to the language considered if at least one of them belongs to that language. Further, the property of being in *strong cyclic form* is an important constraint, which states that an automaton can only accept or reject once it has reduced the length of the given input to the size of its window [5]. In particular, we show that the power of monotone deterministic two-way restarting list automata (det-mon-RLAs) to accept input languages does not decrease, if we use the corresponding type of automaton which, instead of performing (length-reducing) rewrites, can only delete symbols, which are in the so-called (Marcus) contextual form, and which are additionally in the strong cyclic form. In fact, we will establish this result for RLAs that use the restart operation (det-mon-RLWC) and for the same type of RLAs that do not use the restart operation. This strengthens a result from [8] in several ways, which states that det-mon-RLWW-automata are equivalent to det-mon-RLWD-automata with respect to the input languages they accept (see Section 3 for the corresponding definitions). Let us recall that det-mon-RLWW-automata characterize an important class of languages – the class LRR of *left-to-right regular languages* that was introduced and studied in [17]. This characterization was shown in [13].

Let us remark that for the characterization of the class DCFL of deterministic context-free

languages by (one-way) deterministic monotone RLAs that only delete symbols we need the operation restart [7]. In this sense the language class LRR is more robust than the class DCFL.

This paper is structured as follows. In Section 2 we introduce the two-way restarting list automaton (RLA) and establish various basic notions on computations of RLAs. In Section 3 we introduce RLWW-automata as special types of RLAs, present some constraints for them, and derive the aforementioned results on monotone deterministic RLAs. Finally, Section 4 summarizes the main results. The paper concludes with Section 5, in which our results are commented on and problems for future work are presented.

## 2. Definitions

By  $\subset$  we denote the proper subset relation, and  $\mathcal{P}(S)$  denotes the power set of a set  $S$ . Throughout the paper,  $\lambda$  will denote the empty word, and  $\mathbb{N}$  and  $\mathbb{N}_+$  will denote the set of all non-negative integers and the set of all positive integers, respectively. Further, for an alphabet  $\Gamma$  and an integer  $k \in \mathbb{N}_+$ ,  $\Gamma^k$  denotes the set of all words of length  $k$  over  $\Gamma$ , and  $\Gamma^{\leq k}$  is the set of all words over  $\Gamma$  of length at most  $k$ . We will sometimes use regular expressions instead of the corresponding regular languages. We start with the definition of the two-way restarting list automaton.

**Definition 2.1** *A two-way restarting list automaton, an RLA for short, is a one-tape machine that is described by an 8-tuple  $M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta)$ . Here  $Q$  is the finite set of states,  $\Sigma$  is the finite input alphabet,  $\Gamma$  is the finite working alphabet that includes  $\Sigma$ , the symbols  $\triangleright, \triangleleft \notin \Gamma$  are the markers for the left and right border of the work space, respectively,  $q_0 \in Q$  is the initial state,  $k \geq 1$  is the size of the read/write window, and*

$$\delta : Q \times \mathcal{PC}^{(k)} \rightarrow \mathcal{P}((Q \times (\{\text{MVR}, \text{MVL}\} \cup \{\text{W}(v), \text{SL}(v) \mid v \in \mathcal{PC}^{(n)}, 0 \leq n \leq k\})) \cup \{\text{Restart}, \text{Accept}, \text{Reject}\})$$

*is the transition function. Here  $\mathcal{PC}^{(k)} := (\triangleright \cdot \Gamma^{k-1}) \cup \Gamma^k \cup (\Gamma^{\leq k-1} \cdot \triangleleft) \cup (\triangleright \cdot \Gamma^{\leq k-2} \cdot \triangleleft)$  is the set of possible contents of the read/write window of  $M$ .*

*The transition function describes seven different types of transition steps (or operations), where we assume that  $M$  is in state  $q \in Q$  and that it sees the word  $u \in \mathcal{PC}^{(k)}$  in its read/write window:*

1. *A move-right step  $(q, u) \rightarrow (q', \text{MVR})$  assumes that  $(q', \text{MVR}) \in \delta(q, u)$ , where  $q' \in Q$  and  $u \neq \triangleleft$ . This move-right step causes  $M$  to shift the read/write window one position to the right and to enter state  $q'$ .*
2. *A move-left step  $(q, u) \rightarrow (q', \text{MVL})$  assumes that  $(q', \text{MVL}) \in \delta(q, u)$ , where  $q' \in Q$  and  $u \notin \triangleright \cdot \Gamma^* \cdot \{\lambda, \triangleleft\}$ . It causes  $M$  to shift the read/write window one position to the left and to enter state  $q'$ .*
3. *A rewrite step  $(q, u) \rightarrow (q', \text{W}(v))$  assumes that  $(q', \text{W}(v)) \in \delta(q, u)$ , where  $q' \in Q$ ,  $v \in \mathcal{PC}^{(k)}$ ,  $|v| = |u|$ , and the sentinels are at the same positions in  $u$  and  $v$  (if any). It causes  $M$  to replace the contents  $u$  of the read/write window by the string  $v$ , and to enter state  $q'$ . The window does not change its position.*

4. An SL-step  $(q, u) \rightarrow (q', \text{SL}(v))$  assumes that  $(q', \text{SL}(v)) \in \delta(q, u)$ , where  $q' \in Q$  and  $v \in \mathcal{PC}^{(k')}$  for some  $k' < k$ ,  $v$  is shorter than  $u$ , containing all sentinels from  $u$ . It causes  $M$  to replace  $u$  by  $v$ , to enter state  $q'$ , and to shift the window by  $|u| - |v|$  items to the left – but to the left sentinel  $\triangleright$  at most (the contents of the window is ‘completed’ from the left; the distance to the left sentinel decreases if the window position was not already at  $\triangleright$ , while the distance to the right sentinel is preserved unless the window was near the left sentinel  $\triangleright$ , in which case the distance to the right sentinel may decrease).
5. A restart step  $(q, u) \rightarrow \text{Restart}$  assumes that  $\text{Restart} \in \delta(q, u)$ . It causes  $M$  to move its read/write window to the left end of the tape, so that the first symbol it sees is the left sentinel  $\triangleright$ , and to reenter the initial state  $q_0$ .
6. An accept step  $(q, u) \rightarrow \text{Accept}$  assumes that  $\text{Accept} \in \delta(q, u)$ . It causes  $M$  to halt and accept.
7. A reject step  $(q, u) \rightarrow \text{Reject}$  assumes that  $\text{Reject} \in \delta(q, u)$ . It causes  $M$  to halt and reject.

A configuration of an RLA  $M$  is a string  $\alpha q \beta$ , where  $q \in Q$ , and either  $\alpha = \lambda$  and  $\beta \in \{\triangleright\} \cdot \Gamma^* \cdot \{\triangleleft\}$  or  $\alpha \in \{\triangleright\} \cdot \Gamma^*$  and  $\beta \in \Gamma^* \cdot \{\triangleleft\}$ ; here  $q$  represents the current state,  $\alpha\beta$  is the current contents of the tape, and it is understood that the window contains the first  $k$  symbols of  $\beta$  or all of  $\beta$  when  $|\beta| \leq k$ . A *restarting configuration* is of the form  $q_0 \triangleright w \triangleleft$ , where  $w \in \Gamma^*$ ; if  $w \in \Sigma^*$ , then  $q_0 \triangleright w \triangleleft$  is an *initial configuration*. We see that any initial configuration is also a restarting configuration and that any restart transfers  $M$  into a restarting configuration.

In general, the RLA  $M$  is *nondeterministic*, that is, there can be two or more steps (instructions) with the same left-hand side  $(q, u)$ , and thus, there can be more than one computation for an (input) word. If this is not the case, then  $M$  is *deterministic*. We will use the prefix *det-* to denote deterministic RLAs.

An *input word*  $w \in \Sigma^*$  is *accepted by*  $M$ , if there is a computation which starts with the initial configuration  $q_0 \triangleright w \triangleleft$  and ends by executing an accept step. By  $L(M)$  we denote the language consisting of all input words accepted by  $M$ ; we say that  $M$  *recognizes (accepts) the input language*  $L(M)$ .

A *basic (or characteristic) word*  $w \in \Gamma^*$  is *accepted by*  $M$ , if there is a computation which starts with the restarting configuration  $q_0 \triangleright w \triangleleft$  and ends by executing an accept step. By  $L_C(M)$  we denote the language consisting of all basic words accepted by  $M$ ; we say that  $M$  *recognizes (accepts) the basic (characteristic) language*  $L_C(M)$ . Obviously,  $L_C(M) \cap \Sigma^* = L(M)$ .

In the following we only consider finite computations of RLAs which end either by an accept or a reject step.

– **Cycles, tails:** Any finite computation of an RLA  $M$  consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the window moves along the tape performing non-restarting steps until a restart step is performed, which completes the current cycle. Thus, after each cycle a new restarting configuration is reached. If no further restart step is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*.



– **Cycle-rewritings:** By the notation  $q_0 \triangleright u \triangleleft \vdash_M^c q_0 \triangleright v \triangleleft$  we denote a cycle of  $M$  that begins with the restarting configuration  $q_0 \triangleright u \triangleleft$  and ends with the restarting configuration  $q_0 \triangleright v \triangleleft$ . Through this relation we define the relation of *cycle-rewriting* by  $M$ . We write  $u \Rightarrow_M^c v$  iff  $q_0 \triangleright u \triangleleft \vdash_M^c q_0 \triangleright v \triangleleft$  holds. The relation  $u \Rightarrow_M^* v$  is the reflexive and transitive closure of  $u \Rightarrow_M^c v$ .

We point out that the cycle-rewriting is a very important feature of RLAs.

– **Reductions:** If  $u \Rightarrow_M^c v$  is a cycle-rewriting by  $M$  such that  $|u| > |v|$ , then  $u \Rightarrow_M^c v$  is called a *reduction* by  $M$ .

Often we will (implicitly) use the following obvious fact.

**Fact 1 (Error Preserving Property for basic languages of RLAs).**

*Let  $M$  be an RLA. If  $u \Rightarrow_M^* v$  and  $u \notin L_C(M)$ , then  $v \notin L_C(M)$ .*

Observe that this fact only concerns the basic language of an RLA. In general, the Error Preserving Property does not apply to the input language of an RLA, as a cycle-rewriting  $u \Rightarrow_M^c v$  may rewrite a word  $u$  containing non-input symbols (and which therefore does not belong to the input language of  $M$ ) into a word  $v$  that belongs to the input language  $L(M)$ . We can apply a similar observation to the following fact.

**Fact 2 (Correctness Preserving Property for basic languages of det-RLAs).**

*Let  $M$  be a deterministic RLA. If  $u \Rightarrow_M^* v$  and  $u \in L_C(M)$ , then  $v \in L_C(M)$ .*

We remark that both these properties are based on the operation of restart.

## 2.1. Further Refinements and Constraints on RLAs

Here we introduce some constrained types of rewriting steps.

A *delete-left step*  $(q, u) \rightarrow (q', \text{DL}(v))$  is an SL-step  $(q, u) \rightarrow (q', \text{SL}(v))$  such that  $v$  is a proper subsequence of  $u$ , containing all sentinels from  $u$  (if any). Thus, a delete-left step is an SL-step which can only delete symbols.

A *contextual-left step*  $(q, u) \rightarrow (q', \text{CL}(v))$  is an SL-step  $(q, u) \rightarrow (q', \text{SL}(v))$ , where  $u = v_1 u_1 v_2 u_2 v_3$  and  $v = v_1 v_2 v_3$  for some  $v_1, u_1, v_2, u_2, v_3$ , such that  $v$  contains all sentinels from  $u$  (if any). Thus, a contextual-left step is a delete-left step which can delete at most two factors. This corresponds to an inverse of the operation of context-adjointing used in contextual grammars [9].

The set  $\text{OG} = \{\text{MVR}, \text{MVL}, \text{W}, \text{SL}, \text{DL}, \text{CL}, \text{Restart}\}$  represents the set of types of steps (operations), which can be used for characterizations of subclasses of RLAs. This set does not contain the operations **Accept** and **Reject**, corresponding to halting steps, as they are used for all RLAs. For a set  $\text{T} \subseteq \text{OG}$ , we denote by  $\text{T}$ -automata the subset of RLAs which only use

transition steps from the set  $T \cup \{\text{Accept, Reject}\}$ . For example,  $\{\text{MVR, W}\}$ -automata are RLAs which only use move-right steps, W-steps, accept steps, and reject steps.

– **Monotonicity of rewritings:** We introduce various notions of *monotonicity* as important types of constraints for computations of RLAs. They are useful for characterizations of the class of (deterministic) context-free languages.

Let  $M$  be an RLA, and let  $C = C_k, C_{k+1}, \dots, C_j$  be a sequence of configurations of  $M$ , where  $C_{i+1}$  is obtained from  $C_i$  by a single transition step of  $M$  for all  $k \leq i < j$ . We say that  $C$  is a *subcomputation* of  $M$ .

Let  $RW \subseteq \{\text{W, SL, DL, CL}\}$ . Then we denote by  $W(C, RW)$  the maximal subsequence of  $C$ , which contains those configurations from  $C$  that correspond to  $RW$ -steps (that is, those configurations in which a transition step of one of the types from the set  $RW$  is applied). We say that  $W(C, RW)$  is the *rewriting sequence* of  $C$  determined by  $RW$ .

Let  $C$  be a subcomputation of an RLA  $M$ , and let  $C_w = \triangleright \alpha q \beta \triangleleft$  be a configuration from  $C$ . Then  $|\beta \triangleleft|$  is the *right distance* of  $C_w$ , which is denoted by  $D_r(C_w)$ .

We say that a rewriting sequence  $W(C, RW) = (C_1, C_2, \dots, C_n)$  is *RW-monotone* if  $D_r(C_1) \geq D_r(C_2) \geq \dots \geq D_r(C_n)$ .

A subcomputation  $C$  of  $M$  is *RW-monotone* if  $W(C, RW)$  is *RW-monotone*. We say that  $M$  is *RW-monotone* if each of its (sub-) computations is *RW-monotone*. Further, we say that  $M$  is *monotone*, abbreviated by the prefix **mon-**, if it is  $\{\text{W, SL, DL, CL}\}$ -monotone, that is, it is monotone with respect to any type of (allowed) rewriting for the corresponding type of automaton.

Finally, we call  $M$  *completely monotone* if  $D_r(C_1) \geq D_r(C_2)$  holds whenever configuration  $C_2$  can be obtained by a single step from configuration  $C_1$ .

We close this subsection with an observation on complete monotonicity.

**Fact 3** *Let  $M$  be a  $\{\text{MVR, SL, W}\}$ -automaton. Then  $M$  is completely monotone.*

*Notations.* For any class  $\mathbf{B}$  of automata,  $\mathcal{L}(\mathbf{B})$  will denote the class of input languages that are recognized by automata from  $\mathbf{B}$  and  $\mathcal{L}_C(\mathbf{B})$  will denote the class of basic languages that are recognized by automata from  $\mathbf{B}$ .

**Remark on PDA.** It is not hard to see that a  $\{\text{MVR, SL, W}\}$ -automaton with a window of size 1 is a type of normalized pushdown automaton. The top of the pushdown is represented by the position of the window, and the content of the pushdown is represented by the part of the tape between the left sentinel and the position of the window. In fact, in a very similar way the pushdown automaton was introduced by Chomsky [4]. A  $\{\text{MVR, SL, W}\}$ -automaton with a window of size  $k \geq 2$  can be interpreted as a pushdown automaton with a  $k$ -lookahead and with a limited look under the top of the pushdown. A deterministic PDA can be simulated by a **det-** $\{\text{MVR, SL, W}\}$ -automaton with a window of size 1.

	auxiliary symbols possible (-WW)			no auxiliary symbols (-W)		
	SL-steps	DL-steps only	CL-steps only	SL-steps	DL-steps only	CL-steps only
MVL-steps (RL-)	RLWW	RLWWD	RLWWC	RLW	RLWD	RLWC
no MVL-steps, rewrite followed by restart (R-)	RWW	RWWD	RWWC	RW	RWD	RWC

Table 1: Different variants of RLWW-automata

### 3. RLWW-Automata

Here we first describe RLWW-automata and some of their subclasses in our terminology.

An *RLWW-automaton*  $M$  is a  $\{\text{MVR}, \text{MVL}, \text{SL}, \text{Restart}\}$ -automaton, which uses an SL-step exactly once in each cycle and at most once in each tail computation. We see that for RLWW-automata, all cycle-rewritings are reductions.

From the above statements we see that the input language and the basic language recognized by an RLWW-automaton is a context-sensitive language. In the following we will explicitly introduce various notions and notation for subclasses of RLWW-automata.

An *RLW-automaton* is an RLWW-automaton the working alphabet of which coincides with its input alphabet. Note that in this situation, each restarting configuration is necessarily an initial configuration.

An *RLWD-automaton* is an RLW-automaton all rewrite steps of which are DL-steps, and an *RLWC-automaton* is an RLWD-automaton all rewrite steps of which are CL-steps. Further, an *RLWWC-automaton* (that is, an *RLWW-automaton in Marcus contextual form*) is an RLWW-automaton all rewrite steps of which are CL-steps. Similarly, an *RLWWD-automaton* is an RLWW-automaton all rewrite steps of which are DL-steps. Observe that when concentrating on input languages, then DL- and CL-steps ensure that no auxiliary symbols can ever occur on the tape; if, however, we are interested in basic languages, then auxiliary symbols can play an important role even though a given RLWW-automaton uses only DL- or CL-steps. Therefore, we distinguish between RLWWC- and RLWC-automata and between RLWWD- and RLWD-automata.

An *RWW-automaton* is an RLWW-automaton which restarts immediately after executing a rewrite step and does not use any MVL-steps. From these automata, we obtain *RW-*, *RWD-*, *RWC-*, *RWWD-*, and *RWWC-automata*. Table 1 gives an overview of the various subclasses of RLWW-automata defined above.

For our investigations the following notions play an important role, since they formulate correctness preserving properties also for RLAs that do not use the restart operation.

**Definition 3.1** *Let  $M$  be an RLA.*

- (a)  $M$  is said to satisfy the Complete Weak Correctness Preserving Property (CWCPP) for its

basic (input) language if, for each accepting computation  $C_0, C_1, \dots, C_n$  of  $M$ ,  $u_j \in L_C(M)$  ( $u_j \in L(M)$ ) for all  $j = 0, 1, \dots, n$ , where  $u_j$  is the contents of the tape in configuration  $C_j$  ( $0 \leq j \leq n$ ).

- (b)  $M$  is said to satisfy the Complete Strong Correctness Preserving Property (CSCPP) for its basic (input) language if, for each computation  $C_0, C_1, \dots, C_n$  of  $M$ , we have that  $u_j \in L_C(M)$  ( $u_j \in L(M)$ ) for all  $j = 0, 1, \dots, n$ , if  $u_i \in L_C(M)$  ( $u_i \in L(M)$ ) for some  $i$ . Here  $u_j$  is the contents of the tape in configuration  $C_j$  ( $0 \leq j \leq n$ ).

In contrast to Fact 1 and Fact 2, the Complete Weak and Strong Correctness Preserving Properties do not depend on the operation of restart. They express the fact that, in an accepting computation, each and every operation of the automaton  $M$  considered preserves the property of the tape contents to belong to the language  $L_C(M)$  ( $L(M)$ ), that is, no intermediate information is stored on the tape. This is an important novelty of these notions. We illustrate them by a simple example.

**Example 3.2** In [12], Example 3, an RWW-automaton  $M$  is presented that accepts the input language  $L = \{a^n b^n c, a^n b^{2n} d \mid n \geq 0\}$ . Given a word  $a^m b^n x$  as input, where  $m, n \geq 2$  and  $x \in \{c, d\}$ ,  $M$  either performs the cycle  $q_0 \triangleright a^m b^n x \triangleleft \vdash_M^c q_0 \triangleright a^{m-1} C b^{n-1} x \triangleleft$  or the cycle  $q_0 \triangleright a^m b^n x \triangleleft \vdash_M^c q_0 \triangleright a^{m-1} D b^{n-2} x \triangleleft$ , where  $C$  and  $D$  are auxiliary symbols, in this way guessing whether  $x = c$  or  $x = d$ . In the former case it then repeatedly performs CL-steps rewriting  $aCb$  into  $C$ , in this way checking whether  $m = n$ , and it accepts on reaching the tape contents  $\triangleright Cc \triangleleft$ , while in the latter case it repeatedly performs CL-steps rewriting  $aDbb$  into  $D$ , in this way checking whether  $m = 2n$ , and it accepts on reaching the tape contents  $\triangleright Dd \triangleleft$ . Thus, we see that in an accepting computation of  $M$ , all but the initial configuration contain an occurrence of an auxiliary symbol, which shows that  $M$  does not satisfy the Complete Weak Correctness Preserving Property for its input language.

On the other hand, a deterministic RLW-automaton  $M'$  for  $L$  can proceed as follows. In each cycle, it first scans the given input completely and checks whether the last letter is a  $c$  or a  $d$ , and then it can either delete a factor  $ab$  or a factor  $abb$ , respectively. Thus, the (accepting) computations of this automaton  $M'$  are much more transparent than those of the RWW-automaton  $M$ . In fact, the det-RLW-automaton  $M'$  satisfies the Complete Strong Correctness Preserving Property for its input language.

Actually, we have the following result due to the fact that RLW-automata have identical input and working alphabets.

**Fact 4 (Equality of Languages for RLW-Automata).**

For each RLW-automaton  $M$ ,  $L(M) = L_C(M)$ .

Note that if an RLWW-automaton  $M$  starts its accepting computation in a restarting configuration with a word  $w$  on its tape, then  $w$  belongs to its basic language  $L_C(M)$ . However, if  $M$  performs an SL-step during a tail computation, the resulting tape contents need not belong to its basic language. This is illustrated by the following simple example.

**Example 3.3** Let  $M$  be the RLWC-automaton with a window of size 2 on  $\Sigma = \{a, b, c\}$  that works as follows:

- (1)  $M$  scans its tape from left to right, counting the number of occurrences of the letter  $b$  modulo two.
- (2) If it encounters an occurrence of the letter  $c$ , then  $M$  deletes the first occurrence of the letter  $c$ . If there is no other  $c$  after the one deleted, then  $M$  accepts; otherwise it restarts.
- (3) If  $M$  does not encounter any occurrence of the letter  $c$ , then it deletes the last letter and accepts, if the number of occurrences of the letter  $b$  is an uneven number.

It is not hard to see that  $M$  is deterministic and monotone and that

$$L_C(M) = L(M) = \{w \in \Sigma^+ \mid |w|_c \geq 1 \text{ or } (|w|_c = 0 \text{ and } |w|_b \equiv 1 \pmod{2})\}.$$

For the input  $w_1 = aacbb \in L(M)$ ,  $M$  executes an accepting tail computation that contains a CL-step which rewrites the word  $w_1$  into the word  $z_1 = aabb \notin L(M)$  (see (2)). This means that  $M$  does not satisfy the CWCPP for its basic (input) language.

On the other hand, for the input  $w_2 = aabb \notin L(M)$ ,  $M$  executes a rejecting tail computation that contains a CL-step that rewrites the word  $w_2$  into the word  $z_2 = aab \in L(M)$  (see (3)). For this reason the Error Preserving Property (see Fact 1) has been formulated only for cycle-rewritings.

However, we can easily modify  $M$  into an RLWW-automaton  $M'$  such that it performs exactly the same cycles as  $M$ , but it never rewrites during any tail computation. Hence, we have the following simple, but very important facts that illustrate that the computations of RLWW-automata can be transparent with respect to their basic languages.

**Fact 5 (Complete Weak Correctness Preserving Property for RLWW-Automata).** For each RLWW-automaton  $M$ , there exists an RLWW-automaton  $M'$  such that  $M'$  satisfies the Complete Weak Correctness Preserving Property for its basic language and  $u \vdash_M^c w$  iff  $u \vdash_{M'}^c v$ .

By Fact 4 this means that each RLW-automaton can be turned into an RLW-automaton that satisfies the Complete Weak Correctness Preserving Property for its input language, too.

**Fact 6 (Complete Strong Correctness Preserving Property for det-RLWW-Automata).**

For each deterministic RLWW-automaton  $M$ , there exists a deterministic RLWW-automaton  $M'$  such that  $M'$  satisfies the Complete Strong Correctness Preserving Property for its basic language and  $u \vdash_M^c w$  iff  $u \vdash_{M'}^c v$ .

Again by Fact 4 this means that each deterministic RLW-automaton can be turned into a deterministic RLW-automaton that satisfies the Complete Strong Correctness Preserving Property for its input language, too.

### 3.1. Robustness of Monotone Deterministic RLAs

The notion of monotonicity has been considered in various papers. The following results have been established, where DCFL denotes the class of deterministic context-free languages and LRR denotes the class of *left-to-right regular languages* from [17].

**Theorem 3.4** [6, 13]

- (a)  $\text{DCFL} = \mathcal{L}(\text{det-mon-RWC}) \subsetneq \mathcal{L}(\text{det-mon-RLWC})$ .
- (b)  $\text{LRR} = \mathcal{L}(\text{det-mon-RLWW}) = \mathcal{L}(\text{det-mon-RLWD})$ .

In the following, we will improve upon the result in (b) above by showing that, for any det-mon-RLWW-automaton  $M_a$ , there is even an equivalent det-mon-RLWC-automaton  $M_b$ . While the det-mon-RLWW-automaton  $M_a$  will in general not even satisfy the Error Preserving Property for its input language (cf. Fact 1), the det-mon-RLWC-automaton  $M_b$  can be made to fulfill the Complete Strong Correctness Preserving Property for its input language (Fact 6) and, moreover, all its rewrite steps are just contextual deletions. This means in particular that in general the reductions of  $M_a$  and  $M_b$  will differ substantially.

**Theorem 3.5** *For each det-mon-RLWW-automaton  $M_a$ , there exists a det-mon-RLWC-automaton  $M_b$  such that  $L(M_a) = L(M_b)$ .*

*Proof.* Let  $M_a = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, k, \delta)$  be a det-mon-RLWW-automaton. Then the language  $L = L(M_a)$  belongs to the class LRR (Theorem 3.4 (b)). Hence, according to the proof of Theorem 2.1 of [17], there exists a deterministic sequential right-to-left transducer  $G$  such that  $L_1 = G(L)$  is a deterministic context-free language. In fact, proceeding from right to left,  $G$  outputs a pair of the form  $(a, p_a)$  for each letter  $a$  read, where  $p_a$  is some additional information that depends on the suffix already read by  $G$ . Further, we see from Theorem 3.4 (a) that there exists a det-mon-RWC-automaton  $M_1$  for the language  $L_1$ . By combining the transducer  $G$  and the det-mon-RWC-automaton  $M_1$ , we obtain a {MVR, MVL, W, CL, Restart}-automaton  $M_2$  for the language  $L_1$  that works as follows:

- (1) Given a word  $w \in \Sigma^*$  as input,  $M_2$  first rewrites  $w$  from right to left by replacing each letter  $a$  by the pair  $(a, p_a)$  that the right-to-left transducer  $G$  produces on input  $w$  for this letter  $a$ . Thus, the input  $w$  is rewritten into the word  $G(w)$ , and in this phase, only MVR-, MVL-, and W-steps are used.
- (2) Once the input has been rewritten completely,  $M_2$  simulates the det-mon-RWC-automaton  $M_1$  on the word  $G(w)$ . During this phase, MVR-, CL-, and Restart-steps are used.

Thus,  $M_2$  accepts on input  $w$  iff  $M_1$  accepts on input  $G(w)$ , that is, iff  $w \in L$ .

Next we simulate  $M_2$  by a det-mon-RLWC-automaton  $M_3$ . Essentially,  $M_3$  behaves like  $M_2$  without actually doing the rewrites in phase (1). So in this phase  $M_3$  simply reads the word  $w$  from right to left simulating the transducer  $G$  in its finite-state control, just remembering the output produced by  $G$  for all the letters currently inside the window. This is enough for starting the simulation of  $M_1$  in phase (2), but as soon as the window must move right, information on

the result of  $G$  for the new letter that enters the window is missing. However, here we can use the symmetric (that is, right-to-left) variant of the following lemma, which is taken from [1], pages 212–213.

**Lemma 3.6** *Let  $A$  be a deterministic finite-state acceptor. For each word  $x$  and each integer  $i$ ,  $1 \leq i \leq |x|$ , let  $q_A(x, i)$  be the state of  $A$  after processing the prefix of length  $i$  of  $x$ . Then there exists a deterministic two-way finite-state acceptor  $B$  such that, for each input  $x$  and each  $i \in \{2, 3, \dots, |x|\}$ , the following condition is satisfied:*

- *when  $B$  starts its computation on  $x$  in state  $q_A(x, i)$  with its head on the  $i$ -th letter of  $x$ , then  $B$  finishes its computation in state  $q_A(x, i - 1)$  with its head on the  $(i - 1)$ -st letter of  $x$ .*

Now we can describe the det-mon-RLWC-automaton  $M_3$ . Each cycle of the det-mon-RWC-automaton  $M_1$  on input  $G(w)$  is simulated by a cycle of  $M_3$  on input  $w$  as follows:

- (1) First  $M_3$  simulates phase (1) of the computation of  $M_2$  without doing the rewrites. In this way it determines the  $k$  leftmost letters of  $G(w)$  in its finite-state control, where  $k$  is the size of the window of  $M_1$ .
- (2) Next  $M_3$  simulates the behaviour of  $M_1$  on input  $G(w)$  step by step until it reaches a configuration in which  $M_1$  applies a CL-step in the current cycle. Each time  $M_3$  must simulate a MVR-step, it runs the two-way finite-state automaton  $B$  from the symmetric variant of Lemma 3.6 that corresponds to the deterministic right-to-left sequential transducer  $G$  in order to recover the value of  $G$  for the new rightmost letter in its window. Note that at the beginning of the simulation  $M_3$  knows the part of  $G(w)$  that is contained in the window of  $M_1$  (see (1)). Hence, by using  $B$  it is able to satisfy this condition for each successive step.
- (3) Finally,  $M_3$  simulates the current CL-step of  $M_1$  by deleting the appropriate symbols, that is, it executes a CL-step as well.

For a word  $w = a_1 a_2 \cdots a_n \in L$ , where  $a_1, a_2, \dots, a_n \in \Sigma$ , the word  $G(w) \in L_1$  has the form  $G(w) = (a_1, p_1)(a_2, p_2) \cdots (a_n, p_n)$ . If  $n$  is sufficiently large, then on the word  $G(w)$ , the det-mon-RWC-automaton  $M_1$  executes a reduction  $G(w) \Rightarrow_{M_1}^c z$  that deletes (one or) two factors from  $G(w)$ , that is,  $z$  has the form

$$z = (a_1, p_1)(a_2, p_2) \cdots (a_r, p_r)(a_{r+i+1}, p_{r+i+1})(a_{r+i+2}, p_{r+i+2}) \cdots \\ (a_{r+i+s}, p_{r+i+s})(a_{r+i+s+j+1}, p_{r+i+s+j+1})(a_{r+i+s+j+2}, p_{r+i+s+j+2}) \cdots (a_n, p_n).$$

Here the factors

$$(a_{r+1}, p_{r+1})(a_{r+2}, p_{r+2}) \cdots (a_{r+i}, p_{r+i}) \text{ and } (a_{r+i+s+1}, p_{r+i+s+1}) \cdots (a_{r+i+s+j}, p_{r+i+s+j})$$

are deleted, which implies that  $i + j > 0$  and  $r + i + s + j - r = s + i + j \leq k$ . As  $M_1$  satisfies the Correctness Preserving Property for its input language (see Fact 2), the word  $z$  belongs to the language  $L_1 = G(L)$ , which implies that  $z = G(x)$ , where  $x = a_1 a_2 \cdots a_r a_{r+i+1} a_{r+i+2} \cdots a_{r+i+s} a_{r+i+s+j+1} a_{r+i+s+j+2} \cdots a_n \in L$ .

Given the word  $w$  as input, the automaton  $M_3$  executes the reduction  $w \Rightarrow_{M_3}^c x$ . In the next cycle, it starts with the tape contents  $x$ , first simulating the deterministic right-to-left sequential

transducer  $G$  on  $x$ . As  $G$  is deterministic, and as the word  $x$  belongs to the language  $L$ , we see that on input  $x$ ,  $G$  produces the word  $G(x) = z$ . Thus, the **SL**-step of  $M_3$  on  $x$  occurs at the same position as the **SL**-step of  $M_1$  on  $z$ .

It follows that  $L(M_3) = L$  and that the **CL**-steps of  $M_3$  just correspond to the **CL**-steps of  $M_1$ . Hence, we see that  $M_3$  is also monotone. This completes the proof of Theorem 3.5.  $\square$

**Corollary 3.7** *For each det-mon-RLWC-automaton  $M_r$ , there exists a det-mon- $\{\text{MVR}, \text{MVL}, \text{CL}\}$ -automaton  $M$  such that  $L(M) = L(M_r)$ , and  $M$  has the Complete Strong Correctness Preserving Property for its input language  $L(M)$ .*

*Proof.* It is not hard to see that each det-mon-RLWC-automaton  $M_r$  can be simulated by a det-mon- $\{\text{MVR}, \text{MVL}, \text{CL}\}$ -automaton  $M$ , which simply copies all move and **CL**-steps of  $M_r$ , but that replaces all restart steps of  $M_r$  by a series of **MVL**-steps, where the last one transfers  $M$  into the initial state on seeing the left sentinel.  $\square$

On the other hand, we also have the following simulation.

**Proposition 3.8** *For each det-mon- $\{\text{MVR}, \text{MVL}, \text{SL}\}$ -automaton  $M_a$ , there exists a det-mon-RLWW-automaton  $M_b$  such that  $L(M_a) = L(M_b)$ .*

*Proof.* Let  $M_a$  be a det-mon- $\{\text{MVR}, \text{MVL}, \text{SL}\}$ -automaton with a set of states  $Q$ , an input alphabet  $\Sigma$ , a tape alphabet  $\Gamma$ , and a window of size  $k \geq 1$ . We describe an RLWW-automaton  $M_b$  with input alphabet  $\Sigma$  and tape alphabet  $\Gamma \cup \{[q, a] \mid q \in Q, a \in \Gamma\}$  that has a window of size  $k + 1$ . This RLWW-automaton works as follows:

1. On an input word  $w \in \Sigma^*$ ,  $M_b$  simulates the **MVR**- and **MVL**-steps of  $M_a$  until  $M_a$  is to execute an **SL**-step.
2. An **SL**-step  $\delta_a(q, u) \rightarrow (q', v)$  of  $M_a$  is simulated by  $M_b$  by executing the **SL**-step  $\delta_b(q, uc) \rightarrow (q', v[q', c])$ , where  $u \in \Gamma^k$ ,  $v \in \Gamma^*$  satisfying  $|v| < |u|$ , and  $c \in \Gamma$ , and then  $M_b$  restarts immediately. Thus,  $M_b$  executes the same **SL**-step as  $M_a$ , but it encodes the state of  $M_a$  reached by that **SL**-step together with the letter immediately to the right of the rewritten factor. Here some technical adjustments must be made in case  $u$  is the suffix of the current tape contents, that is, it is followed by the right sentinel  $\triangleleft$  only. In that case the state information is encoded together with the last letter of  $v$  or, in case that  $v = \lambda$ , with the first letter to the left of  $u$  (and  $v$ ).
3. After a restart  $M_a$  scans the tape and looks for the rightmost occurrence of a pair of the form  $[q', c]$  on the tape. As the **SL**-steps of  $M_b$  just simulate the **SL**-steps of  $M_a$ ,  $M_b$  is monotone, and hence, the information on the last **SL**-step executed before the current cycle is encoded in the rightmost of these code symbols. After locating  $[q', c]$ , the automaton  $M_b$  continues the simulation from state  $q'$  while ignoring the state information in any pair  $[\bar{q}, \bar{c}]$  it sees on the tape.

It is now easily seen that  $M_b$  is a det-mon-RLWW-automaton satisfying  $L(M_b) = L(M_a)$ .  $\square$

Let us note that in general a det-mon- $\{\text{MVR}, \text{MVL}, \text{SL}\}$ -automaton does not ensure any type of correctness preserving property for its input language.



In summary, the results above yield the following identities, where the prefix **cscpp**- denotes the Complete Strong Correctness Preserving Property.

$$\begin{aligned} \text{Corollary 3.9 } \text{LRR} &= \mathcal{L}(\text{det-mon-RLWW}) = \mathcal{L}(\text{det-mon-cscpp-RLWC}) \\ &= \mathcal{L}(\text{det-mon-}\{\text{MVR,MVL,SL}\}) = \mathcal{L}(\text{det-mon-cscpp-}\{\text{MVR,MVL,SL}\}) \\ &= \mathcal{L}(\text{det-mon-cscpp-}\{\text{MVR,MVL,CL}\}). \end{aligned}$$

### 3.2. Strong Cyclic Form

Here we carry some restricted forms of restarting automata called *weak cyclic form* (wcf) (see [11]) and *strong cyclic form* (see [5]) over to RLAs. An RLA  $M$  is said to be in *weak cyclic form* if  $|uv| \leq k$  for each accepting configuration  $\triangleright uqv \triangleleft$  of  $M$ , where  $k$  is the size of the read/write window of  $M$ . Thus, before  $M$  can accept, it must erase sufficiently many letters from its tape. Further,  $M$  is said to be in *strong cyclic form* if  $|uv| \leq k$  for each accepting and each rejecting configuration  $\triangleright uqv \triangleleft$  of  $M$ , where  $k$  is the size of the read/write window of  $M$ . Thus, before  $M$  can halt, either accepting or rejecting, it must erase sufficiently many letters from its tape. We express the latter property by the prefix **scf**-.

**Theorem 3.10** *For each det-mon-RLWC-automaton  $M$ , there is a det-mon-RLWC-automaton  $M_{\text{scf}}$  in strong cyclic form such that  $L(M) = L(M_{\text{scf}})$  and, for all  $u \Rightarrow_M^c v$ , also  $u \Rightarrow_{M_{\text{scf}}}^c v$ .*

*Proof.* Let  $M$  be a det-mon-RLWC-automaton. The set of words  $z$  for which  $M$ , starting from the restarting configuration  $q_0 \triangleright z \triangleleft$ , will execute an accepting tail computation is a regular sublanguage  $L_{\text{sim}}$  of  $L(M)$ . Analogously, the set of words  $z$  for which  $M$ , starting from the restarting configuration  $q_0 \triangleright z \triangleleft$ , will execute a rejecting tail computation (or get stuck in a configuration to which no operation applies) is a regular sublanguage  $L_{\text{nsim}}$  of the complement of  $L(M)$ . From the Pumping Lemma for regular languages we conclude that there exists a constant  $c$  such that, for each word  $z_1 \in L_{\text{sim}}$ , if  $|z_1| \geq c$ , then  $z_1$  has a factorization of the form  $z_1 = uvw$  such that  $|v| \geq 1$ ,  $|vw| \leq c$ , and  $uv^i w \in L_{\text{sim}}$  for each  $i \geq 0$ . Analogously, for each word  $z_2 \in L_{\text{nsim}}$ , if  $|z_2| \geq c$ , then  $z_2$  has a factorization of the form  $z_2 = uvw$  such that  $|v| \geq 1$ ,  $|vw| \leq c$ , and  $uv^i w \in L_{\text{nsim}}$  for each  $i \geq 0$ .

Let  $A_+$  and  $A_-$  be DFAs for the languages  $L_{\text{sim}}$  and  $L_{\text{nsim}}$ , respectively. Now the det-mon-RLWC-automaton  $M_{\text{scf}}$  has a window of size  $k' = \max\{k, c + 1\}$ , where  $k$  is the size of the window of the automaton  $M$ . It will proceed as follows:

1. Every word  $w \in L(M)$  satisfying  $|w| \leq k'$  is accepted immediately, and every word  $z \notin L(M)$  satisfying  $|z| \leq k'$  is rejected immediately.
2. Starting from a restarting configuration of the form  $q_0 \triangleright z \triangleleft$ , where  $|z| > k'$ ,  $M_{\text{scf}}$  first scans its tape from left to right, simulating both the DFAs  $A_+$  and  $A_-$  in parallel. If one of them would accept, then  $z = uvw$ , where  $|v| \geq 1$ ,  $|vw| < c$ , and  $uw$  is accepted by the same DFA. Hence,  $M_{\text{scf}}$  simply cuts the infix  $v$  from  $z$  by performing a **CL**-step at the end of the tape.
3. If none of  $A_+$  or  $A_-$  accepts the word  $z$ , then  $M_{\text{scf}}$  returns to the left end of the tape and simulates the cycle that  $M$  would execute starting from the corresponding restarting configuration.

Observe that, for each word  $z$ , starting from the restarting configuration  $q_0 \triangleright z \triangleleft$ ,  $M$  either executes a complete cycle, or it accepts, which means that  $z \in L_{\text{sim}}$ , or it rejects (or gets stuck), which means that  $z \in L_{\text{nsim}}$ . As  $M$  is deterministic, these three cases are mutually disjoint. Thus,  $M_{\text{scf}}$  accepts the same language as  $M$ , and as it simulates all the CL-steps of  $M$ , we see that the required property holds for all cycle rewritings of  $M$ . Further, as the delete operations of  $M_{\text{scf}}$  that are obtained from the application of the Pumping Lemma are all executed at the right end of the tape, we see that all computations of  $M_{\text{scf}}$  are monotone. Finally, it follows immediately from the construction that  $M_{\text{scf}}$  is in strong cyclic form.  $\square$

Thus, the det-mon-RLWC-automaton  $M$  from Example 3.3 can be turned into an equivalent automaton  $M_{\text{scf}}$  of the same type that is in strong cyclic form and that avoids the rewrites that  $M$  executes in tail computations.

The above proof can also easily be adapted to yield the following result.

**Proposition 3.11** *For each det-mon- $\{\text{MVR}, \text{MVL}, \text{SL}\}$ -automaton  $M_a$ , there exists a det-mon- $\{\text{MVR}, \text{MVL}, \text{SL}\}$ -automaton  $M_b$  in strong cyclic form such that  $L(M_a) = L(M_b)$ . Moreover, if  $M_a$  has the CSCPP, then so does  $M_b$ .*

## 4. The Main Results

The results of the previous subsections are summarized in the following corollaries.

**Corollary 4.1** *For all  $Y \in \{\lambda, \text{scf}, \text{scf-cscpp}\}$ , the following holds:*

$$\begin{aligned} \mathcal{L}(\text{det-mon-RLWW}) &= \mathcal{L}(\text{det-mon-Y-RLW}) = \\ \mathcal{L}(\text{det-mon-Y-RLWD}) &= \mathcal{L}(\text{det-mon-Y-RLWC}) = \text{LRR}. \end{aligned}$$

Let us note that (deterministic) RLWC-, RLWD-, and RLW-automata can always be modified to satisfy the Complete Weak (Strong) Correctness Preserving Property for input and basic languages, while RLWW-automata do in general not satisfy this property for both types of languages. The RLWW-automata can be modified to satisfy CSCPP only for basic languages.

**Corollary 4.2** *For all  $X \in \{\{\text{MVR}, \text{MVL}, \text{SL}\}, \{\text{MVR}, \text{MVL}, \text{DL}\}, \{\text{MVR}, \text{MVL}, \text{CL}\}\}$  and all  $Y \in \{\lambda, \text{scf}, \text{scf-cscpp}\}$ , the following holds:*

$$\mathcal{L}(\text{det-mon-Y-X}) = \text{LRR}.$$

The latter corollary shows characterizations of the language class LRR by automata with the Complete Strong Correctness Preserving Property on the one hand and without any type of correctness preserving property on the other hand. Moreover, these automata work without the operation of restart while preserving the remaining sets of operations. In that way the language class LRR is more robust than the class DCFL of deterministic context-free languages (see, e.g., [7]).

## 5. Conclusion

For RLAs, we have two types of associated languages. The first type are the basic languages. For lexicalized syntax, basic languages have a similar meaning as the sets of sentential forms for Chomsky's phrase-structure grammars. Basic languages are (implicitly) recognized by categorial grammars ([2, 3]). Categorical grammars are typical formal grammars for lexicalized syntax. The second type of languages are the input languages. Input languages correspond to those languages which are commonly used in automata theory. We have introduced RLWW-automata and RLW-automata as special types of RLAs. We have shown that deterministic RLWW-automata can be easily transformed into deterministic RLWW-automata with the CSCPP for basic languages, and that deterministic RLW-automata can be easily transformed into deterministic RLW-automata with the CSCPP for both basic and input languages. As another step we have presented a transformation from monotone deterministic RLAs without the operation restart into monotone deterministic RLWW-automata with the CSCPP for basic languages. Then the technical main result was a transformation from monotone deterministic RLWW-automata to monotone deterministic RLWC-automata with the CSCPP for basic and input languages. Further, we have given a transformation from monotone deterministic RLWC-automata with the CSCPP for basic and input languages to monotone deterministic RLAs without the operation restart satisfying the CSCPP for input languages. Finally, we have presented a transformation of all the above classes of automata into corresponding automata in the strong cyclic form.

Monotone deterministic RLWC-automata in strong cyclic form ensure a deterministic analysis by reduction for any LRR-language, and for any word (sentence) from such a language, this analysis by reduction yields an unambiguous parsing into a system of immediate constituents which are given by individual reductions and by the final irreducible sentence. We have seen that  $\text{det-mon-cscpp-scf-}\{MVR, MVL, CL\}$ -automata have this ability, too.

Moreover, monotone deterministic RLWC-automata in strong cyclic form ensure a deterministic analysis by reduction for the complement of any LRR-language. Again, this also holds for  $\text{det-mon-cscpp-scf-}\{MVR, MVL, CL\}$ -automata. This property can be used for the localization of syntactical errors and for syntactic error recovery.

In the future, we plan to study monotone deterministic RLWC-automata in strong cyclic form that have a minimal look-ahead window and minimal reductions for a given LRR-language.

## References

- [1] A. V. AHO, J. E. HOPCROFT, J. D. ULLMAN, A general theory of translation. *Mathematical Systems Theory* 3 (1969), 193–221.
- [2] K. AJDUKIEWICZ, Die syntaktische Konnexität. *Studia Philosophica* 1 (1935), 1–27.

- [3] Y. BAR-HILLEL, C. GAIFMAN, E. SHAMIR, On categorial and phrase-structure grammars. In: Y. BAR-HILLEL (ed.), *Language and Information*. Addison-Wesley, New York, 1960, 99–116.
- [4] N. CHOMSKY, Formal properties of grammars. In: D. LUCE, R. R. BUSH, E. GALANTER (eds.), *Handbook of Mathematical Psychology II*. John Wiley and Sons, Inc., 1963, 323–418.
- [5] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting automata. In: H. REICHEL (ed.), *International Symposium on Fundamentals of Computation Theory (FCT 1995)*. Lecture Notes in Computer Science 965, Springer, 1995, 283–292.
- [6] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On monotonic automata with a restart operation. *Journal of Automata, Languages, and Combinatorics (JALC)* 4 (1999), 287–311.
- [7] P. JANČAR, M. PLÁTEK, J. VOGEL, Generalized linear list automata. In: *Proceedings ITAT 2004*. Univerzita P. J. Šafárika v Košiciach, 2005, 97–105.
- [8] T. JURDZIŃSKI, F. OTTO, F. MRÁZ, M. PLÁTEK, Deterministic two-way restarting automata and Marcus contextual grammars. *Fundamenta Informaticae* 64 (2005), 217–228.
- [9] S. MARCUS, Contextual grammars. *Revue Roumaine de Mathématiques Pures et Appliquées* 14 (1969), 1473–1482.
- [10] H. A. MAURER, A. SALOMAA, D. WOOD, Pure Grammars. *Information and Control* 44 (1980), 47–72.
- [11] F. MRÁZ, M. PLÁTEK, M. PROCHÁZKA, On special forms of restarting automata. *Grammars* 2 (1999), 223–233.
- [12] F. OTTO, Restarting automata. In: Z. ÉSIK, C. MARTÍN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence 25, Springer, 2006, 269–303.
- [13] F. OTTO, Left-to-right regular languages and two-way restarting automata. *RAIRO – Theoretical Informatics and Applications (RAIRO: ITA)* 43 (2009), 653–665.
- [14] M. PLÁTEK, F. OTTO, On h-lexicalized restarting automata. In: E. CSUHAI-VARJÚ, P. DÖMÖSI, GY. VASZIL (eds.), *Proceedings 15th International Conference on Automata and Formal Languages (AFL 2017)*. EPTCS 252, Open Publishing Association, 2017, 219–233.
- [15] M. PLÁTEK, F. OTTO, F. MRÁZ, *On h-Lexicalized Restarting List Automata*. Technical report, Universität Kassel, 2017. [www.theory.informatik.uni-kassel.de/projekte/RL2016v6.4.pdf](http://www.theory.informatik.uni-kassel.de/projekte/RL2016v6.4.pdf).
- [16] M. PROCHÁZKA, M. PLÁTEK, Taxonomy of red-automata motivated by error localization. In: R. FREUND, M. HOLZER, B. TRUTHE, U. ULTES-NITSCHKE (eds.), *Fourth Workshop on Non-Classical Models of Automata and Applications (NCMA 2012)*. books@ocg.at 290, Österreichische Computer Gesellschaft, Wien, 2012, 181–195.
- [17] K. ČULIK II, R. COHEN, LR-regular grammars – an extension of LR( $k$ ) grammars. *Journal of Computer and System Sciences* 7 (1973), 66–96.

# NETWORKS OF EVOLUTIONARY PROCESSORS WITH RESOURCES RESTRICTED FILTERS

Bianca Truthe

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany  
bianca.truthe@informatik.uni-giessen.de

## **Abstract**

*In this paper, we continue the research on networks of evolutionary processors where the filters belong to several special families of regular languages. These subregular families are defined by restricting the resources needed for generating or accepting them (the number of states of the minimal deterministic finite automaton accepting a language of the family as well as the number of non-terminal symbols or the number of production rules of a right-linear grammar generating such a language). We insert the newly defined language families into the hierarchy of language families obtained by using as filters languages of other subregular families (such as ordered, non-counting, power-separating, circular, suffix-closed regular, union-free, definite, combinational, finite, monoidal, nilpotent, or commutative languages).*

## **1. Introduction**

Networks of language processors have been introduced in [3] by E. CSUHÁJ-VARJÚ and A. SALOMAA. Such a network can be considered as a graph where the nodes represent processors which apply production rules to the words they contain. In a derivation step (an evolutionary step), any node derives from its language all possible words as its new language. In a communication step, any node sends those words to other nodes which satisfy an output condition given as a regular language (called the output filter) and any node adopts words sent by the other nodes if the words satisfy an input condition also given by a regular language (called the input filter). The language generated by a network of language processors consists of all (terminal) words which occur in the languages associated with a given node.

Inspired by biological processes, in [1] a special type of networks of language processors was introduced. The nodes of such networks are called evolutionary processors because the allowed productions model the point mutation known from biology. The productions of a node allow that one letter is substituted by another letter, letters are inserted, or letters are deleted; the nodes are then called substitution nodes, insertion nodes, or deletion nodes, respectively. Results on networks of evolutionary processors can be found, e. g., in [1], [2], [11]. For instance, in [2], it was shown that networks of evolutionary processors are complete in that sense that

they can generate any recursively enumerable language.

In [4], the generative capacity of networks of evolutionary processors was investigated for cases that all filters belong to a certain subfamily of the set of all regular languages. It was shown that the use of filters from the family of ordered, non-counting, power-separating, circular regular, suffix-closed regular, union-free, definite, and combinational languages is as powerful as the use of arbitrary regular languages and yields networks which can generate all recursively enumerable languages. The use of filters which are only finite languages allows only the generation of regular languages, but not all regular languages can be generated. If filters are used which are monoids, nilpotent languages, or commutative regular languages, the same family of languages is obtained which contains non-context-free languages but not all regular languages.

In [5], networks were considered where the filters are accepted by deterministic finite automata with at most two states and which are all defined over the whole network alphabet.

In this paper, we continue the research on networks of evolutionary processors where the filters are restricted by further bounded resources, namely the number of non-terminal symbols or the number of production rules which are necessary for generating the languages. Additionally, we consider filters which are accepted by deterministic finite automata over an arbitrary alphabet with a bounded number of states.

## 2. Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see, e. g., [12]). and recall here only some notations used in the paper.

Let  $V$  be an alphabet. By  $V^*$  we denote the set of all words (strings) over the alphabet  $V$  (including the empty word  $\lambda$ ). For a natural number  $k$ , we denote by  $V^k$  the set of all words over the alphabet  $V$  with length  $k$ . The cardinality of a set  $A$  is denoted by  $|A|$ .

A phrase structure 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 which are written as  $\alpha \rightarrow \beta$  with  $\alpha \in (N \cup T)^* \setminus T^*$  and  $\beta \in (N \cup T)^*$ , and  $S \in N$  is the axiom. A grammar is context-free if, for any rule  $\alpha \rightarrow \beta$ , the left-hand side  $\alpha$  consists of a non-terminal symbol only:  $\alpha \in N$ . A grammar is right-linear if it is context-free and, for any rule  $\alpha \rightarrow \beta$ , the right-hand side  $\beta$  contains at most one non-terminal symbol and this is at the right end of the word:  $\beta \in T^* \cup T^*N$ . A special case of right-linearity is regularity where each rule contains exactly one terminal symbol (with the only possible exception  $S \rightarrow \lambda$ ):  $\beta \in T \cup TN$ . The language  $L(G)$  generated by a grammar  $G$  is the set of all words which consist of terminal symbols and which are derivable from the axiom  $S$  by a successive substitution of the non-terminal symbols according to the rules of the grammar.

Regular and right-linear grammars generate the same family of languages (the regular languages). Therefore, also right-linear grammars are often called regular. In the context of descriptive complexity, when the number of non-terminal symbols or the number of produc-

tion rules which are necessary for generating a language are considered then there is a difference whether a language is generated by means of regular or right-linear rules. We use in this paper right-linear grammars.

By *REG*, *CF*, and *RE*, we denote the families of languages generated by regular, context-free, and arbitrary phrase structure grammars, respectively.

A finite automaton is a quintuple  $\mathcal{A} = (V, Z, z_0, F, \delta)$  where  $V$  is an alphabet called the input alphabet,  $Z$  is a non-empty finite set of elements which are called states,  $z_0 \in Z$  is the so-called start state,  $F \subseteq Z$  is the set of accepting states, and  $\delta : Z \times V \rightarrow \mathcal{P}(Z)$  is a mapping which is also called the transition function where  $\mathcal{P}(Z)$  denotes the power set of  $Z$  (the set of all subsets of  $Z$ ). A finite automaton is called deterministic if every set  $\delta(z, a)$  for  $z \in Z$  and  $a \in V$  is a singleton set.

The transition function  $\delta$  can be extended to a function  $\delta^* : Z \times V^* \rightarrow \mathcal{P}(Z)$  where  $\delta^*(z, \lambda) = \{z\}$  and

$$\delta^*(z, va) = \bigcup_{z' \in \delta^*(z, v)} \delta(z', a).$$

We will use the same symbol  $\delta$  in both the original and extended version of the transition function.

Let  $\mathcal{A} = (V, Z, z_0, F, \delta)$  be a finite automaton. A word  $w$  is accepted by the finite automaton  $\mathcal{A}$  if and only if the automaton has reached an accepting state after reading the input word  $w$ :

$$\delta(z_0, w) \cap F \neq \emptyset.$$

The family of the language accepted by finite automata is also the family of the regular language.

For a language  $L$  over  $V$ , we set

$$\begin{aligned} Comm(L) &= \{a_{i_1} \dots a_{i_n} \mid a_1 \dots a_n \in L, n \geq 1, \{i_1, i_2, \dots, i_n\} = \{1, 2, \dots, n\}\}, \\ Circ(L) &= \{vu \mid uv \in L, u, v \in V^*\}, \\ Suf(L) &= \{v \mid uv \in L, u, v \in V^*\}. \end{aligned}$$

In [4], the following restrictions for regular languages are considered. In order to relate our results of this paper to the results there, we explain here those special regular languages. Let  $L$  be a language and  $V = alph(L)$  the minimal alphabet of  $L$ . We say that the language  $L$ , with respect to the alphabet  $V$ , is

- *monoidal* if  $L = V^*$ ,
- *combinational* if it has the form  $L = V^*A$  for some subset  $A \subseteq V$ ,
- *definite* if it can be represented in the form  $L = A \cup V^*B$  where  $A$  and  $B$  are finite subsets of  $V^*$ ,
- *nilpotent* if  $L$  is finite or  $V^* \setminus L$  is finite,
- *commutative* if  $L = Comm(L)$ ,

- *circular* if  $L = \text{Circ}(L)$ ,
- *suffix-closed* if the relation  $xy \in L$  for some words  $x, y \in V^*$  implies that also the suffix  $y$  belongs to  $L$  or equivalently,  $L = \text{Suf}(L)$ ,
- *non-counting* (or star-free) if there is an integer  $k \geq 1$  such that, for any  $x, y, z \in V^*$ , the relation  $xy^kz \in L$  holds if and only if also  $xy^{k+1}z \in L$  holds,
- *power-separating* if for any word  $x \in V^*$  there is a natural number  $m \geq 1$  such that either the equality  $J_x^m \cap L = \emptyset$  or the inclusion  $J_x^m \subseteq L$  holds where  $J_x^m = \{x^n \mid n \geq m\}$ ,
- *ordered* if  $L$  is accepted by some finite automaton  $\mathcal{A} = (Z, V, \delta, z_0, F)$  where  $(Z, \preceq)$  is a totally ordered set and, for any  $a \in V$ ,  $z \preceq z'$  implies  $\delta(z, a) \preceq \delta(z', a)$ ,
- *union-free* if  $L$  can be described by a regular expression which is only built by product and star.

Among the commutative, circular, suffix-closed, non-counting, and power-separating languages, we consider only those which are also regular.

By *FIN*, *MON*, *COMB*, *DEF*, *NIL*, *COMM*, *CIRC*, *SUF*, *NC*, *PS*, *ORD*, and *UF*, we denote the families of all finite, monoidal, combinational, definite, nilpotent, regular commutative, regular circular, regular suffix-closed, regular non-counting, regular power-separating, ordered, and union-free languages, respectively.

In this paper, families of languages are of special interest which are defined by bounding the resources which are necessary for accepting or generating these languages.

Let *RLG* be the set of all right-linear grammars and *DFA* the set of all deterministic finite automata. Further, let  $G = (N, T, P, S) \in \text{RLG}$  and  $\mathcal{A} = (V, Z, z_0, F, \delta) \in \text{DFA}$ . Then we define the following measures of descriptional complexity:

$$\text{Var}(G) = |N|, \quad \text{Prod}(G) = |P|, \quad \text{State}(\mathcal{A}) = |Z|.$$

For these complexity measures, we define the following families of languages (we abbreviate the measure *Var* by *V*, the measure *Prod* by *P*, and the measure *State* by *Z*):

$$\begin{aligned} \text{RL}_n^V &= \{ L \mid \exists G \in \text{RLG} : L = L(G) \text{ and } \text{Var}(G) \leq n \}, \\ \text{RL}_n^P &= \{ L \mid \exists G \in \text{RLG} : L = L(G) \text{ and } \text{Prod}(G) \leq n \}, \\ \text{REG}_n^Z &= \{ L \mid \exists \mathcal{A} \in \text{DFA} : L = L(\mathcal{A}) \text{ and } \text{State}(\mathcal{A}) \leq n \}. \end{aligned}$$

The relations between the considered families are investigated, e. g., in [7], [8], [13], [14], [15], and [16]. They are illustrated in Figure 1.

An edge label in Figure 1 refers to the paper where the respective inclusion is proved. The proper inclusions where no reference is given in the figure as well as the incomparabilities are proved in [15].

Regarding the families defined by bounded resources, we note the following relations:  $K_i \subset K_{i+1}$  for  $K \in \{\text{RL}^V, \text{RL}^P, \text{REG}^Z\}$  and  $i \geq 1$  as well as  $\text{RL}_{2i}^P \subset \text{RL}_i^V$  and  $\text{REG}_i^Z \subset \text{RL}_i^V$ .



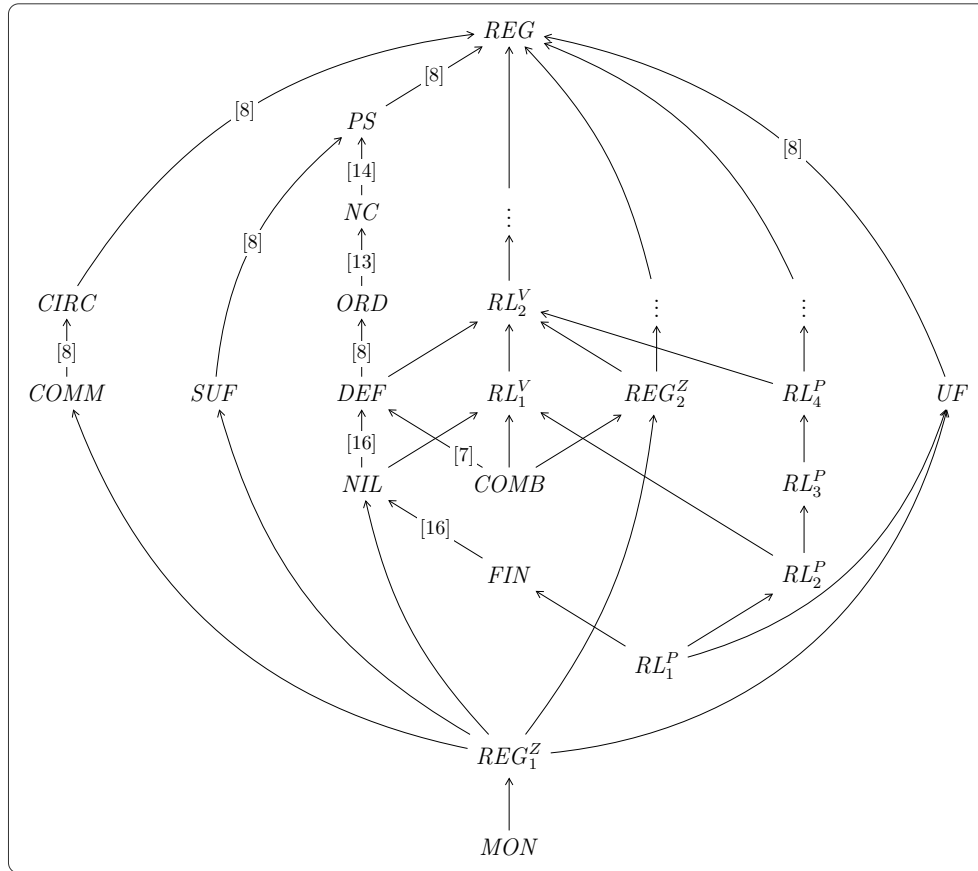


Figure 1: Hierarchy of subregular language families

**Theorem 2.1** *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.*

We call a production  $\alpha \rightarrow \beta$  a

- substitution if  $|\alpha| = |\beta| = 1$ ,
- deletion if  $|\alpha| = 1$  and  $\beta = \lambda$ .

The productions are applied like context-free rewriting rules. We say that a word  $v$  derives a word  $w$ , written as  $v \implies w$ , if there are words  $x, y$  and a production  $\alpha \rightarrow \beta$  such that  $v = x\alpha y$  and  $w = x\beta y$ . In order to indicate also the applied rule  $p$ , we write  $v \implies_p w$ .

We introduce insertion as a counterpart of deletion. We write  $\lambda \rightarrow a$ , where  $a$  is a letter. The application of an insertion  $\lambda \rightarrow a$  derives from a word  $w$  any word  $w_1aw_2$  with  $w = w_1w_2$  for some (possibly empty) words  $w_1$  and  $w_2$ .

We now present the basic concept of this paper, the networks of evolutionary processors (NEPs for short).

**Definition 2.2** Let  $X$  be a family of regular languages.

(i) A network of evolutionary processors with filters from the family  $X$  is a tuple

$$\mathcal{N} = (V, N_1, N_2, \dots, N_n, E, j)$$

where

- $V$  is a finite alphabet,
- $N_i = (M_i, A_i, I_i, O_i)$  for  $1 \leq i \leq n$  are the processors where
  - $M_i$  is a set of rules of a certain type:  $M_i \subseteq \{a \rightarrow b \mid a, b \in V\}$  or  $M_i \subseteq \{a \rightarrow \lambda \mid a \in V\}$  or  $M_i \subseteq \{\lambda \rightarrow b \mid b \in V\}$ ,
  - $A_i$  is a finite subset of  $V^*$ ,
  - $I_i$  and  $O_i$  are languages from  $X$  over some subset of the alphabet  $V$ ,
- $E$  is a subset of  $\{1, 2, \dots, n\} \times \{1, 2, \dots, n\}$ , and
- $j$  is a natural number such that  $1 \leq j \leq n$ .

(ii) A configuration  $C$  of  $\mathcal{N}$  is an  $n$ -tuple  $C = (C(1), C(2), \dots, C(n))$  where  $C(i)$  is a subset of  $V^*$  for  $1 \leq i \leq n$ .

(iii) Let  $C = (C(1), C(2), \dots, C(n))$  and  $C' = (C'(1), C'(2), \dots, C'(n))$  be two configurations of  $\mathcal{N}$ . We say that  $C$  derives  $C'$  in one

- evolutionary step (written as  $C \Longrightarrow C'$ ) if, for  $1 \leq i \leq n$ ,  $C'(i)$  consists of all words  $w \in C(i)$  to which no rule of  $M_i$  is applicable (no left-hand side of a rule is present in the word  $w$ ; since the empty word is always a subword, insertion rules can always be applied) and of all words  $w$  for which there are a word  $v \in C(i)$  and a rule  $p \in M_i$  such that  $v \Longrightarrow_p w$  holds,
- communication step (written as  $C \vdash C'$ ) if, for  $1 \leq i \leq n$ ,

$$C'(i) = (C(i) \setminus O_i) \cup \bigcup_{(k,i) \in E} (C(k) \cap O_k \cap I_i).$$

The computation of an evolutionary network  $\mathcal{N}$  is a sequence of configurations

$$C_t = (C_t(1), C_t(2), \dots, C_t(n)), \quad t \geq 0,$$

such that

- $C_0 = (A_1, A_2, \dots, A_n)$ ,
- for any  $t \geq 0$ ,  $C_{2t}$  derives  $C_{2t+1}$  in one evolutionary step,
- for any  $t \geq 0$ ,  $C_{2t+1}$  derives  $C_{2t+2}$  in one communication step.

(iv) The language  $L(\mathcal{N})$  generated by  $\mathcal{N}$  is defined as

$$L(\mathcal{N}) = \bigcup_{t \geq 0} C_t(j)$$

where  $C_t = (C_t(1), C_t(2), \dots, C_t(n))$ ,  $t \geq 0$  is the computation of  $\mathcal{N}$ .

Intuitively, a network with  $n$  evolutionary processors is a graph consisting of nodes  $N_1, \dots, N_n$  (also called processors) and a set of edges given by  $E$  such that there is a directed edge from node  $N_k$  to node  $N_i$  if and only if  $(k, i) \in E$ .

Any processor  $N_i$  consists of a set  $M_i$  of evolutionary rules, a set  $A_i$  of words, an input filter  $I_i$ , and an output filter  $O_i$ . We say that  $N_i$  is

- a substitution node if  $M_i \subseteq \{a \rightarrow b \mid a, b \in V\}$  (by any rule, a letter is substituted by another one),
- a deletion node if  $M_i \subseteq \{a \rightarrow \lambda \mid a \in V\}$  (by any rule, a letter is deleted), or
- an insertion node if  $M_i \subseteq \{\lambda \rightarrow b \mid b \in V\}$  (by any rule, a letter is inserted).

Every node has rules from one type only. The input filter  $I_i$  and the output filter  $O_i$  control the words which are allowed to enter and to leave the node, respectively. With any node  $N_i$  and any time moment  $t \geq 0$ , we associate a set  $C_t(i)$  of words (the words contained in the node at time  $t$ ). Initially,  $N_i$  contains the words of  $A_i$ . In an evolutionary step, we derive from  $C_t(i)$  all words by applying rules from the set  $M_i$ . In a communication step, any processor  $N_i$  sends out all words from the set  $C_t(i) \cap O_i$  (which pass the output filter) to all processors to which a directed edge exists (only the words from  $C_t(i) \setminus O_i$  remain in the set associated with  $N_i$ ) and, moreover, it receives from any processor  $N_k$  such that there is an edge from  $N_k$  to  $N_i$  all words sent by  $N_k$  and passing the input filter  $I_i$  of  $N_i$ , i. e., the processor  $N_i$  gets in addition all words of  $C_t(k) \cap O_k \cap I_i$ . We start with an evolutionary step and then communication steps and evolutionary steps are alternately performed. The language consists of all words which are in the node  $N_j$  (also called the output node,  $j$  is chosen in advance) at some moment  $t$ ,  $t \geq 0$ .

If two networks of evolutionary processors generate the same language, we say that the networks are equivalent to each other.

For a family  $X$ , we denote the family of languages generated by networks of evolutionary processors where all filters are of type  $X$  by  $\mathcal{E}(X)$ . We consider the filters independently from the environment. A filter language belongs to some family  $X$  if it belongs to it with respect to its smallest alphabet, not necessarily to the the alphabet of all letters which might occur in the node or even in the entire network. A word passes a filter if it is an element of the language representing the filter otherwise it does not pass the filter.

The following theorem is known (see, e. g., [2]).

**Theorem 2.3**  $\mathcal{E}(REG) = RE$ .

As an extension to Lemma 1 from [4], we have the following result.

**Lemma 2.4** *Let  $X$  and  $Y$  be two families of languages such that  $X \subseteq Y$ . Then the inclusion*

$$\mathcal{E}(X) \subseteq \mathcal{E}(Y)$$

*holds.*

*Proof.* Let  $X$  and  $Y$  be two language families with  $X \subseteq Y$ . Further, let  $L$  be a language generated by a network  $\mathcal{N}$  with all filters from the family  $X$ . Then the network  $\mathcal{N}$  has also all filters from the family  $Y$ . Hence,  $L \in \mathcal{E}(Y)$ , which yields the inclusion.  $\square$

### 3. Results

We first consider the number of non-terminal symbols sufficient for generating a language. We obtain that any recursively enumerable language is generated by a network of evolutionary processors where each filter is generated by a right-linear grammar with only one non-terminal symbol.

**Theorem 3.1** *We have  $\mathcal{E}(RL_i^V) = RE$  for all natural numbers  $i \geq 1$ .*

*Proof.* Every combinational language can be generated by a regular grammar with only one non-terminal symbol: For generating a language  $V^*A$ , the rules  $S \rightarrow vS$  for every letter  $v \in V$  and  $S \rightarrow a$  for every letter  $a \in A$  are sufficient ([15]). According to Lemma 2.4, the chain of inclusions

$$COMB \subseteq RL_1^V \subseteq RL_2^V \subseteq \dots \subseteq REG$$

implies the chain of inclusions

$$\mathcal{E}(COMB) \subseteq \mathcal{E}(RL_1^V) \subseteq \mathcal{E}(RL_2^V) \subseteq \dots \subseteq \mathcal{E}(REG).$$

In [4], the relation  $\mathcal{E}(COMB) = RE$  was proved. Hence, together with Theorem 2.3, we also have  $\mathcal{E}(RL_i^V) = RE$  for all natural numbers  $i \geq 1$ .  $\square$

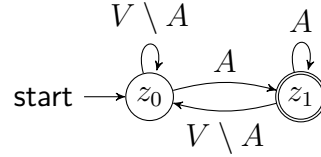
We now consider networks where the filters are accepted by deterministic finite automata with a bounded number of states. In [4] and [5], such networks have been considered where the filters are even more restricted, namely that every filter of a network over an alphabet  $V$  is accepted by a complete deterministic finite automaton whose input alphabet is also  $V$ . Here, we consider the filters independently of each other. Therefore, the families investigated previously are special cases of the families studied in the present paper. The generative capacity of networks where the filters are arbitrary languages from a family  $REG_i^Z$  with  $i \geq 1$  is not smaller than the generative capacity of networks where the filters are languages from the family  $REG_i^Z$  but such that all deterministic finite automata representing the filters of a network have the same input alphabet.

For networks with filters which are accepted by deterministic finite automata with two states all defined over the same alphabet, the computational completeness was already shown in [5]. If we generalize the filters to languages over an arbitrary alphabet, the generative capacity cannot be smaller. Therefore, we obtain the same result here which can also be proved analogously to Theorem 3.1.

**Theorem 3.2** *We have  $\mathcal{E}(REG_i^Z) = RE$  for all natural numbers  $i \geq 2$ .*

*Proof.* Every combinational language can be accepted by a deterministic finite automaton with two states only:

A language  $V^*A$  with an alphabet  $V$  and a subset  $A \subseteq V$  is accepted by an automaton whose transition graph is as follows ([15]):



According to Lemma 2.4, the chain of inclusions

$$COMB \subseteq REG_2^Z \subseteq REG_3^Z \subseteq \dots \subseteq REG$$

implies the chain of inclusions

$$\mathcal{E}(COMB) \subseteq \mathcal{E}(REG_2^Z) \subseteq \mathcal{E}(REG_3^Z) \subseteq \dots \subseteq \mathcal{E}(REG).$$

Since  $\mathcal{E}(COMB) = RE = \mathcal{E}(REG)$  ([4] and Theorem 2.3), also the relation  $\mathcal{E}(REG_i^Z) = RE$  holds for every natural number  $i \geq 2$ .  $\square$

For networks with filters which are accepted by deterministic finite automata with one state only, the situation is different. If all automata are defined over the same alphabet, then a proper subset of the family  $\mathcal{E}(MON)$  is obtained ([4]). If arbitrary alphabets are allowed, the two families coincide, as is stated in the next theorem.

**Theorem 3.3** *We have  $\mathcal{E}(REG_1^Z) = \mathcal{E}(MON)$ .*

*Proof.* The inclusion  $MON \subset REG_1^Z$  ([15]) implies also the inclusion  $\mathcal{E}(MON) \subseteq \mathcal{E}(REG_1^Z)$  (Lemma 2.4). We now prove that the inverse inclusion  $\mathcal{E}(REG_1^Z) \subseteq \mathcal{E}(MON)$  holds as well.

Every language of the family  $REG_1^Z$  is the empty set or a monoidal language. A network with filters in  $REG_1^Z$  which contains empty sets as filters can be transformed into a network which has only monoidal languages as filters and which generates the same language ([4]), which can be seen as follows.

If the input filter of a node  $N$  is the empty set, then no word can enter this node. Thus, we can remove all edges that lead to the node  $N$  and set the input filter to an arbitrary monoidal language without changing the language generated. If the output filter of a node is the empty set, then no word can leave this node. If this is the case for a node  $N$  which is not the output node, then this node  $N$  is useless in that sense that it does not contribute to the language generated. So, we can eliminate this node together with all incident edges without changing the language. If the output node has an empty output filter, then we replace this filter by the language  $V^*$  where  $V$  is the alphabet of the network, delete all outgoing edges, and add a new edge to this node itself (or to a new node  $N' = (\emptyset, \emptyset, V^*, V^*)$  and back if loops should be avoided). This new network generates the same language and has only monoidal filters. Hence, also the inclusion  $\mathcal{E}(REG_1^Z) \subseteq \mathcal{E}(MON)$  holds, and, thus, also the equality  $\mathcal{E}(REG_1^Z) = \mathcal{E}(MON)$ .  $\square$

We finally investigate networks where the filters are restricted by the number of production rules necessary for generating the filters. Here, we obtain an infinite hierarchy. Further inclusion relations and incomparability results are proved in the sequel.

**Theorem 3.4** *The proper inclusions  $FIN \subset \mathcal{E}(RL_1^P)$  and  $MON \subset \mathcal{E}(RL_1^P)$  hold.*

*Proof.* Let  $L$  be a finite language. This language is generated by the NEP with the unique node

$$N_f = (\emptyset, L, \emptyset, \emptyset).$$

Hence,  $FIN \subseteq \mathcal{E}(RL_1^P)$ . Let  $V$  be an alphabet. The language  $V^*$  is generated by the NEP with the unique node

$$N_m = (\{ \lambda \rightarrow a \mid a \in V \}, \{ \lambda \}, \emptyset, \emptyset).$$

Hence,  $MON \subseteq \mathcal{E}(RL_1^P)$ . Since the family  $\mathcal{E}(RL_1^P)$  contains finite and monoidal languages and  $FIN \cap MON = \emptyset$ , each inclusion is proper.  $\square$

**Theorem 3.5** *The proper inclusion  $\mathcal{E}(RL_1^P) \subset \mathcal{E}(FIN)$  holds.*

*Proof.* According to Theorem 2.1 and Lemma 2.4, we obtain the inclusion  $\mathcal{E}(RL_1^P) \subseteq \mathcal{E}(FIN)$ . A witness language for the properness is

$$L = \{a^2, a^3, a^5, a^6\} \cup \{a^n \mid n \geq 8\}.$$

In [6], it was shown that the language  $L$  belongs to the family  $\mathcal{E}(FIN)$ .

Assume that it also belongs to the family  $\mathcal{E}(RL_1^P)$ . A filter which is generated by a right-linear grammar with one rule contains at most one word. If a node has an empty input filter, then any edge leading to this node can be removed and this input filter can be replaced by a singleton set without changing the generated language. Nodes which are not the output node and which have an empty output filter do not contribute to the generated language and can therefore be removed from the network without changing the generated language. Since the language  $L$  is infinite, the output node contains the rule  $\lambda \rightarrow a$ . If the output node has an empty output filter, then also the word  $a^4 \notin L$  is generated by the network which is a contradiction. Otherwise, the output filter is a singleton set (as well as all other filters by our construction) and therefore a code ([12]). But in [6], it was also shown that the language  $L$  cannot be generated by a network with codes as filters. Thus,  $L \notin \mathcal{E}(RL_1^P)$ .  $\square$

The idea from the previous proof will be generalized for proving further results.

**Lemma 3.6** *For any natural number  $i \geq 1$ , let  $V_i$  be an alphabet with  $i$  pairwise different letters:*

$$V_i = \{a_1, a_2, \dots, a_i\}.$$

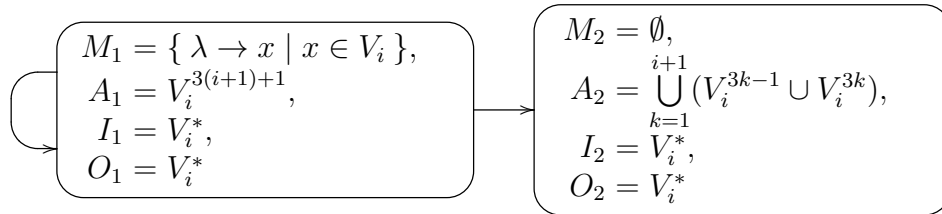
*Further, let*

$$L_{i+1} = \bigcup_{k=1}^{i+1} (V_i^{3k-1} \cup V_i^{3k}) \cup \bigcup_{n \geq 3(i+1)+2} V_i^n.$$

*Then*

$$L_{i+1} \in (\mathcal{E}(RL_{i+1}^P) \cap \mathcal{E}(FIN) \cap NIL \cap COMM) \setminus \mathcal{E}(RL_i^P).$$

*Proof.* Let  $i \geq 1$  be a natural number. The language  $L_{i+1}$  is generated by a network with two nodes where each filter belongs to the family  $\mathcal{E}(RL_{i+1}^P)$ :



The second node is the output node which provides the ‘finite’ part of the language  $L_{i+1}$  itself. The first node provides the ‘infinite’ part of the language  $L_{i+1}$ . The language  $V_i^*$  can be generated by the  $i + 1$  regular rules  $S \rightarrow x$  for  $x \in V_i$  and  $S \rightarrow \lambda$ . Hence,  $L_{i+1} \in \mathcal{E}(RL_{i+1}^P)$ .

The language  $L_{i+1}$  is also generated by a network with the unique node

$$N = (\{ \lambda \rightarrow x \mid x \in V_i \}, \bigcup_{k=1}^{i+2} V_i^{3k-1}, \emptyset, \bigcup_{k=1}^{i+1} V_i^{3k}).$$

From the part  $\bigcup_{k=1}^{i+1} V_i^{3k-1}$ , the part  $\bigcup_{k=1}^{i+1} V_i^{3k}$  is generated which then leaves the network and cannot be further derived. From the part  $V_i^{3(i+2)-1} = V_i^{3(i+1)+2}$ , the words of the set  $V_i^n$  with  $n \geq 3(i + 1) + 2$  will successively be generated because no word from these can leave the network. Hence,  $L_{i+1} \in \mathcal{E}(FIN)$ .

The complement of the language  $L_{i+1}$  is

$$\bar{L}_{i+1} = V_i^* \setminus L_{i+1} = \{ \lambda \} \cup \bigcup_{0 \leq k \leq i+1} V_i^{3k+1}$$

which is a finite language. Therefore, the language  $L_{i+1}$  is nilpotent.

Since every set  $V_i^k$  for  $k \geq 0$  is commutative and union preserves commutativity, the language  $L_{i+1}$  is also commutative.

Assume that  $L_{i+1} \in \mathcal{E}(RL_i^P)$ . Since the language  $L_{i+1}$  is infinite and every letter of  $V_i$  occurs in an arbitrary number, the network contains the rules  $\lambda \rightarrow x$  for every letter  $x \in V_i$ . If the output node has no rules then its input filter must cover infinitely many words of the language  $L_{i+1}$  (all with a finite number of exceptions which can be present in the output node in the beginning). Also in this language (the input filter), every letter of  $V_i$  occurs in an unbounded number. Therefore, any right-linear grammar generating the filter needs, for any letter  $x \in V_i$ , a rule which produces the letter  $x$  independently from all other letters and in an unbounded number. Finally, also a terminating rule is necessary. Hence,  $i$  rules are not sufficient for generating the filter. If the output node has rules, then its output filter must contain all words of the set

$$\bigcup_{k=1}^{i+1} V_i^{3k}$$

because these words are present in the node at some time (they belong to the generated language  $L_{i+1}$ ) but, from these, no word may be derived, otherwise words (of length  $3k + 1$  with  $1 \leq k \leq i + 1$ ) would be generated which do not belong to the language  $L_{i+1}$ . But also such a filter cannot be achieved if only  $i$  rules are allowed for generating the filter. Therefore, we obtain  $L_{i+1} \notin \mathcal{E}(RL_i^P)$ .  $\square$

The previous lemma is now used to prove an infinite hierarchy.

**Theorem 3.7** *For all natural numbers  $i \geq 1$ , the proper inclusion  $\mathcal{E}(RL_i^P) \subset \mathcal{E}(RL_{i+1}^P)$  holds.*

*Proof.* Let  $i \geq 1$  be a natural number. The inclusion  $\mathcal{E}(RL_i^P) \subseteq \mathcal{E}(RL_{i+1}^P)$  follows from Theorem 2.1 and Lemma 2.4. A witness language for the properness is the language  $L_{i+1}$  as was shown in Lemma 3.6.  $\square$

We now prove some incomparability results.

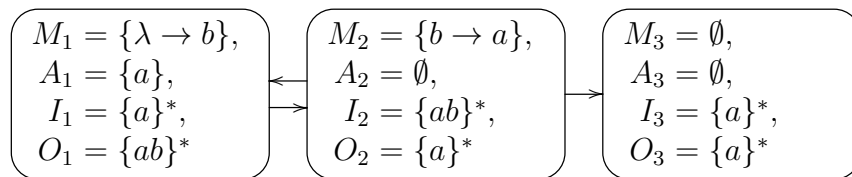
**Theorem 3.8** *Each of the families  $\mathcal{E}(FIN)$ ,  $REG$ , and  $CF$  is incomparable to every family  $\mathcal{E}(RL_i^P)$  for  $i \geq 2$ .*

*Proof.* Due to the inclusion relations  $\mathcal{E}(FIN) \subset REG \subset CF$  and  $\mathcal{E}(RL_i^P) \subset \mathcal{E}(RL_{i+1}^P)$ , it suffices to show that, for all  $i \geq 2$ , there is a language which belongs to the set  $\mathcal{E}(FIN) \setminus \mathcal{E}(RL_i^P)$  and that there is a language in the family  $\mathcal{E}(RL_2^P)$  which is not context-free. From Lemma 3.6, we know that  $L_{i+1} \in \mathcal{E}(FIN) \setminus \mathcal{E}(RL_i^P)$  for  $i \geq 1$ .

A witness language for the other case is

$$L = \{ a^{2^n} \mid n \geq 1 \}.$$

This language is not context-free and we now prove that it belongs to the family  $\mathcal{E}(RL_2^P)$ . The language  $L$  is accepted by the following network where the output node is  $N_3 = (M_3, A_3, I_3, O_3)$ :



In the first node, the number of letters is doubled: starting from a string  $a^{2^n}$  with  $n \geq 0$ , the string  $(ab)^{2^n}$  is produced (words with less than  $2^n$  letters  $b$  cannot leave the node yet; words with  $2^n$  letters  $b$  but not the correct form and words with more than  $2^n$  letters  $b$  can not be derived to a string anymore which could leave the node). The second node receives a word  $(ab)^{2^n}$  with  $n \geq 0$  and substitutes all occurrences of  $b$  by  $a$ , obtaining the word  $a^{2^{n+1}}$  with  $n \geq 0$ , which is then sent to the third node (the output node) and to the first node (where its length is again doubled before it can leave the node again).

The network contains the filters  $\{a\}^*$  and  $\{ab\}^*$ . These sets are generated by the right-linear grammars  $G_w = (\{S\}, \{a, b\}, \{S \rightarrow w, S \rightarrow \lambda\}, S)$  with  $w \in \{a, ab\}$ . Thus,  $L \in \mathcal{E}(RL_2^P)$ .  $\square$

**Theorem 3.9** *The family  $NIL$  is incomparable to every family  $\mathcal{E}(RL_i^P)$  for  $i \geq 1$ .*



*Proof.* Due to the inclusion relations  $\mathcal{E}(RL_i^P) \subset \mathcal{E}(RL_{i+1}^P)$  for  $i \geq 1$ , it suffices to show that, for all  $i \geq 1$ , there is a language which belongs to the set  $NIL \setminus \mathcal{E}(RL_i^P)$  and that there is a language in the family  $\mathcal{E}(RL_1^P)$  which is not nilpotent. From Lemma 3.6, we know that  $L_{i+1} \in NIL \setminus \mathcal{E}(RL_i^P)$  for  $i \geq 1$ .

A witness language for the other case is

$$L = \{c^k ac^m bc^n \mid k \geq 1, m \geq 1, n \geq 1\}.$$

This language is not nilpotent (both the languages  $L$  and  $\{a, b, c\}^* \setminus L$  are infinite). However, it is generated by the network with the unique node  $N = (\{\lambda \rightarrow c\}, \{ab\}, \emptyset, \emptyset)$  and, therefore, it belongs to the family  $\mathcal{E}(RL_1^P)$ .  $\square$

**Theorem 3.10** *The family COMM is incomparable to every family  $\mathcal{E}(RL_i^P)$  for  $i \geq 1$ .*

*Proof.* Due to the inclusion relations  $\mathcal{E}(RL_i^P) \subset \mathcal{E}(RL_{i+1}^P)$  for  $i \geq 1$ , it suffices to show that, for all  $i \geq 1$ , there is a language which belongs to the set  $COMM \setminus \mathcal{E}(RL_i^P)$  and that there is a language in the family  $\mathcal{E}(RL_1^P)$  which is not commutative. From Lemma 3.6, we know that  $L_{i+1} \in COMM \setminus \mathcal{E}(RL_i^P)$  for  $i \geq 1$ .

A witness language for the other case is

$$L = \{ab\}.$$

This language is not commutative but finite and therefore, according to Theorem 3.4, it belongs to the family  $\mathcal{E}(RL_1^P)$ .  $\square$

**Theorem 3.11** *The family  $\mathcal{E}(MON)$ ,  $\mathcal{E}(NIL)$ ,  $\mathcal{E}(COMM)$ , and  $\mathcal{E}(REG_1^Z)$  are incomparable to every family  $\mathcal{E}(RL_i^P)$  for  $i \geq 2$ .*

*Proof.* Due to the equalities

$$\mathcal{E}(MON) = \mathcal{E}(NIL) = \mathcal{E}(COMM)$$

proved in [4] and

$$\mathcal{E}(MON) = \mathcal{E}(REG_1^Z)$$

proved in Theorem 3.3 as well as the inclusion relations

$$\mathcal{E}(RL_i^P) \subset \mathcal{E}(RL_{i+1}^P)$$

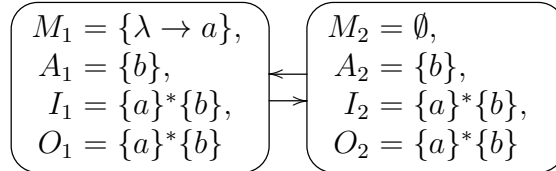
for  $i \geq 1$ , it suffices to show that, for all  $i \geq 2$ , there is a language which belongs to the set  $\mathcal{E}(MON) \setminus \mathcal{E}(RL_i^P)$  and that there is a language in the family  $\mathcal{E}(RL_2^P)$  which does not belong to the family  $\mathcal{E}(MON)$ . From Lemma 3.6, we know that  $L_{i+1} \in \mathcal{E}(FIN) \setminus \mathcal{E}(RL_i^P)$  for  $i \geq 1$ . Since  $\mathcal{E}(FIN) \subset \mathcal{E}(MON)$  (see [4]), we have, for every  $i \geq 1$ , also

$$L_{i+1} \in \mathcal{E}(MON) \setminus \mathcal{E}(RL_i^P).$$

A witness language for the other case is

$$L = \{a\}^* \{b\}.$$

This language belongs to the family  $RL_2^P$  (it is generated by a regular grammar with an axiom  $S$  and the two rules  $S \rightarrow aS$  and  $S \rightarrow b$ ) and also to the family  $\mathcal{E}(RL_2^P)$  since it is generated by the following network where the node  $N_2 = (M_2, A_2, I_2, O_2)$  is the output node:



Assume that the language  $L$  can also be generated by a network with monoidal filters. Since the number of the letter  $a$  in the words of this language is unbounded, there is the rule  $\lambda \rightarrow a$  in some node of the network. In an evolutionary step, it cannot be avoided that a letter  $a$  is inserted after the letter  $b$  or that the letter  $b$  is inserted before the last letter  $a$ . By means of monoidal filters, it cannot be avoided that such words reach the output node (if a word  $a^n b$  can enter the output node then also the word  $ba^n$ ). Hence, also other words than those of the language  $L$  would be generated which is a contradiction.  $\square$

## 4. Conclusions

We have investigated networks of evolutionary processors where the filters belong to subregular language families which are defined by restricting the resources needed for generating or accepting them (the number of states of the minimal deterministic finite automaton accepting a language of the family, the number of non-terminal symbols, or the number of production rules of a right-linear grammar generating such a language). We have inserted the newly defined language families into the hierarchy of language families obtained by using languages of other subregular families as filters (such as ordered, non-counting, power-separating, circular, suffix-closed regular, union-free, definite, combinational, finite, monoidal, nilpotent, or commutative languages) which was published in [4]. The hierarchy with the new results is shown in Figure 2.

**Theorem 4.1** *The relations shown in Figure 2 hold.*

In [6], networks of evolutionary processors were investigated where the filters are ideals or special codes. It remains for future research to compare those language families obtained there to the ones obtained here.

In [10], an accepting variant of networks of evolutionary processors was introduced where one node contains a word (the input word) and all other nodes are empty in the beginning and the input word is accepted if at some moment some word arrives in a distinguished node. It was shown that accepting networks with regular filters can accept all recursively enumerable languages. The power of some subregular filters was studied in [9]. The results differ from those obtained for generating networks. Therefore, also for accepting networks, the effect of using filters from subregular language families defined by bounded resources should be investigated.

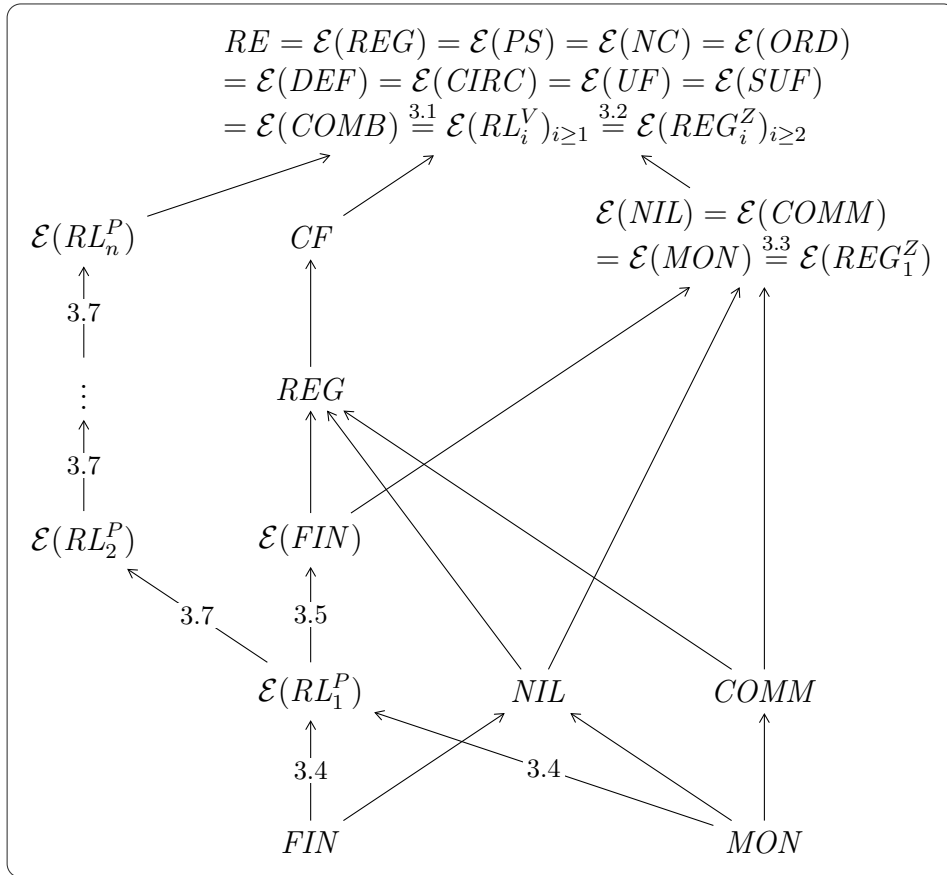


Figure 2: Hierarchy of language families by NEPs with filters from subregular families. An arrow from a language family  $X$  to a language family  $Y$  stands for the proper inclusion  $X \subset Y$ . If two families  $X$  and  $Y$  are not connected by a directed path, then the families are incomparable. The labels at the arrows or equality signs refer to the theorems where the respective relation is shown.

## References

- [1] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, J. M. SEMPERE, Solving NP-complete problems with networks of evolutionary processors. In: *IWANN '01: Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks*. Lecture Notes in Computer Science 2084, Springer, 2001, 621–628.
- [2] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, J. M. SEMPERE, Networks of evolutionary processors. *Acta Informatica* 39 (2003) 6–7, 517–529.
- [3] E. CSUHAI-VARJÚ, A. SALOMAA, Networks of parallel language processors. In: *New Trends in Formal Languages – Control, Cooperation, and Combinatorics*. Lecture Notes in Computer Science 1218, Springer, 1997, 299–318.
- [4] J. DASSOW, F. MANEA, B. TRUTHE, Networks of evolutionary processors: the power of subregular filters. *Acta Informatica* 50 (2013) 1, 41–75.
- [5] J. DASSOW, B. TRUTHE, On networks of evolutionary processors with filters accepted by two-state-automata. *Fundamenta Informaticae* 112 (2011) 2–3, 157–170.

- [6] J. DASSOW, B. TRUTHE, Networks with evolutionary processors and ideals and codes as Filters. *International Journal of Foundations of Computer Science* (2018). Accepted.
- [7] I. M. HAVEL, The theory of regular events II. *Kybernetika* 5 (1969) 6, 520–544.
- [8] M. HOLZER, B. TRUTHE, On relations between some subregular language families. In: R. FREUND, M. HOLZER, N. MOREIRA, R. REIS (eds.), *Seventh Workshop on Non-Classical Models of Automata and Applications (NCMA 2015)*. books@ocg.at 318, Österreichische Computer Gesellschaft, Wien, 2015, 109–124.
- [9] F. MANEA, B. TRUTHE, Accepting networks of evolutionary processors with subregular filters. *Theory of Computing Systems* 55 (2014) 1, 84–109.
- [10] M. MARGENSTERN, V. MITRANA, M.-J. PERÉZ-JIMÉNEZ, Accepting hybrid networks of evolutionary processors. In: C. FERRETTI, G. MAURI, C. ZANDRON (eds.), *DNA Computing – 10th International Workshop on DNA Computing*. Lecture Notes in Computer Science 3384, Springer, 2004, 235–246.
- [11] C. MARTÍN-VIDE, V. MITRANA, Networks of evolutionary processors: results and perspectives. In: M. GHEORGHE (ed.), *Molecular Computational Models: Unconventional Approaches*. Idea Group Publishing, 2005, 78–114.
- [12] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. Springer, 1997.
- [13] H.-J. SHYR, G. THIERRIN, Ordered automata and associated languages. *Tankang Journal of Mathematics* 5 (1974) 1, 9–20.
- [14] H.-J. SHYR, G. THIERRIN, Power-separating regular languages. *Mathematical Systems Theory* 8 (1974) 1, 90–95.
- [15] B. TRUTHE, *Hierarchy of subregular language families*. Technical Report 1801, Justus-Liebig-Universität Giessen, Institut für Informatik, 2018.
- [16] B. WIEDEMANN, *Vergleich der Leistungsfähigkeit endlicher determinierter Automaten*. Diplomarbeit, Universität Rostock, 1978.

# JUMPING RESTARTING AUTOMATA

Qichao Wang      Yongming Li

College of Computer Science  
Shaanxi Normal University  
Xi'an 710119, China  
{wangqc,liyongm}@snnu.edu.cn

## **Abstract**

Restarting automata have been introduced as a formal model for the analysis by reduction, and it is well known that this model is very expressive under investigation. Here we introduce new variants of restarting automata, where the move-right operation is replaced by a jump operation, which is a restriction on language recognition. Fortunately, we will see that they also have a surprisingly large expressive power. In this work, several variants of jumping restarting automata are investigated, with respect to the computational power. First, we prove that the jumping restarting automata that are allowed to use auxiliary symbols can accept all growing context-sensitive languages. For these automata with auxiliary symbols, the variant that may keep on reading after performing a rewrite step can accept a language that is even not growing context-sensitive. Further, we will see that the deterministic and monotone versions of the jumping restarting automata with auxiliary symbols have the same expressive power as the corresponding types of general restarting automata. Finally, we show that for the types without auxiliary symbols the general restarting automata are strictly more expressive than jumping restarting automata.

## 1. Introduction

The *restarting automaton* was invented as a formal model for the analysis by reduction, which is a linguistic technique that is used to check the correctness of sentences of natural languages through sequences of local simplifications [2]. Such an automaton consists of a finite-state control and a flexible tape with end markers, on which a read/write window of a fixed positive size operates. Based on the state and the window content, the automaton may perform a *move-right step*, which shifts the window one position to the right and changes the state. It may also execute a *rewrite step*, which replaces the content of the window by a word that is strictly shorter, places the window immediately to the right of the newly written word, and changes the state. Finally, it may perform a *restart step*, which moves the window back to the left end of the tape and resets the automaton to its initial state, or it may make an *accept step*. Observe that a rewrite step shortens the content of the tape, and it is assumed that the length of the tape is shortened accordingly. In addition, it is required that before a restart operation can be executed, exactly one rewrite must have taken place, that is, the automaton can be

seen as working in cycles, where each cycle begins with the window at the left end of the tape and the finite-state control being in the initial state, then some move-right steps are executed, then a single rewrite step is performed, then again some move-right steps may be executed, and finally the cycle is completed by a restart step. Thus, a computation consists of a finite sequence of cycles that is followed by a tail computation, which consists of a number of move-right steps, possibly a single rewrite step, and which is completed by an accept step or ends by reaching a configuration for which no further step is defined. In the latter case we say that the current computation halts without acceptance. Many well-known classes of formal languages, like the regular languages REG, the deterministic context-free languages DCFL, the context-free languages CFL, the Church-Rosser languages CRL, and the growing context-sensitive languages GCSL have been characterized by various types of restarting automata. An overview on various types of restarting automata is given by [8].

In this work, we introduce new variants of restarting automata, where the move-right operation is replaced by a jump operation. This means that in such an operation, the automaton jumps ahead on the tape to the first position where the next step of the computation is defined. This kind of operation was previously introduced and studied in the case of finite automata (see, e.g., [5]). Following the same fundamental idea, instead of move-right steps, a jumping restarting automaton is able to directly jump to the position of rewrite by performing a so-called *jump-right* step, so that the computational time can be significantly reduced. However, in a jump-right step, the automaton may skip across some letters in the prefix of input, which is a restriction on language recognition. Fortunately, we will see that jumping restarting automata have a surprisingly large expressive power.

This paper is structured as follows. In Section 2, we recall some basic notions concerning restarting automata, and in Section 3, we introduce the concept of jumping restarting automata. Further, in Section 4, we present a variant of jumping restarting automata, so-called *fast jumping restarting automata*, and study their expressive power. Then, in Section 5, we investigate the *monotone* versions of fast jumping restarting automata, and compare them to the corresponding types of general restarting automata. Finally, the paper closes with a short summary and some problems for future work.

## 2. Restarting Automata

We assume that the reader is familiar with the standard notions and concepts of theoretical computer science. Throughout the paper, we will use  $|w|$  to denote the length of a word  $w$ , and  $\lambda$  to denote the empty word. Further,  $\mathbb{P}(X)$  denotes the power set of a set  $X$ , and  $\mathbb{P}_{\text{fin}}(X)$  denotes the set of all finite subsets of  $X$ . In addition, let  $N$  denote the set of natural numbers.

A *restarting automaton* (RRWW-automaton for short) is a one-tape machine that is defined as an 8-tuple  $M = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, \delta)$ , where  $Q$  is a finite set of states,  $\Sigma$  is a finite input alphabet,  $\Gamma$  is a finite tape alphabet containing  $\Sigma$ , the symbols  $\mathfrak{c}, \$ \notin \Gamma$  serve as markers for the left and right border of the work space, respectively,  $q_0 \in Q$  is the initial state,  $k \in \mathbb{N}_+$  is

the size of the *read/write window*, and

$$\delta : Q \times \mathcal{PC}^{(k)} \rightarrow \mathbb{P}_{\text{fin}}(Q \times (\{\text{MVR}\} \cup \mathcal{PC}^{\leq(k-1)}) \cup \{\text{Restart}, \text{Accept}\})$$

is the *transition function*. Here  $\mathcal{PC}^{(k)}$  is the set of *possible contents* of the read/write window of  $M$ , where  $\mathcal{PC}^{(0)} = \{\lambda\}$  and, for  $i \geq 1$ ,

$$\mathcal{PC}^{(i)} = (\mathfrak{c} \cdot \Gamma^{i-1}) \cup \Gamma^i \cup (\Gamma^{\leq i-1} \cdot \$) \cup (\mathfrak{c} \cdot \Gamma^{\leq i-2} \cdot \$),$$

and

$$\Gamma^{\leq i} = \bigcup_{j=0}^i \Gamma^j, \quad \text{and} \quad \mathcal{PC}^{\leq(k-1)} = \bigcup_{i=0}^{k-1} \mathcal{PC}^{(i)}.$$

The function  $\delta$  describes four different types of transition steps:

- (1) A *move-right step* has the form  $(q', \text{MVR}) \in \delta(q, u)$  (also written as  $(q, u) \rightarrow (q', \text{MVR})$ ), where  $q, q' \in Q$  and  $u \in \mathcal{PC}^{(k)}$ ,  $u \neq \$$ . If  $M$  is in state  $q$  and sees the string  $u$  in its read/write window, then this move-right step causes  $M$  to shift the read/write window one position to the right and to enter state  $q'$ . However, if the content  $u$  of the read/write window is only the symbol  $\$$ , then no move-right step is possible.
- (2) A *rewrite step* has the form  $(q', v) \in \delta(q, u)$  (also written as  $(q, u) \rightarrow (q', v)$ ), where  $q, q' \in Q$ ,  $u \in \mathcal{PC}^{(k)}$ ,  $u \neq \$$ , and  $v \in \mathcal{PC}^{\leq(k-1)}$  such that  $|v| < |u|$ . It causes  $M$  to replace the content  $u$  of the read/write window by the string  $v$ , and to enter state  $q'$ . Further, the read/write window is placed immediately to the right of the string  $v$ . However, some additional restrictions apply in that the border markers  $\mathfrak{c}$  and  $\$$  must not disappear from the tape nor that new occurrences of these markers are created. Further, the read/write window must not move across the right border marker  $\$$ , that is, if the string  $u$  ends in  $\$$ , then so does the string  $v$ , and after performing the rewrite operation, the read/write window is placed on the  $\$$ -symbol.
- (3) A *restart step* has the form  $\text{Restart} \in \delta(q, u)$  (also written as  $(q, u) \rightarrow \text{Restart}$ ), where  $q \in Q$  and  $u \in \mathcal{PC}^{(k)}$ . It causes  $M$  to move its read/write window to the left end of the tape, so that the first symbol it contains is the left border marker  $\mathfrak{c}$ , and to reenter the initial state  $q_0$ .
- (4) An *accept step* has the form  $\text{Accept} \in \delta(q, u)$  (also written as  $(q, u) \rightarrow \text{Accept}$ ), where  $q \in Q$  and  $u \in \mathcal{PC}^{(k)}$ . It causes  $M$  to halt and accept.

For every  $q \in Q$  and  $u \in \mathcal{PC}^{(k)}$ , if  $\delta(q, u) = \emptyset$ , then  $M$  necessarily halts in a corresponding situation, and we say that  $M$  *rejects* in this case. Further, the letters in  $\Gamma \setminus \Sigma$  are called *auxiliary symbols*.

A *configuration* of  $M$  is a string  $\alpha q \beta$ , where  $q \in Q$ , and either  $\alpha = \lambda$  and  $\beta \in \{\mathfrak{c}\} \cdot \Gamma^* \cdot \{\$\}$  or  $\alpha \in \{\mathfrak{c}\} \cdot \Gamma^*$  and  $\beta \in \Gamma^* \cdot \{\$\}$ ; here  $q \in Q$  represents the current state,  $\alpha \beta$  is the current content of the tape, and it is understood that the read/write window contains the first  $k$  symbols of  $\beta$  or all of  $\beta$  when  $|\beta| \leq k$ . A *restarting configuration* is of the form  $q_0 \mathfrak{c} w \$$ , where  $w \in \Gamma^*$ ; if  $w \in \Sigma^*$ , then  $q_0 \mathfrak{c} w \$$  is an *initial configuration*. Thus, initial configurations are a particular type of restarting configurations. Further, we use **Accept** to denote the *accepting configurations*, which

are those configurations that  $M$  reaches by executing an accept instruction. A configuration of the form  $\alpha q \beta$  such that  $\delta(q, \beta_1) = \emptyset$ , where  $\beta_1$  is the current content of the read/write window, is a *rejecting configuration*. A *halting configuration* is either an accepting or a rejecting configuration. By  $\vdash_M$  we denote the single-step computation relation that  $M$  induces on the set of configurations, and its reflexive and transitive closure  $\vdash_M^*$  is the *computation relation* of  $M$ .

In general, the automaton  $M$  is *nondeterministic*, that is, there can be two or more instructions with the same left-hand side  $(q, u)$ , and thus, there can be more than one computation for an input word. If this is not the case, the automaton is *deterministic*. We use the prefix **det-** to denote deterministic classes of restarting automata.

Any finite computation of a restarting automaton  $M$  consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the head moves along the tape performing move-right operations and a rewrite operation until a restart operation is performed and thus, a new restarting configuration is reached. If no further restart operation is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. We require that  $M$  performs *exactly one* rewrite operation during any cycle – thus each new phase starts on a shorter word than the previous one. During a tail at most one rewrite operation may be executed.

An input  $w \in \Sigma^*$  is accepted by  $M$ , if there exists a computation of  $M$  which starts with the initial configuration  $q_0 \mathfrak{c} w \$$ , and which finally ends with executing an accept instruction. The language  $L(M)$  accepted by  $M$  is the set that consists of all input strings that are accepted by  $M$ , that is,

$$L(M) = \{ w \in \Sigma^* \mid q_0 \mathfrak{c} w \$ \vdash_M^* \text{Accept} \}.$$

A restarting automaton is called an **RWW**-automaton if it makes a restart immediately after performing a rewrite operation. In particular, this means that it cannot perform a rewrite step during the tail of a computation. An **RRWW**-automaton is called an **RRW**-automaton if its tape alphabet  $\Gamma$  coincides with its input alphabet  $\Sigma$ , that is, if no auxiliary symbols are available. It is an **RR**-automaton if it is an **RRW**-automaton for which the right-hand side  $v$  of each rewrite step  $(q', v) \in \delta(q, u)$  is a scattered subword of the left-hand side  $u$ . Analogously, we obtain the **RW**-automaton and the **R**-automaton from the **RWW**-automaton. For a type  $\mathbf{X}$  of restarting automata, let  $\mathcal{L}(\mathbf{X})$  denote the class of languages that are accepted by the restarting automata of type  $\mathbf{X}$ .

We complete this section with an example of an **RR**-automaton.

**Example 2.1** Let  $M_1 = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, \delta)$  be the **RR**-automaton that is defined by taking



$Q = \{q_0, q_1, q_2, q_r\}$ ,  $\Gamma = \Sigma = \{a, b, c, d\}$ , and  $k = 3$ , where  $\delta$  is defined as follows:

$$\begin{array}{ll}
 t_1 : (q_0, \mathfrak{c}aa) \rightarrow (q_0, \text{MVR}), & t_{10} : (q_1, bc\$) \rightarrow \text{Restart}, \\
 t_2 : (q_0, \mathfrak{c}ab) \rightarrow (q_0, \text{MVR}), & t_{11} : (q_1, bbc) \rightarrow \text{Restart}, \\
 t_3 : (q_0, aaa) \rightarrow (q_0, \text{MVR}), & t_{12} : (q_2, bbd) \rightarrow \text{Restart}, \\
 t_4 : (q_0, aab) \rightarrow (q_0, \text{MVR}), & t_{13} : (q_2, d\$) \rightarrow \text{Restart}, \\
 t_5 : (q_0, abb) \rightarrow (q_1, b), & t_{14} : (q_2, bd\$) \rightarrow \text{Restart}, \\
 t_6 : (q_0, abb) \rightarrow (q_2, \lambda), & t_{15} : (q_0, abc) \rightarrow (q_r, c), \\
 t_7 : (q_1, bbb) \rightarrow (q_1, \text{MVR}), & t_{16} : (q_r, \$) \rightarrow \text{Restart}, \\
 t_8 : (q_2, bbb) \rightarrow (q_2, \text{MVR}), & t_{17} : (q_0, \mathfrak{c}c\$) \rightarrow \text{Accept}, \\
 t_9 : (q_1, c\$) \rightarrow \text{Restart}, & t_{18} : (q_0, \mathfrak{c}d\$) \rightarrow \text{Accept}.
 \end{array}$$

It is easily seen that  $L(M_1) = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}$ . In each cycle  $M_1$  guesses the suffix of input, and correspondingly removes the factor  $ab$  or  $abb$ . Finally, it moves to the right end of the tape in order to verify its guess.

### 3. Jumping Restarting Automata

In earlier works, restarting automata have been shown to be quite expressive (see, e.g., [4, 8]). Here we introduce a new variant – *jumping restarting automata*.

In analogy to a general restarting automaton, a jumping restarting automaton is also defined as an 8-tuple  $M = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, \delta)$ , and it works in a quite similar way. Instead of move-right steps, a jumping restarting automaton moves by performing *jump-right steps* of the form  $(q, \text{JMR}) \in \delta(p, u)$  (also written as  $(p, u) \rightarrow (q, \text{JMR})$ ). This means that if  $M$  is in state  $p$  and sees the string  $u$  in its read/write window, then this jump-right step causes  $M$  to enter state  $q$ , and to jump right to the nearest factor  $v$ , such that  $\delta(q, v) \neq \emptyset$ . Note that the nearest factor  $v$  can include some symbols of current factor  $u$ . If the tape does not contain such a factor  $v$  that satisfies this condition, then the automaton halts and rejects. In a jump-right step the read/write window moves at least one position to the right. Here we use the prefix **J** to denote the types of jumping restarting automata.

We continue with an example of a JRR-automaton for the language  $L(M_1)$  from Example 2.1. By  $\curvearrowright_M$  we denote the single-step computation relation that a jumping restarting automaton  $M$  induces on the set of configurations, and by  $\curvearrowright_M^*$  we denote its reflexive and transitive closure.

**Example 3.1** Let  $M_2 = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, \delta)$  be the JRR-automaton that is defined by taking

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_r\}$ ,  $\Gamma = \Sigma = \{a, b, c, d\}$ , and  $k = 5$ , where  $\delta$  is defined as follows:

$$\begin{array}{ll}
t_1 : (q_0, \mathbb{C}aaaa) \rightarrow (q_1, \text{JMR}), & t_{11} : (q_3, \text{bbbbd}) \rightarrow \text{Restart}, \\
t_2 : (q_0, \mathbb{C}aaab) \rightarrow (q_1, \text{JMR}), & t_{12.x} : (q_2, xc\$) \rightarrow \text{Restart for } x \in \{\lambda, b, bb, bbb\}, \\
t_3 : (q_0, \mathbb{C}abb) \rightarrow (q_1, \text{JMR}), & t_{13.x} : (q_3, xd\$) \rightarrow \text{Restart for } x \in \{\lambda, b, bb, bbb\}, \\
t_4 : (q_1, aabbb) \rightarrow (q_2, abb), & t_{14} : (q_4, \text{bbbc\$}) \rightarrow \text{Restart}, \\
t_5 : (q_1, aabbb) \rightarrow (q_3, ab), & t_{15} : (q_5, \text{bbbd\$}) \rightarrow \text{Restart}, \\
t_6 : (q_1, aabbc) \rightarrow (q_r, c), & t_{16} : (q_r, \$) \rightarrow \text{Restart}, \\
t_7 : (q_0, \mathbb{C}abd) \rightarrow (q_r, \mathbb{C}d), & t_{17} : (q_0, \mathbb{C}abc\$) \rightarrow \text{Accept}, \\
t_8 : (q_2, \text{bbbbb}) \rightarrow (q_4, \text{JMR}), & t_{18} : (q_0, \mathbb{C}c\$) \rightarrow \text{Accept}, \\
t_9 : (q_3, \text{bbbbb}) \rightarrow (q_5, \text{JMR}), & t_{19} : (q_0, \mathbb{C}d\$) \rightarrow \text{Accept}. \\
t_{10} : (q_2, \text{bbbbc}) \rightarrow \text{Restart}, &
\end{array}$$

We see that in each cycle  $M_2$  directly jumps to the boundary between  $a$ - and  $b$ -symbols, then it nondeterministically removes  $ab$  or  $abb$ , and finally it jumps to the right end of the tape in order to verify its guess for suffix. For example,  $M_2$  can execute the following computation on the input  $aaabbbbbb$ , where we write  $\curvearrowright$  for  $\curvearrowright_{M_2}$ :

$$\begin{array}{l}
q_0\mathbb{C}aaabbbbbb\$ \curvearrowright \mathbb{C}aq_1aabbbbbb\$ \curvearrowright \mathbb{C}aabq_3bbbd\$ \curvearrowright q_0\mathbb{C}aabbbbd\$ \\
\quad \curvearrowright \mathbb{C}q_1aabbbbd\$ \quad \curvearrowright \mathbb{C}abq_3bd\$ \quad \curvearrowright q_0\mathbb{C}abbd\$ \\
\quad \curvearrowright \mathbb{C}dq_r\$ \quad \quad \quad \curvearrowright q_0\mathbb{C}d\$ \quad \quad \quad \curvearrowright \text{Accept}.
\end{array}$$

We now investigate the relation between general restarting automata and jumping restarting automata. First, we establish the following inclusion result.

**Lemma 3.2** For each type  $X \in \{\text{R}, \text{RR}, \text{RW}, \text{RRW}, \text{RWW}, \text{RRWW}\}$ ,  $\mathcal{L}(\text{JX}) \subseteq \mathcal{L}(X)$ .

*Proof.* Let  $M$  be a jumping restarting automaton. We will construct a general restarting automaton  $M'$  that proceeds as follows.  $M'$  performs rewrite, restart, and accept steps exactly as  $M$ . In order to simulate a jump-right transition of  $M$ ,  $M'$  moves its read/write window from left to right across the tape. If  $M'$  discovers the left-hand side of a transition  $t$  of  $M$ , then it executes the transition that corresponds to  $t$ ; otherwise, it halts and rejects.  $\square$

In fact, also the converse inclusions hold.

**Lemma 3.3** For each type  $X \in \{\text{R}, \text{RR}, \text{RW}, \text{RRW}, \text{RWW}, \text{RRWW}\}$ ,  $\mathcal{L}(X) \subseteq \mathcal{L}(\text{JX})$ .

*Proof.* Let  $M = (Q, \Sigma, \Gamma, \mathbb{C}, \$, q_0, k, \delta)$  be a general restarting automaton. For  $M$  we build up a jumping restarting automaton  $M' = (Q', \Sigma, \Gamma, \mathbb{C}, \$, q_0, k + 1, \delta')$  such that  $L(M') = L(M)$ . First, we consider the case that  $X$  is a type with a single  $\text{R}$ , that is, after rewriting  $M$  must immediately restart. The main problem in simulating  $M$  is the fact that we have to ensure that in each jump-right step,  $M'$  moves exactly one position to the right. To do this,  $M'$  needs a read/write window that is larger than that of  $M$ , so that when simulating a transition of  $M$ , it can look ahead for the window content in the next step of  $M$ . Assume that in a cycle,

$M$  reaches the rewrite configuration  $\mathbb{C}a_1a_2 \cdots a_{l-1}q_{l+1}a_la_{l+1} \cdots a_n\mathbb{S}$  by performing the following move-right transitions:

$$\begin{aligned} t_1 &: (q_1, \mathbb{C}a_1 \cdots a_{k-1}) && \rightarrow (q_2, \text{MVR}), \\ t_2 &: (q_2, a_1 \cdots a_k) && \rightarrow (q_3, \text{MVR}), \\ &&& \vdots \\ t_{l-1} &: (q_{l-1}, a_{l-2} \cdots a_{k+l-3}) && \rightarrow (q_l, \text{MVR}), \\ t_l &: (q_l, a_{l-1} \cdots a_{k+l-2}) && \rightarrow (q_{l+1}, \text{MVR}), \end{aligned}$$

and the rewrite transition that will be executed is of the form

$$t_{l+1} : (q_{l+1}, a_l \cdots a_{k+l-1}) \rightarrow (q_{l+2}, v),$$

where  $v$  is a scattered subword of the factor  $a_l \cdots a_{k+l-1}$ . In order to simulate this cycle of  $M$ ,  $\delta'$  contains the following jump-right transitions:

$$\begin{aligned} t'_1 &: (q_1, \mathbb{C}a_1 \cdots a_{k-1}a_k) && \rightarrow (q_2, \text{JMR}), \\ t'_2 &: (q_2, a_1 \cdots a_k a_{k+1}) && \rightarrow (q_3, \text{JMR}), \\ &&& \vdots \\ t'_{l-1} &: (q_{l-1}, a_{l-2} \cdots a_{k+l-3}a_{k+l-2}) && \rightarrow (q_l, \text{JMR}), \\ t'_l &: (q_l, a_{l-1} \cdots a_{k+l-2}a_{k+l-1}) && \rightarrow (q_{l+1}, \text{JMR}), \end{aligned}$$

and the rewrite transition for all  $x \in \Gamma$

$$t'_{l+1} : (q_{l+1}, a_l \cdots a_{k+l-1}x) \rightarrow (q_{l+2}, vx).$$

Analogously, the tail computation can also be dealt with.

Finally, we consider the case that  $\mathbf{X}$  is a type with double  $\mathbf{R}$ , that is, after rewriting  $M$  is able to perform move-right steps. As the window size of  $M'$  is larger than that of  $M$ , after performing a rewrite step  $M'$  skips across the prefix of the window content in next step of  $M$ . Therefore,  $M'$  has to store the prefix in its finite-state control. To do this, we introduce the following additional states of  $M'$ :

$$Q_{rw} = \{\hat{q}_a \mid q \in Q, a \in \Gamma\}.$$

If  $\delta$  contains a rewrite transition of the form

$$(p, u) \rightarrow (q, v),$$

then  $\delta'$  contains the following rewrite transitions for all  $x \in \Gamma$ :

$$(p, ux) \rightarrow (\hat{q}_x, vx).$$

In addition, let  $z = z_1z_2 \cdots z_kz_{k+1}$  be the window content after performing the above rewrite step.  $M'$  can just combine the transitions of  $M$  on the window contents  $xz_1z_2 \cdots z_{k-1}$ ,  $z_1z_2 \cdots z_k$ , and  $z_2z_3 \cdots z_kz_{k+1}$  by using the same technique in the proof of Theorem 6 from [9], which completes our proof.  $\square$

Together Lemma 3.2 and 3.3 yield the following equality result.

**Theorem 3.4** *For each type  $\mathbf{X} \in \{\mathbf{R}, \mathbf{RR}, \mathbf{RW}, \mathbf{RRW}, \mathbf{RWW}, \mathbf{RRWW}\}$ ,  $\mathcal{L}(\mathbf{JX}) = \mathcal{L}(\mathbf{X})$ .*

Obviously, the above equality result also holds in the deterministic case.

## 4. Fast Jumping Restarting Automata

In Section 3, jumping restarting automata have been introduced. However, we see that without any restriction on jump-right operation, the computational power does not change, with respect to the corresponding variants of restarting automata. Here we introduce *fast jumping restarting automata*, where the number of jump-right steps in a cycle is restricted.

In each cycle (or tail computation) a fast jumping restarting automaton is allowed to perform at most one jump-right step before rewrite (or accept), and in tail phase it can perform at most one jump-right step before accept. Further, for the automata of types with RR, there exists a constant  $c \in \mathbb{N}$ , such that after rewriting the automaton is allowed to execute at most  $c$  jump-right steps. This means that in each cycle the number of steps performed is bounded by a constant. Note that the automaton  $M_2$  from Example 3.1 is actually a fast jumping restarting automaton. Here we use the prefix FJ to denote the types of fast jumping restarting automata.

As a fast jumping restarting automaton can perform at most one jump-right step before rewrite, it may skip across some letters in the prefix of the input, which can be seen as a restriction for computation. Hence, the following result is easily obtained.

**Corollary 4.1** *For each type  $X \in \{R, RR, RW, RRW, RWW, RRWW\}$ ,  $\mathcal{L}(\text{FJ}X) \subseteq \mathcal{L}(X)$ .*

In this section, we study the expressive power of fast jumping restarting automata. We begin with FJRWW-automata, and establish the following inclusion result.

**Theorem 4.2**  $\text{GCSL} \subseteq \mathcal{L}(\text{FJRWW})$ .

*Proof.* In [7] it is shown that the language class GCSL is characterized by length-reducing *two-pushdown automata* (TPDA for short). This means that for each language  $L \in \text{GCSL}$ , there exists a length-reducing TPDA  $P$  such that  $L = L(P)$ . A TPDA with pushdown windows of size  $l$  is a nondeterministic automaton  $T = (Q, \Sigma, \Gamma, \delta, l, q_0, Z_0, t_1, t_2, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is a finite input alphabet,  $\Gamma$  is a finite tape alphabet including  $\Sigma$ ,  $q_0 \in Q$  is the initial state,  $Z_0 \in \Gamma \setminus \Sigma$  is the bottom marker of the pushdown stores,  $t_1, t_2 \in (\Gamma \setminus \Sigma)^*$  is the preassigned content of the first and second pushdown store, respectively,  $F \subseteq Q$  is the set of final states, and  $\delta$  is the transition function. In [7] it was claimed that a TPDA can be transformed in a standard form to start the computation with empty string on the first pushdown and the input on the second one. To each triple  $(q, u, v)$ , where  $q \in Q$  is a state,  $u \in \Gamma^l \cup \{Z_0\} \cdot \Gamma^{<l}$  is the content of the topmost part of the first pushdown, and  $v \in \Gamma^l \cup \Gamma^{<l} \cdot \{Z_0\}$  is the content of the topmost part of the second pushdown, it associates a finite set of triples from  $Q \times \Gamma^* \times \Gamma^*$ . A TPDA is called *length-reducing*, if  $|u'v'| < |uv|$  holds for all transitions  $(q, u', v') \in (p, u, v)$ .

A configuration of a TPDA is described by  $uqv$ , where  $q \in Q$  is the actual state,  $u \in \Gamma^*$  is the content of the first pushdown store with the first symbol of  $u$  at the bottom and the last symbol of  $u$  at the top, and  $v \in \Gamma^*$  is the content of the second pushdown store with the last symbol of  $v$  at the bottom and the first symbol of  $v$  at the top. Observe that the input for a TPDA is provided as a part of the initial content of the second pushdown. Let  $P = (Q, \Sigma, \Gamma, \delta, l, q_0, Z_0, t_1, t_2, F)$  be a length-reducing TPDA. In order to prove the above inclusion, we construct an FJRWW-auto-

maton  $M = (Q', \Sigma, \Gamma', \mathfrak{c}, \mathfrak{s}, q'_0, k, \delta')$  such that  $L(M) = L(P)$ . In order to simulate a computation of  $P$ , for the letters on the tape,  $M$  has to distinguish between the contents of the first and second pushdown stores, and remember the state before restart, as it forgets the state after performing a restart step. To do this, let  $\Gamma'$  be defined as

$$\Gamma' = \Sigma \cup \{ \hat{a} \mid a \in \Gamma \} \cup \{ [\hat{a}, q] \mid a \in \Gamma, q \in Q \}.$$

Then a configuration  $uaqv$  of  $P$  can be encoded by tape content  $\mathfrak{c}\hat{u}[\hat{a}, q]v\mathfrak{s}$ , where  $u, v \in \Gamma^*$ ,  $a \in \Gamma$ ,  $q \in Q$ , and  $\hat{u}$  is a copy of  $u$  that consists of marked symbols. It is easily seen that the position of  $[\hat{a}, q]$  corresponds to the boundary between the contents of the first and second pushdown stores, and  $q$  is the current state of  $P$ . In each cycle  $M$  directly jumps to the symbol of the form  $[\hat{a}, q]$ , and let this symbol stand in the middle of the read/write window. If we set  $k = 2l$ , then  $M$  can see the actual state of  $P$ , and the top  $l$  symbols on the first and second pushdown stores of  $P$ , so that  $M$  can simulate the following operation of  $P$ . In addition, for each transition  $(q, u', v') \in (p, u, v)$  of  $P$  it holds that  $|u'v'| < |uv|$ . Hence,  $M$  is able to simulate  $P$  in length-reducing fashion.  $\square$

It is still open whether or not the above inclusion is proper. By using the same technique, we can prove that **det-FJRW**-automata can simulate deterministic length-reducing TPDAs. It is well-known that the language class **CRL** is characterized by deterministic length-reducing TPDAs [7], which coincide with general **det-(R)RWW**-automata [6]. Hence, the following equality result can be immediately given.

**Theorem 4.3**

$$\begin{aligned} \text{CRL} &= \mathcal{L}(\text{det-FJRW}) = \mathcal{L}(\text{det-FJRRWW}) \\ &= \mathcal{L}(\text{det-JRWW}) = \mathcal{L}(\text{det-JRRWW}) \\ &= \mathcal{L}(\text{det-RWW}) = \mathcal{L}(\text{det-RRWW}). \end{aligned}$$

We see that deterministic fast jumping restarting automata with auxiliary symbols are as expressive as the corresponding types of general restarting automata.

Now we turn to **FJRRWW**-automata. In [1] the Gladkij language is defined as

$$L_{Gl} = \{ w\#w^R\#w \mid w \in \{a, b\}^* \},$$

and it is well-known that  $L_{Gl} \notin \text{GCSL}$ . For the language  $L_{Gl}$  we have the following result.

**Proposition 4.4**  $L_{Gl} \in \mathcal{L}(\text{FJRRWW})$ .

*Proof.* We will construct an **FJRRWW**-automaton  $M$  that accepts the language  $L_{Gl}$ . This means that for an input of the form  $u\#v\#w$   $M$  can determine whether  $u = w = v^R$ . We only consider the case that  $|u|, |v|$  and  $|w|$  are much larger than the window size  $k$  of  $M$ . In this case  $M$  has to alternately shorten these strings, and during this process it compares the corresponding parts of them. At the beginning of a cycle, the read/write window is on the prefix of  $u$ , and here  $M$  rewrites some letters of  $u$ , and in the next cycle it jumps to the boundary between the infix  $v$  and suffix  $w$  and rewrites some letters of them, where these two stages are repeated alternately. However, as  $M$  directly jumps to the position of rewrite, and

there are two  $\#$ -symbols, it has to ensure that it jumps to the second one. To do this,  $M$  has to rewrite the first  $\#$ -symbol in order to distinguish it from the second one.

Let  $M = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, \delta)$  be an FJRRWW-automaton that proceeds as follows. First,  $M$  jumps to the first  $\#$ -symbol, and combines the first  $\#$ -symbol with the suffix letter  $u_r$  of  $u$  into a special symbol of the form  $[u_r, \#]$ . Then  $M$  replaces the first two symbols  $u_1$  and  $u_2$  of  $u$  by an auxiliary of the form  $[u_1, u_2, 1]$ . In the following cycle it jumps to the boundary between the infix  $v$  and the suffix  $w$ . Let  $u = u_1u_2 \dots u_r$ ,  $v = v_1v_2 \dots v_r$ , and  $w = w_1w_2 \dots w_r$ . If the  $u_1 = v_r = w_1$  and  $u_2 = v_{r-1} = w_2$ , then  $M$  replaces  $v_{r-1}v_r$  by an auxiliary symbol of the form  $[v_{r-1}, v_r, 2]$ , and replaces  $w_1w_2$  by an auxiliary of the form  $[w_1, w_2, 3]$ . Then,  $M$  removes the symbol  $[u_1, u_2, 1]$ , and in the following cycle removes the symbols  $[v_{r-1}, v_r, 2]$  and  $[w_1, w_2, 3]$ . Therefore, we can describe the above process by the following sequence

$$\begin{aligned}
& \mathfrak{c}u_1u_2 \dots u_r\#v_1 \dots v_{r-1}v_r\#w_1w_2 \dots w_r\$ & (0) \\
\rightarrow & \mathfrak{c}u_1u_2 \dots u_{r-1}[u_r, \#]v_1 \dots v_{r-1}v_r\#w_1w_2 \dots w_r\$ & (1) \\
\rightarrow & \mathfrak{c}[u_1, u_2, 1]a_3 \dots u_{r-1}[u_r, \#]v_1 \dots v_{r-1}v_r\#w_1w_2 \dots w_r\$ & (2) \\
\rightarrow & \mathfrak{c}[u_1, u_2, 1]a_3 \dots u_{r-1}[u_r, \#]v_1 \dots v_{r-2}[v_{r-1}, v_r, 2]\#[w_1, w_2, 3]w_3 \dots w_r\$ & (3) \\
\rightarrow & \mathfrak{c}a_3 \dots u_{r-1}[u_r, \#]v_1 \dots v_{r-2}[v_{r-1}, v_r, 2]\#[w_1, w_2, 3]w_3 \dots w_r\$ & (4) \\
\rightarrow & \mathfrak{c}a_3 \dots u_{r-1}[u_r, \#]v_1 \dots v_{r-2}\#w_3 \dots w_r\$ & (5)
\end{aligned}$$

Of course,  $M$  often needs to guess which operation it should perform. For a window content  $\mathfrak{c}\alpha$  on a restarting configuration, if  $\alpha \in \Sigma^*$ ,  $M$  has to guess whether the first  $\#$ -symbol is already rewritten in the form  $[a, \#]$ , or whether the symbols of the form  $[a, b, 2]$  and  $[a, b, 3]$  are already removed. Thus, on Stage (1), (2) and (4) after rewriting  $M$  has to jump to the first and second  $\#$ -symbols in order to verify its guess. It is rather clear that after a rewrite step  $M$  needs at most two jump-right steps, and this is bounded by a constant. The above process is repeated, until all letters of  $u, v$  and  $w$  are erased, and the symbol  $[a, \#]$  is rewritten in  $[\#]$ . Finally,  $M$  accepts on the tape content  $\mathfrak{c}[\#]\#\$$ . If  $M$  discovers that there are more than two  $\#$ -symbols, or that the second  $\#$  is rewritten, then it halts and rejects.  $\square$

The above result separates the class GCSL from the class  $\mathcal{L}(\text{FJRRWW})$ . Hence, we immediately obtain the following consequence.

**Theorem 4.5**  $\text{GCSL} \subsetneq \mathcal{L}(\text{FJRRWW})$ .

Note that it remains open whether or not the language  $L_{GI}$  can be accepted by an FJRRWW-automaton. Now we investigate the relation between fast jumping restarting automata without auxiliary symbols and the corresponding types of general jumping restarting automata. To do this, we consider the following example language:

$$L_0 = \{ a^i c a^j b^l \mid i, j, l \geq 0, i + j = l \} \cup \{ a^i d a^j b^l \mid i, j, l \geq 0, 2(i + j) = l \}.$$

For the following result, we need the so-called *error preserving property* (see, e.g., [3]), which states that for each restarting automaton  $M = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, \delta)$ , if  $q_0 \mathfrak{c}u\$ \vdash_M^{\mathfrak{c}^*} q_0 \mathfrak{c}v\$$  holds and  $u \notin L(M)$ , then  $v \notin L(M)$ , either.

**Lemma 4.6**  $L_0 \notin \mathcal{L}(\text{FJRRW})$ .

*Proof.* We assume that there exists an FJRRW-automaton  $M = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, \delta)$  such that  $L_0 = L(M)$ . For an input of the form  $a^i x a^j b^l$  ( $x \in \{c, d\}$ ), here we choose  $i, j, l > 2k$ . As the length of the input is greater than the size of read/write window of  $M$ , and as the language  $L_0$  is not regular, the input cannot be accepted in a tail computation, and  $M$  needs to shorten both prefix  $a^i x a^j$  and the suffix  $b^l$  in order to compare the numbers of  $a$ - and  $b$ -symbols. Here we distinguish between two cases.

- (1) In each cycle,  $M$  will have the option to either execute a rewrite step on the restarting configuration or jump to the right. We assume that  $M$  directly jumps to the boundary between the prefix and the suffix. As  $i, j, l > 2k$ , the symbol  $x \in \{c, d\}$  cannot be in the read/write window at the beginning of a cycle, and also not in the window when it stands on the boundary between the prefix and the suffix. Therefore,  $M$  can only guess the value of  $x$ , and then it removes correspondingly many  $a$ - and  $b$ -symbols. However, without auxiliary symbols  $M$  cannot remember its guess after a restart step. Therefore, an input  $a^i c a^i b^{2i+r} \notin L_0$  can be reduced to the word  $a^r c a^r b^{2r} \in L_0$  for  $0 < r < i$ , which contradicts the error preserving property.
- (2) In order to solve the above problem,  $M$  can first jump to the symbol  $x$ , and then based on the value of  $x$  it removes a corresponding number of  $a$ - and  $b$ -symbols. However, as in each cycle a fast jumping restarting automaton is allowed to execute at most one jump-right step before rewrite,  $M$  must perform a rewrite step on a factor of the form  $a^r x a^t$  for  $r + t + 1 = k$ . It follows that in the same cycle,  $M$  cannot shorten the segments of  $a$ - and  $b$ -symbols anymore. Further, after restarting  $M$  reenters the initial state  $q_0$ , and it is not able to remember the value of  $x$ , as in a rewrite step the factor  $a^r x a^t$  is replaced by a shorter string  $v$  consisting of only input symbols. In addition, in such a rewrite step, the number of  $b$  will not be changed, while the number of  $a$ -symbols is reduced, or the symbol  $x$  or some  $a$ -symbols are replaced by other input symbols, which also leads to a contradiction with the error preserving property.

□

Based on Lemma 4.6 the following proper inclusion result can be obtained.

**Theorem 4.7** For each type  $X \in \{\text{R}, \text{RR}, \text{RW}, \text{RRW}\}$ ,

- (1)  $\mathcal{L}(\text{FJX}) \subsetneq \mathcal{L}(\text{JX})$ ,
- (2)  $\mathcal{L}(\text{det-FJX}) \subsetneq \mathcal{L}(\text{det-JX})$ .

*Proof.* It is rather clear that the above inclusions follow from the definition of (fast) jumping restarting automata. In order to prove their properness, we construct a **det-R**-automaton  $M$  that accepts the language  $L_0$ , and it proceeds as follows. For an input of the form  $a^i x a^j b^l$  ( $x \in \{c, d\}$ ), in each cycle,  $M$  moves from left to right. On seeing the symbol  $x$ , it stores the value of  $x$  in its finite-state control, and continues moving to the boundary between  $a$ - and  $b$ -symbols. According to the value of  $x$ ,  $M$  removes correspondingly many  $a$ - and  $b$ -symbols, and then restarts. Note that the symbol  $x$  will not be deleted, and finally  $M$  accepts on the

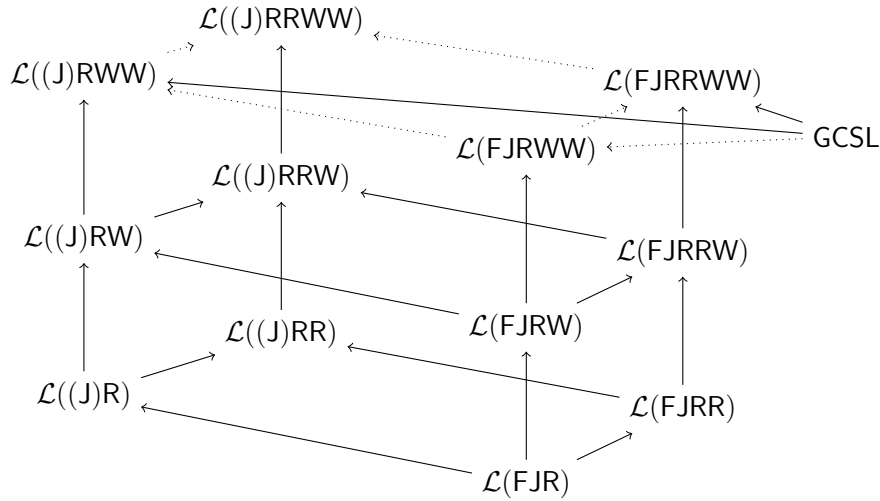


Figure 1: Hierarchy of classes of languages that are computed by the various types of (fast) jumping restarting automata. A dotted arrow denotes an inclusion, and a solid arrow denotes a proper inclusion.

tape content  $\epsilon x \$$ . It is easily seen that  $L_0 = L(M)$ . Further, by Theorem 3.4, it follows that  $L_0 \in \mathcal{L}(\text{det-JR})$ , and by Lemma 4.6, it follows that  $L_0 \notin \mathcal{L}(\text{FJRRW})$ , which completes this proof.  $\square$

Obviously, the relations between various types of general restarting automata without auxiliary symbols carry over to the corresponding types of fast jumping restarting automata, which are separated from each other by the same languages given in [3, 8]. The inclusion results obtained on fast jumping restarting automata and deterministic versions of them are summarized in Figure 1 and 2, respectively.

## 5. Monotone Fast Jumping Restarting Automata

There are many restricted types of restarting automata. Here we restate the notion of *monotonicity* for restarting automata [3]. Let  $C = \alpha q \beta$  be a *rewrite configuration* of a restarting automaton  $M$ , that is, a configuration in which a rewrite step can be applied. Then  $|\beta|$  is called the *right distance* of  $C$ , which is denoted by  $D_r(C)$ . A *sequence of rewrite configurations*  $S = (C_1, C_2, \dots, C_n)$  is called *monotone* if  $D_r(C_1) \geq D_r(C_2) \geq \dots \geq D_r(C_n)$ , that is, if the distance of the place of rewriting to the right end of the tape does not increase from one rewrite step to the next. A computation of a restarting automaton  $M$  is called *monotone* if the sequence of rewrite configurations that is obtained from the cycles of that computation is monotone. Observe that here the rewrite configuration which corresponds to the possible rewrite step that is executed in the tail of the computation considered is not taken into account. Finally, a restarting automaton  $M$  is called *monotone* if all its computations that start with an initial configuration are monotone. We use the prefix **mon-** to denote monotone types of restarting automata. In this section, we consider monotone fast jumping restarting automata. We begin with **mon-FJRRW**-automata, and establish the following inclusion result.



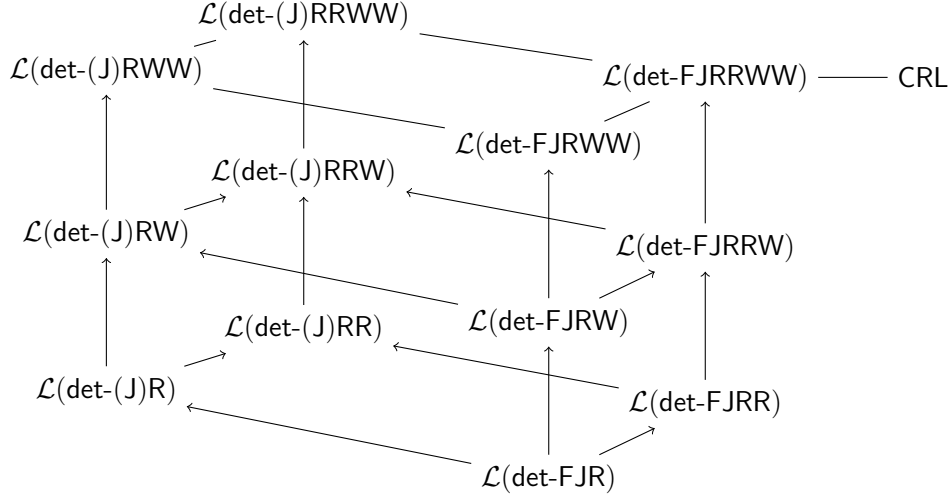


Figure 2: Hierarchy of classes of languages that are computed by the various types of deterministic (fast) jumping restarting automata. An arrow denotes a proper inclusion, and a solid line denotes an equality relation.

**Lemma 5.1**  $\mathcal{L}(\text{mon-RWW}) \subseteq \mathcal{L}(\text{mon-FJRWW})$ .

*Proof.* Let  $M = (Q, \Sigma, \Gamma, \epsilon, \$, q_0, k, \delta)$  be a **mon-RWW**-automaton. We will construct a **mon-FJRWW**-automaton  $M'$  such that  $L(M') = L(M)$ . The main problem in simulating  $M$  is the fact that in each cycle,  $M'$  is allowed to perform only at most one jump-right step, after which it must immediately execute a rewrite/restart step, while  $M$  is able to move right along the tape to the right end. Here we apply the simulation technique that is used in the proof of Theorem 4.2. In order to simulate move-right transitions of  $M$ ,  $M'$  has to store the current state of  $M$  on the tape, as it cannot remember the state after a rewrite/restart step. Further, each rewrite operation must be strictly length-reducing. In order to satisfy this condition,  $M'$  combines two move-right steps of  $M$  in a single step, and thus it needs a read/write window that is larger than  $M$ . Then,  $M'$  rewrites two letters on the tape in an auxiliary symbol with the current state of  $M$ , i.e., of the form  $[a, b, q]$ , where  $a, b \in \Gamma$ , and  $q \in Q$ . As  $M$  is monotone, in each cycle  $M'$  can directly jump to the rightmost symbol of the form  $[a, b, q]$ , and on seeing the current state of  $M$  it continues simulating the rest computation of  $M$ .  $\square$

In [3] it is shown that the language class CFL is characterized by **mon-(R)RWW**-automata. Hence, the following result can be easily obtained.

**Theorem 5.2**

$$\begin{aligned} \text{CFL} &= \mathcal{L}(\text{mon-FJRWW}) = \mathcal{L}(\text{mon-FJRRWW}) \\ &= \mathcal{L}(\text{mon-JRWW}) = \mathcal{L}(\text{mon-JRRWW}) \\ &= \mathcal{L}(\text{mon-RWW}) = \mathcal{L}(\text{mon-RRWW}). \end{aligned}$$

It is easy to see that the simulation technique used in the proof of Lemma 5.1 can also be applied in the deterministic case. Further, it is well-known that the language class DCFL coincides with the classes of languages that are computed by **det-mon-R(R)(W)(W)**-automata [3]. Note that

a fast jumping restarting automaton without auxiliary symbols is not able to apply the above simulation technique, as it cannot store the current state of  $M$  on the tape. Hence, we can establish the following result.

**Theorem 5.3** *For each type  $X \in \{R, RR, RW, RRW, RWW, RRWW\}$ ,*

$$\begin{aligned} \text{DCFL} &= \mathcal{L}(\text{det-mon-FJRW}) = \mathcal{L}(\text{det-mon-FJRRW}) \\ &= \mathcal{L}(\text{det-mon-X}) = \mathcal{L}(\text{det-mon-JX}). \end{aligned}$$

We see that monotone fast jumping restarting automata with auxiliary symbols have the same computational expressive power as general restarting automata of corresponding types, and this is also true in the deterministic case.

Now we turn to the monotone fast jumping restarting automata without auxiliary symbols. As the  $\text{det-R}$ -automaton for the language  $L_0$  given in the proof of Theorem 4.7 is actually monotone, the following proper inclusions can be easily obtained.

**Theorem 5.4** *For each type  $X \in \{R, RR, RW, RRW\}$ ,*

- (1)  $\mathcal{L}(\text{mon-FJX}) \subsetneq \mathcal{L}(\text{mon-JX})$ ,
- (2)  $\mathcal{L}(\text{det-mon-FJX}) \subsetneq \mathcal{L}(\text{det-mon-JR})$ .

It is rather clear the relations between various types of general monotone restarting automata without auxiliary symbols carry over to the corresponding types of fast jumping restarting automata, which are separated from each other by the same languages given in [3, 8]. However, the relations between deterministic fast jumping restarting automata without auxiliary symbols are still unknown. Further, in [8] it was claimed that the class DCFL is a proper subset of the class  $\mathcal{L}(\text{mon-R})$ , and the language  $L_4 = \{a^n b^m \mid 0 < n \leq m \leq 2n\} \in \mathcal{L}(\text{mon-R}) \setminus \text{DCFL}$ . Obviously, the language  $L_4$  can also be accepted by a  $\text{mon-FJR}$ -automaton. Hence, we have the following incomparability result.

**Theorem 5.5** *The language class DCFL is incomparable to the language classes  $\mathcal{L}(\text{mon-FJR})$ ,  $\mathcal{L}(\text{mon-FJRR})$ ,  $\mathcal{L}(\text{mon-FJRW})$ , and  $\mathcal{L}(\text{mon-FJRRW})$  with respect to inclusion.*

Finally, we summarize the inclusion relations between the classes of languages that are accepted by various types of monotone jumping restarting automata in the diagram in Figure 3.

## 6. Conclusions

We have studied the class of languages that are computed by various types of (fast) jumping restarting automata, both in deterministic case as well as in the nondeterministic case. We have seen that FJRW-automata can accept all languages from the class GCSL, which is even strictly contained in the class  $\mathcal{L}(\text{FJRRW})$ . Further, the class CRL can be characterized by

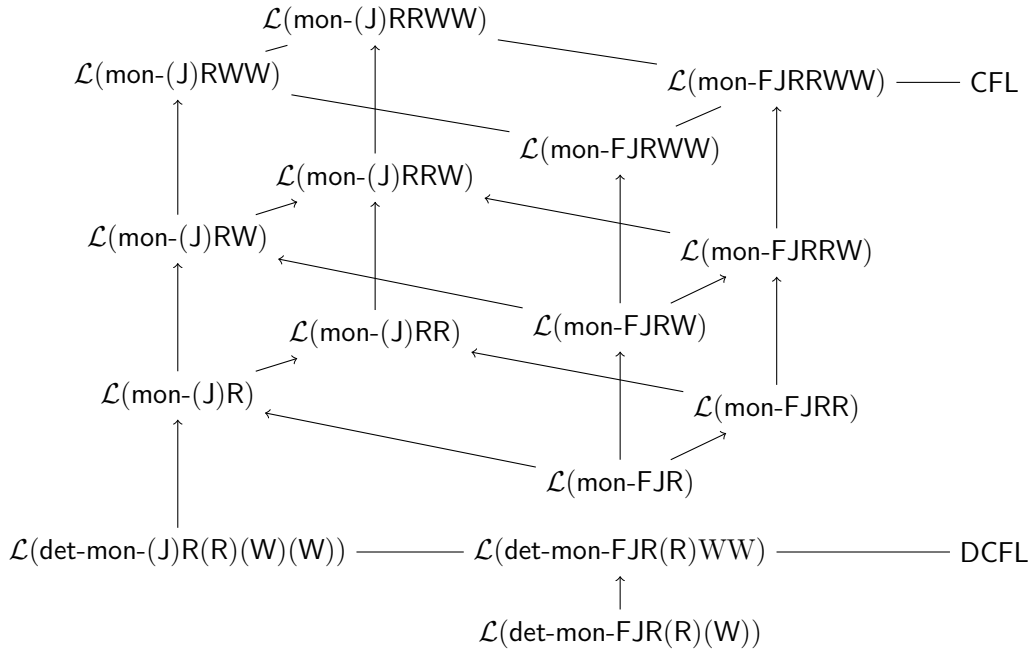


Figure 3: Inclusions between the language classes defined by various types of (fast) jumping restarting automata. An arrow denotes a proper inclusion, a solid line denotes an equality relation, and classes that are not connected through a sequence of arrows are incomparable with respect to inclusion.

det-FJR(R)WW-automata just like general det-R(R)WW-automata. Then, we have proved that (monotone) fast jumping restarting automata that are not allowed to use auxiliary symbols have a weaker expressive power than the corresponding types of general restarting automata. Finally, we have obtained that the class of languages that are accepted by mon-FJR(R)WW coincides with the class CFL, and this is also true in the deterministic case, which means that these automata are expressive exactly as (det-)mon-R(R)WW. In addition, it is easily seen that (fast) jumping restarting automata have the same closure properties as general restarting automata.

However, it is still open whether the inclusion  $GCSL \subseteq \mathcal{L}(FJRWW)$  is proper. Further, it remains to derive a characterization of the classes of languages that are computed by the fast jumping restarting automata without auxiliary symbols. In particular, the inclusion relation between the class REG and the class of languages that are computed by these automata without auxiliary symbols is still unknown.

## References

[1] A. W. GLADKIJ, On the complexity of derivations for context-sensitive grammars [in Russian]. *Algebra i Logika Sem.* 3 (1964), 29–44.

[2] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting automata. In: H. REICHEL (ed.), *International Symposium on Fundamentals of Computation Theory (FCT 1995)*. Lecture

Notes in Computer Science 965, Springer, 1995, 283–292.

- [3] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On monotonic automata with a restart operation. *Journal of Automata, Languages, and Combinatorics (JALC)* 4 (1999) 4, 287–311.
- [4] T. JURDZIŃSKI, K. LORYŚ, G. NIEMANN, F. OTTO, Some results on RWW- and RRWW-automata and their relation to the class of growing context-sensitive languages. *Journal of Automata, Languages and Combinatorics (JALC)* 9 (2004) 4, 407–437.
- [5] A. MEDUNA, P. ZEMEK, Jumping finite automata. *International Journal of Foundations of Computer Science* 23 (2012) 7, 1555–1578.
- [6] G. NIEMANN, F. OTTO, Further results on restarting automata. In: M. ITO, T. IMAOKA (eds.), *Proceedings of the International Colloquium on Words, Languages & Combinatorics III*. World Scientific, 2000, 352–369.
- [7] G. NIEMANN, F. OTTO, The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages. *Information and Computation* 197 (2005) 1–2, 1–21.
- [8] F. OTTO, Restarting automata. In: Z. ÉSIK, C. MARTÍN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence 25, Springer, 2006, 269–303.
- [9] F. OTTO, Q. WANG, Weighted restarting automata. *Soft Computing* 22 (2018) 4, 1067–1083.

# ONE-WAY TOPOLOGICAL AUTOMATA AND THE TANTALIZING EFFECTS OF THEIR TOPOLOGICAL FEATURES

Tomoyuki Yamakami

Faculty of Engineering, University of Fukui  
3-9-1 Bunkyo, Fukui, 910-8507 Japan  
TomoyukiYamakami@gmail.com

## **Abstract**

*We cast new light on the existing models of 1-way deterministic topological automata by introducing a new, convenient model, in which, as each input symbol is read, an interior system of an automaton known as a configuration continues to evolve in a topological space by applying continuous transition operators one by one. The acceptance and rejection of a given input are determined by observing the interior system after the input is completely processed. Such automata naturally generalize 1-way finite automata of various types, including deterministic, probabilistic, and measure-many quantum finite automata. We examine the strengths and weaknesses of the power of this new automata model when recognizing formal languages. We investigate tantalizing effects of various topological features of our topological automata by analyzing the behaviors of the automata when different kinds of topological spaces and continuous maps, which are used respectively as configuration spaces and transition operators, are provided to the automata.*

**Keywords:** *topological automata, topological space, continuous map, compact, quantum finite automata, probabilistic finite automata*

## 1. Topological Automata as Input Acceptors

*Finite-state automata (finite automata, or even automata) are one of the simplest and the most intuitive mathematical models used to describe “mechanical procedures,” each of which depicts a finite number of “operations” in order to determine the membership of any given input word to a fixed language. Over decades of their study, these machines have found numerous applications in the fields of engineering, physics, biology, and even economy (see, e.g., [4]). Each machine reads input symbols one by one and then processes them by changing a status of the machine’s interior system step by step. In particular, a one-way<sup>1</sup> (or real-time) finite automaton receives*

---

<sup>1</sup>Here, we use the term “1-way” to exclude the use of  $\lambda$ -moves, which are particular transitions of the machine with its tape head staying still, where  $\lambda$  refers to the empty string. On the contrary, finite automata that make  $\lambda$ -moves are sometimes called *1.5-way* finite automata.

streamlined input data and processes it piece by piece by applying operations predetermined for each of the input symbols. For such a machine, a *computation* is a description of a series of “evolutions” of the interior system.

To cope with numerous computational problems, various types of finite automata have been proposed in the past literature as their appropriate computational models. In the 1970s, many features of the existing 1-way finite automata were generalized into so-called “topological automata” (see [3] for early expositions and references therein). Topological automata embody characteristic features of various types of finite automata and this fact has helped us take a unified approach toward the study of formal languages and automata theory. The analysis of topological features of the topological automata thus guides us to a better understanding of the theory itself.

Back in the time of 1970s, Brauer (see references in [3]) and Ehrig and Kühnel [3] discussed topological automata as a topological generalization of *Mealy machines*, which behave as “transducers,” which simply produce outputs from inputs. In contrast, following a discussion of Bozapalidis [2] on a generalization of stochastic functions and quantum functions (see also [10]), Jeandel [5] studied another type of topological automata that behave as “acceptors” of inputs. Jeandel’s model naturally generalizes not only *probabilistic finite automata* [9] but also *measure-once quantum finite automata* [8]. The main motivation of Jeandel’s work, nonetheless, was to study a nondeterministic variant of quantum finite automata and he thus used his topological automata to obtain an upper-bound of the language recognition power of nondeterministic quantum finite automata. Concerning the types of “inputs” fed into topological automata, in contrast, Ehrig and Kühnel [3] applied a quite general framework to inputs, which are taken from arbitrary *compactly generated Hausdorff spaces*, whereas Jeandel [5] used the standard framework with finite alphabets and languages generated over them. Jeandel further took “measures” (which assign real numbers to final configurations) to determine the acceptance or rejection of inputs. Since we are more concerned with the computational power of topological automata in comparison with the existing finite automata, we wish to make our model as simple and intuitive as possible by introducing, unlike the use of measures, sets of accepting and rejecting configurations, into which the machine’s interior system finally falls.

Given an input string over a fixed alphabet  $\Sigma$ , the evolution of an interior status of our topological automaton is described in the form of a series of *configurations*, which constitutes a *computation* of the machine. A list of transition operators thus serves as a “program”, which completely dictates the behaviors of the machine on each input. Since arbitrary topological spaces can be used as configuration spaces, topological automata are no longer “finite-state” machines; however, they evolve sequentially as they read input symbols one by one until they completely read the entire inputs and final configurations are observed once (referred to as an “observe once” feature). Moreover, our topological automata enjoy a “deterministic” nature in the sense that which transition operators are applied to the current configurations is completely determined by input symbols alone. This gives rise to “1-way deterministic topological automata” (or *1dta’s*, in short). Although their tape heads move in one direction from the left to the right, 1dta’s turn out to be quite powerful in recognizing formal languages.

This paper intends to shed new light on the basic structures of topological automata and the

acting roles of their transition operators that force configurations to evolve consecutively. For this purpose, we start our study with a suitable abstraction of 1-way finite automata using arbitrary topological spaces for configurations and arbitrary continuous maps for transitions. Such an abstraction serves as a skeleton to construct our topological automata. We call this skeleton an *automata base*.

In general, the choice of topologies significantly affects the computational power of topological automata. As shown later, the trivial topology induces the language family composed only of  $\emptyset$  and  $\Sigma^*$  (for each fixed alphabet  $\Sigma$ ) whereas the discrete topology allows topological automata to recognize arbitrary complex languages. All topologies on a fixed space  $V$  form a complete lattice; thus, it is possible to classify the topologies according to the endowed power of associated topological automata.

Our study on topological automata may be focused on achieving the following four goals.

1. Understand how various choices of topological spaces and continuous maps affect the computational power of underlying machines by clarifying the strengths and weaknesses of the language recognition power of the machines.
2. Determine what kinds of topological features of topological automata nicely characterize the existing finite automata of various types by examining the descriptive power of such features.
3. Explore different types of topological automata to capture fundamental properties (such as closure properties) of formal languages and finite automata.
4. Find useful applications of topological automata to other fields of science.

In Section 2, we will formulate our basic model of 1dta's. These automata are naturally induced from automata bases and they can express numerous types of the existing 1-way finite automata. Through Section 3, we will discuss basic properties of the 1dta's, including closure properties and the elimination of two endmarkers. Following an exploration of such basic properties, we will compare different topologies in Section 4 by measuring how much computational power is endowed to underlying topological automata. In Section 5, we will show that unique features of well-known topological concepts, such as compactness and equicontinuity, help us characterize 1-way deterministic finite automata (or 1dfa's). We will lay out a necessary and sufficient condition on a topological space for which underlying machines are no more powerful than 1dfa's. In Section 6, we will consider a nondeterministic variant of our topological automata (called 1nta's) by introducing *multi-valued* transition operators. It is known that, for weak machine models, nondeterministic machines can be easily simulated by deterministic ones. By formalizing this situation, we will argue what kind of topology makes 1nta's simulated by 1dta's.

We strongly hope that this work initiates a systematic study on the significant roles of topologies played by topological automata and also it leads to better understandings of ordinary finite automata in the end.

## 2. A Basic Model of Our Topological Automata

One-way (deterministic) topological automata can represent many of the existing 1-way finite automata. We begin our study of such powerful automata by describing their “basic” framework, which we intend to call an *automata base*.

Let  $\mathbb{Z}$ ,  $\mathbb{R}$ , and  $\mathbb{C}$  respectively indicate the sets of all *integers*, of all *real numbers*, and of all *complex numbers*. We denote by  $\mathbb{N}$  the set of all nonnegative integers (or *natural numbers*) and set  $\mathbb{N}^+$  to be  $\mathbb{N} - \{0\}$ . For any two integers  $m$  and  $n$  with  $m \leq n$ , an *integer interval*  $[m, n]_{\mathbb{Z}}$  expresses the set  $\{m, m + 1, m + 2, \dots, n\}$  in contrast with a real interval  $[\alpha, \beta]$  for  $\alpha, \beta \in \mathbb{R}$ . We further abbreviate  $[1, n]_{\mathbb{Z}}$  as  $[n]$  for any number  $n \in \mathbb{N}^+$ .

Given a set  $X$ , the notation  $\mathcal{P}(X)$  denotes the *power set* of  $X$ , i.e., the set of all subsets of  $X$ , and  $\mathcal{P}(X)^+$  expresses  $\mathcal{P}(X) - \{\emptyset\}$ . A *monoid*  $C$  is a semigroup with an identity  $I$  in  $C$  and an associative binary operator  $\circ$  on  $C$ .

### 2.1. Topologies and Automata Bases

In 1970s, *topological automata* were sought to take inputs from arbitrary topological spaces (e.g., [3]). Although such a general treatment of topological automata provides a bird-eye view of a topological landscape of a standard setting of formal languages and automata theory, as noted in Section 1, we wish to limit our interest within fixed discrete alphabets because our intention is to compare the language recognition power of topological automata with the existing finite automata that deal only with languages over small discrete alphabets.

Hereafter, an *alphabet* refers to a nonempty finite set  $\Sigma$  of “symbols” (or “letters”) and a *string* over alphabet  $\Sigma$  is a finite sequence of symbols in  $\Sigma$ . The *length*  $|x|$  of string  $x$  is the number of all occurrences of symbols in  $x$ . The *empty string* is a special string of length 0 and is denoted by  $\lambda$ . Given two strings  $x$  and  $y$ ,  $x$  is an *initial segment* of  $y$  if  $y = xz$  holds for a certain string  $z$ . For each number  $n \in \mathbb{N}$ ,  $\Sigma^n$  denotes the set of all strings of length exactly  $n$ ; moreover, we set  $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$  and a *language* over  $\Sigma$  is a subset of  $\Sigma^*$ . A language is called *unary* (or *tally*) if it is defined over a single-letter alphabet. Given a language  $L$  over  $\Sigma$ , we use the same symbol  $L$  to denote its *characteristic function*; that is, for any  $x \in \Sigma^*$ ,  $L(x) = 1$  if  $x \in L$ , and  $L(x) = 0$  otherwise. For two languages  $A$  and  $B$  over  $\Sigma$ , the notation  $AB$  denotes  $\{xy \mid x \in A, y \in B\}$ . In particular, when  $A$  is a singleton  $\{s\}$ , we write  $sB$  in place of  $\{s\}B$ . Similarly, we write  $As$  for  $A\{s\}$ .

To discuss structures of topological automata, we first introduce a fundamental notion of “automata base,” which is a skeleton of various topological automata introduced in Section 2.2. For this purpose, we want to review basic terminology in the theory of *general topology* (or *point-set topology*). Given a set  $V$  of *points*, a *topology*  $T_V$  on  $V$  is an axiomatic<sup>2</sup> collection of subsets of  $V$ , called *open sets*. Hence,  $T_V$  is a subset of  $\mathcal{P}(V)$ . With respect to  $V$ , the

<sup>2</sup>There are three axioms for  $T_V$  to satisfy. (1)  $\emptyset, V \in T_V$ . (2) Any (finite or infinite) union of sets in  $T_V$  is also in  $T_V$ . (3) Any finite intersection of sets in  $T_V$  belongs to  $T_V$ .



complement of each open set of  $V$  is called a *closed set*. Moreover, a *clopen set* is a set that is both open and closed. Clearly,  $\emptyset$  and  $V$  are clopen with respect to  $V$ . A *neighborhood* of a point  $x$  in  $V$  is a set in  $T_V$  that contains  $x$ . Often, we write  $N_x$  to indicate a neighborhood of  $x$ . A *topological space* is a pair  $(V, T_V)$ . When  $T_V$  is clear from the context, we call  $V$  a topological space. For a practical reason, we implicitly assume that  $V \neq \emptyset$  throughout this paper. For two topological spaces  $(V_1, T_{V_1})$  and  $(V_2, T_{V_2})$ , we say that  $(V_2, T_{V_2})$  is *finer* than  $(V_1, T_{V_1})$  (also  $(V_1, T_{V_1})$  is *coarser* than  $(V_2, T_{V_2})$ ) if both  $V_1 \subseteq V_2$  and  $T_{V_1} \subseteq T_{V_2}$ . In such a case, we write  $(V_1, T_{V_1}) \sqsubseteq (V_2, T_{V_2})$ , or simply  $T_{V_1} \sqsubseteq T_{V_2}$  if  $V_1$  and  $V_2$  are clear from the context. For a topological space  $V$ , a *basis* of its topology  $T_V$  is a set  $B$  of subsets of  $V$  such that every open set in  $T_V$  is expressed as a union of sets of  $B$ . Given two topological spaces  $V$  and  $W$ , the *product topology* (or Tychonoff topology)  $T_{V \times W}$  on the Cartesian product  $V \times W$  is the topology induced by the basis  $\{A \times B \mid A \in T_V, B \in T_W\}$ . We write  $T_V^+$  for  $T_V - \{\emptyset\}$ .

There are two typical topologies on  $V$ : the *trivial topology*  $T_{trivial}(V) = \{\emptyset, V\}$  and the *discrete topology*  $\mathcal{P}(V)$ . Notice that any topology  $T_V$  on  $V$  is located between  $T_{trivial}(V)$  and  $\mathcal{P}(V)$ .

Take a point set  $V$  and consider all possible topologies on  $V$ . Let  $\mathcal{T}(V)$  denote the collection of all topologies  $T_V$  on  $V$ . This set  $\mathcal{T}(V)$  forms a *complete lattice* in which the *meet* and the *join* of a collection  $A$  of topologies on  $V$  correspond to the intersection of all elements in  $A$  and the meet of the collection of all topologies on  $V$  that contain every element of  $A$ .

A map  $B$  from a topological space  $V$  to another  $W$  is said to be *continuous* if, for any  $v \in V$  and any neighborhood  $N$  of  $B(v)$ , there exists a neighborhood  $N'$  of  $v$  satisfying  $B(N') \subseteq N$ , where  $B(N') = \{B(w) \mid w \in N'\}$ . Given a set  $\mathcal{B}$  of continuous maps, the notation  $C_{\mathcal{B}}(V)$  denotes the set of all continuous maps in  $\mathcal{B}$  on  $V$  (i.e., from  $V$  to itself) together with a certain given topology, expressed as  $T_{C_{\mathcal{B}}(V)}$ . When  $\mathcal{B}$  is the set of all continuous maps on  $V$ , we often omit subscript  $\mathcal{B}$  from  $C_{\mathcal{B}}(V)$  and  $T_{C_{\mathcal{B}}(V)}$ .

We are now ready to introduce a fundamental concept of automata base used as a foundation to our model of topological automata. A triplet  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  is called an *automata base* if  $\mathcal{V}$ ,  $\mathcal{B}$ , and  $\mathcal{O}$  satisfy the following three conditions (1)–(3). (1)  $\mathcal{V}$  is a set of topological spaces (which are called *configuration spaces*). (2)  $\mathcal{B}$  is a set of continuous maps (called *transition operators*) from any space  $V$  in  $\mathcal{V}$  to itself for which (i)  $(C_{\mathcal{B}}(V), \circ)$  is a monoid with a multiplication operator  $\circ$ , (ii)  $\circ$  is also a continuous map on  $C_{\mathcal{B}}(V)$ , (iii)  $(V, \bullet)$  is a *left act*<sup>3</sup> over  $C_{\mathcal{B}}(V)$  with  $\circ$ , and (iv)  $\bullet$  must be a continuous map on  $V$ . (3)  $\mathcal{O}$  is a set of *observable pairs*  $(E_{acc}, E_{rej})$ , both of which are clopen sets in a certain space  $V$  in  $\mathcal{V}$  (where  $E_{acc}$  and  $E_{rej}$  are respectively called by an *accepting space* and a *rejecting space*). For our convenience, a pair  $(\mathcal{V}, \mathcal{B})$  is briefly called a *sub-automata base*. For operators  $A, B$  in  $\mathcal{B}$  and a point  $v$  of  $V$ , we simply write  $A(v)$  or even  $Av$  for  $A \bullet v$  and abbreviate  $A \circ B$  as  $AB$ . Note that  $AB(v) = (A \circ B)(v) = A(B(v))$  for every  $v \in V$ .

We say that an automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  is *reasonable* if  $\mathcal{V}$ ,  $\mathcal{B}$ , and  $\mathcal{O}$  are all nonempty. In the rest of this paper, we implicitly assume that  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  is reasonable. Given an operator  $B$ , we say that  $\mathcal{O}$  is *closed under B* if  $(B(E_1), B(E_2)) \in \mathcal{O}$  holds for any  $(E_1, E_2) \in \mathcal{O}$ . Given a

---

<sup>3</sup>A left act satisfies that  $(B_1 \circ B_2) \bullet v = B_1 \bullet (B_2 \bullet v)$  and  $I \bullet v = v$

“property”<sup>4</sup>  $P$  associated with topological spaces, we say that  $\mathcal{V}$  *satisfies*  $P$  if all topological spaces in  $\mathcal{V}$  satisfy  $P$ .

## 2.2. One-Way Deterministic Topological Automata

Formally, let us introduce our model of topological automata, each of which reads input symbols taken from a fixed alphabet, modifies configurations in a deterministic manner, and finally observes the final configurations to determine the acceptance or rejection of the given inputs. This model is called “observe once” because we observe only the final configuration.

Hereafter, let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  denote an arbitrary reasonable automata base. Here, we use two endmarkers  $\clubsuit$  (left endmarker) and  $\$$  (right endmarker) that surround each input string  $x$  as  $\clubsuit x \$$ .

**Basic Model of  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta’s** Assuming an arbitrary input alphabet  $\Sigma$  with  $\clubsuit, \$ \notin \Sigma$ , let us define a basic model of our topological automata. An *1-way (observe-once<sup>5</sup>) deterministic  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -topological automaton* with the endmarkers (succinctly called a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta)  $M$  is a tuple  $(\Sigma, \{\clubsuit, \$\}, V, \{B_\sigma\}_{\sigma \in \check{\Sigma}}, v_0, E_{acc}, E_{rej})$ , where  $\check{\Sigma} = \Sigma \cup \{\clubsuit, \$\}$ ,  $V$  is a configuration space in  $\mathcal{V}$  with a certain topology  $T_V$  on  $V$ ,  $v_0 (\in V)$  is the *initial configuration*, each  $B_\sigma$  is a transition operator in  $\mathcal{B}$  acting on  $V$ , and  $(E_{acc}, E_{rej})$  is an observable pair in  $\mathcal{O}$  for  $V$  satisfying the *exclusion principle*:  $E_{acc}$  and  $E_{rej}$  are disjoint (i.e.,  $E_{acc} \cap E_{rej} = \emptyset$ ). For convenience, let  $E_{non} = V - (E_{acc} \cup E_{rej})$ .

Notice that the use of the endmarkers helps us avoid an introduction of a special transition operator associated with  $\lambda$  (the empty string). *Without endmarker*, by contrast, 1dta’s must read input strings with no help of two endmarkers.

Our definition of 1dta’s is different from the existing ones, as stated in Section 1, in the following ways. Ehrig and Kühnel [3] took compactly generated Hausdorff spaces for our  $\Sigma$  and  $V$ . Jeandel [5] took a metric space for  $V$  and also used a measure, which maps  $V$  to  $\mathbb{R}$  instead of our observable pair  $(E_{acc}, E_{rej})$ .

As another possible formulation of transition operators, we may use a single map  $B : \check{\Sigma} \times V \rightarrow V$  instead of a series  $\{B_\sigma\}_{\sigma \in \check{\Sigma}}$ . Such a map was used by, e.g., Ehrig and Kühnel [3]; however, as pointed out in [3],  $B$  is no longer continuous, and thus we need additional requirements.

**Configurations and Computation.** Let  $x = x_1 x_2 \cdots x_n$  be an input string of length  $n$  in  $\Sigma^*$ . We set  $\tilde{x} = x_0 x_1 \cdots x_n x_{n+1}$  to be an *endmarked input string*, which includes  $x_0 = \clubsuit$  (left endmarker) and  $x_{n+1} = \$$  (right endmarker).

<sup>4</sup>This informal term “property” is used in a general sense throughout this paper, not limited to “topological properties,” which usually means the “invariance under homeomorphisms.”

<sup>5</sup>It is possible to consider an *observe-many* model of 1dta in which, at each step, the 1dta checks if, at each step, the current configuration falls into  $E_{acc} \cup E_{rej}$ .

The machine  $M$  works as follows. A *configuration* of  $M$  on  $x$  is a point of  $V$ . A configuration in  $E_{acc}$  (resp.,  $E_{rej}$ ) is called an *accepting configuration* (resp., a *rejecting configuration*). Both accepting and rejecting configurations are simply called *halting configurations*. We begin with the initial configuration  $v_0 \in V$ , which is the *0th configuration* of  $M$  on  $x$ . In the first step, we apply  $B_{\dagger}$  and obtain  $v_1 = B_{\dagger}(v_0)$ . For an index  $i \in [n]$ , we assume that  $v_i$  is the  $i$ th configuration of  $M$  on  $x$ . In step  $i + 1$  ( $0 \leq i \leq n$ ), the  $i + 1$ st configuration  $v_{i+1}$  is obtained from  $v_i$  by applying the operator  $B_{x_i}$  corresponding to  $x_i$ ; that is,  $v_{i+1} = B_{x_i}(v_i)$ . For any series  $\sigma_1, \sigma_2, \dots, \sigma_{j-1}, \sigma_j \in \Sigma$ , we abbreviate as  $B_{\sigma_1\sigma_2\dots\sigma_j}$  the multiplication  $B_{\sigma_j}B_{\sigma_{j-1}} \cdots B_{\sigma_2}B_{\sigma_1}$ . Notice that, since  $\mathcal{B}$  is a monoid with the multiplication,  $B_{\sigma_1\sigma_2\dots\sigma_j}$  also belongs to  $\mathcal{B}$ . The final configuration  $v_{n+2}$  is obtained from  $v_{n+1}$  by  $v_{n+2} = B_{\S}(v_{n+1})$  and it coincides with  $B_{\dagger x \S}(v_0)$ . The obtained series of configurations,  $(v_0, v_1, \dots, v_{n+2})$ , is called a *computation* of  $M$  on the input  $x$ . When a 1dta has no endmarker, by contrast, a computation  $(v_0, v_1, \dots, v_{n+1})$  is simply generated by  $v_i = B_{x_i}(v_{i-1})$  for each  $i \in [n]$ , and the final configuration  $v_{n+1}$  coincides with  $B_x(v_0)$ .

**Acceptance and Rejection.** Finally, we determine the acceptance and the rejection of input strings by checking whether the final configuration  $v_{n+2}$  falls into  $E_{acc}$  and  $E_{rej}$ , respectively. We say that  $M$  *accepts* (resp., *rejects*)  $x$  if  $v_{n+2} \in E_{acc}$  (resp.,  $v_{n+2} \in E_{rej}$ ). We say that  $M$  *recognizes*  $L$  if, for every string  $x \in \Sigma^*$ , the following two conditions are met: (1) if  $x \in L$ , then  $M$  accepts  $x$  and (2) if  $x \notin L$ , then  $M$  rejects  $x$ . We use the notation  $L(M)$  to denote the language that is recognized by  $M$ .

We define  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA to be the family of all languages, each of which is defined over a certain alphabet  $\Sigma$  and is recognized by a certain  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta.

Two 1dta's  $M_1$  and  $M_2$  having the common  $\Sigma$  and  $V$  are said to be (*computationally*) *equivalent* if  $L(M_1) = L(M_2)$ .

### 2.3. Conventional Finite Automata are 1dta's

Our topological-automata framework naturally extends the existing 1-way finite automata of various types. To see this fact, let us demonstrate that typical models of 1-way finite automata can be nicely fit into our framework.

As concrete examples, we here consider only the following four types of finite automata.

**(i) Deterministic Finite Automata.** A *1-way deterministic finite automaton* (or a 1dfa) with two endmarkers can be viewed as a special case of  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta when  $\mathcal{V}$  equals  $\{[k] \mid k \in \mathbb{N}^+\}$  with the discrete topology,  $\mathcal{B}$  contains all maps from  $[k]$  to  $[k]$  for each  $k \in \mathbb{N}^+$ , and  $\mathcal{O}$  contains all nonempty partitions  $(E_{acc}, E_{rej})$  of  $[k]$  for each  $k \in \mathbb{N}^+$ . Languages recognized by 1dfa's are called *regular* and REG denotes the set of all regular languages.

**(ii) Probabilistic Finite Automata [9].** A *stochastic matrix* is a nonnegative-real matrix in which every column<sup>6</sup> sums up to exactly 1. A bounded-error *1-way probabilistic finite automaton* (or 1pfa) is a special case of  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta, where  $\mathcal{V} = \{[0, 1]^k \mid k \in \mathbb{N}^+\}$  (in which each point of  $[0, 1]^k$  is expressed as a column vector),  $\mathcal{B}$  is composed of all  $k \times k$  stochastic matrices, and  $\mathcal{O}$  is the set of all pairs  $(E_{acc}, E_{rej})$ , each of which is defined as the inverse images of projections onto unit basis vectors in  $[0, 1]^k$ . The notation 1BPFA denotes the set of all languages recognized by 1pfa's with bounded-error probability. When we consider unbounded-error probability, we write SL to denote the set of all *stochastic languages* (i.e., languages that are recognized by 1pfa's with unbounded-error probability). It is known that  $1BPFA = \text{REG}$  [9] and  $\text{REG} \subsetneq \text{SL}$  since  $L_{<} = \{a^m b^n \mid m, n \in \mathbb{N}, m < n\}$  is in SL.

**(iii) Measure-Many Quantum Finite Automata [6].** A bounded-error *measure-many 1-way quantum finite automaton* (or 1qfa) is a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta, where  $\mathcal{V}$  contains all sets  $V$  of the form  $(\mathbb{C}^{\leq 1})^k \otimes [0, 1] \otimes [0, 1]$  and  $\mathcal{B}$  consists of all maps  $T$  defined in [11] as

$$T(v, \gamma_1, \gamma_2) = \left( \Pi_{non} Bv, \text{sgn}(\gamma_1) \sqrt{\gamma_1^2 + \|\Pi_{acc} Bv\|_2^2}, \text{sgn}(\gamma_2) \sqrt{\gamma_2^2 + \|\Pi_{rej} Bv\|_2^2} \right), \quad (1)$$

where  $\text{sgn}(\gamma) = +1$  if  $\gamma \geq 0$  and  $-1$  if  $\gamma < 0$ , for a certain  $k \times k$  unitary matrix  $B$  and 3 projections  $\Pi_{acc}$ ,  $\Pi_{rej}$ , and  $\Pi_{non}$  onto different unit basis vectors. Concerning bounded-error 1qfa's, we set  $E_{acc} = \{(v, \gamma_1, \gamma_2) \in V \mid \gamma_1 \geq 1 - \varepsilon\}$  and  $E_{rej} = \{(v, \gamma_1, \gamma_2) \in V \mid \gamma_2 \geq 1 - \varepsilon\}$  for each constant  $\varepsilon \in [0, 1/2)$ . Let  $\mathcal{O}$  express the set of all such pairs  $(E_{acc}, E_{rej})$ . For basic properties of  $T$ , see [11, Appendix]. We write MM-1QFA to denote the collection of all languages recognized by 1qfa's with bounded-error probability. It is known that  $\text{MM-1QFA} \subsetneq \text{REG}$  [6].

**(iv) Deterministic Pushdown Automata.** A *1-way deterministic pushdown automaton* (or a 1dpda)  $M$  can be seen as a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta when  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  satisfies the following properties. Let  $\mathcal{V} = \{[k] \times \perp \Gamma^* \mid k \in \mathbb{N}^+, \Gamma: \text{alphabet}\}$ , where  $\perp$  is a distinguished bottom marker not in  $\Gamma$ . Let  $\mathcal{B}$  be composed of all maps of the form  $B(q, \perp z) = (\mu_1(q, z_n), \perp z_1 z_2 \cdots z_{n-1} \mu_2(q, z_n))$  for two functions  $\mu_1 : [k] \times \Gamma \rightarrow [k]$  and  $\mu_2 : [k] \times \Gamma \rightarrow \Gamma^{\leq l}$ , where  $z = z_1 z_2 \cdots z_n$  and  $l \in \mathbb{N}^+$ . Intuitively,  $B$  simulates a series of moves in which  $M$  reads one symbol and then makes a single non- $\lambda$ -move followed by a certain number of  $\lambda$ -moves. Let  $\mathcal{O}$  consist of all pairs  $(E_{acc}, E_{rej})$  with  $E_{acc} = Q_1 \times \perp \Gamma^*$  and  $E_{rej} = Q_2 \times \perp \Gamma^*$ , where  $Q_1$  and  $Q_2$  are partitions of  $[k]$ . We write DCFL for the class of these languages.

### 3. Basic Properties of $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta's

We have introduced the model of  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta's in Section 2.2. We will begin with exploring basic properties of those  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta's and their language families  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA.

<sup>6</sup>Unlike the standard definition, in accordance with our topological automata, we apply each stochastic matrix to column vectors from the left, not from the right in the early literature.

### 3.1. Elimination of Endmarkers

In many cases, it is possible to eliminate endmarkers from a 1dta  $M$  without changing the languages recognized by  $M$ . A simple way to do so is to modify the initial configuration, say,  $v_0$  of  $M$  to a new initial configuration  $B_{\dagger}(v_0)$  using an operator  $B_{\dagger}$  of  $M$ . Even if we stick to the same  $v_0$ , a slight modification of all operators  $B_{\sigma}$  of  $M$  provides the same effect as shown in the following lemma.

We say that a set  $\mathcal{B}$  of operators is *continuously invertible* if every operator  $B$  in  $\mathcal{B}$  is invertible and its inverse  $B^{-1}$  is in  $\mathcal{B}$  and continuous.

**Lemma 3.1** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be any reasonable automata base. Assume that  $\mathcal{B}$  is continuously invertible. For every  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M = (\Sigma, \{\dagger, \$\}, V, \{B_{\sigma}\}_{\sigma \in \Sigma}, v_0, E_{acc}, E_{rej})$ , there exists another equivalent  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $N$  with the same  $\Sigma, V, v_0, E_{acc}$ , and  $E_{rej}$  but no left-endmarker.*

*Proof.* We define a new set of operators  $B'_{\sigma}$  for each symbol  $\sigma \in \Sigma \cup \{\dagger\}$ . Define  $B'_{\sigma} = B_{\dagger}^{-1} B_{\sigma} B_{\dagger}$  for any  $\sigma \in \Sigma$  and let  $B'_{\dagger} = B_{\dagger} B_{\dagger}$ . The desired  $N$  reads an input  $x\dagger$  and behaves exactly as  $M$  does by applying  $\{B_{\sigma}\}_{\sigma \in \Sigma \cup \{\dagger\}}$ .  $\square$

We can eliminate  $\dagger$  as well by slightly changing observable pairs of  $M$ .

**Lemma 3.2** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be any reasonable automata base. Assume that  $\mathcal{B}$  is continuously invertible and that  $\mathcal{O}$  is closed under all operators in  $\mathcal{B}$ . For every  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M$ , there exists its equivalent  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $N$  with the same  $\Sigma, V, B_{\sigma}$ , and  $v_0$  but no right-endmarker.*

*Proof.* We define a new observable pair  $(E'_{acc}, E'_{rej})$  of  $N$  by setting  $E'_{acc} = \{v \in V \mid B_{\dagger}(v) \in E_{acc}\}$  and  $E'_{rej} = \{v \in V \mid B_{\dagger}(v) \in E_{rej}\}$ . Clearly,  $E'_{acc}$  and  $E'_{rej}$  are disjoint because so are  $E_{acc}$  and  $E_{rej}$ . Since  $B_{\dagger}$  is invertible and  $B_{\dagger}^{-1}$  is continuous,  $E'_{acc}$  and  $E'_{rej}$  are written as  $\{B_{\dagger}^{-1}(v) \mid v \in E_{acc}\}$  and  $\{B_{\dagger}^{-1}(v) \mid v \in E_{rej}\}$ , respectively. Since  $B_{\dagger}^{-1}$  is in  $\mathcal{B}$  and  $\mathcal{O}$  is closed under  $B_{\dagger}^{-1}$ , it follows that  $(E'_{acc}, E'_{rej}) \in \mathcal{O}$ .  $\square$

### 3.2. Closure Properties

We discuss a few closure properties of  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA. We say that  $\mathcal{O}$  is *symmetric* if, for any pair  $(A, B) \in \mathcal{O}$ ,  $(B, A)$  also belongs to  $\mathcal{O}$ . An automata base  $(\mathcal{V}, \mathcal{B})$  is said to be *closed under product* if, for any  $V_1, V_2 \in \mathcal{V}$  and any  $B_1, B_2 \in \mathcal{B}$ , the following holds. We first define “products” of them by setting  $V = V_1 \times V_2$  and  $B = B_1 \times B_2$  and by taking the associated product topology  $T_V = T_{V_1 \times V_2}$ . We also require these  $V$  and  $B$  to be in  $\mathcal{V}$  and  $\mathcal{O}$ , respectively. Next, we say that  $(\mathcal{V}, \mathcal{O})$  is *closed under left-union product* if  $(E_{acc}, E_{rej}) \in \mathcal{O}$ , where  $E_{acc} = (V_1 \times E_{2,acc}) \cup (E_{1,acc} \times V_2)$  and  $E_{rej} = E_{1,rej} \times E_{2,rej}$ . Similarly, we can define the notion of the *closure under right-union product* by swapping the roles of two subscripts “acc” and “rej” in the above definition.

**Lemma 3.3** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be any reasonable automata base.*

1. *If  $\mathcal{O}$  is symmetric, then  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA is closed under complementation.*
2. *If  $(\mathcal{V}, \mathcal{B})$  is closed under product and  $(\mathcal{V}, \mathcal{O})$  is closed under left-union product, then  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA is closed under union.*
3. *If  $(\mathcal{V}, \mathcal{B})$  is closed under product and  $(\mathcal{V}, \mathcal{O})$  is closed under right-union product, then  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA is closed under intersection.*

Ambainis et al. [1] proved that MM-1QFA is not closed under union. Hence, the premise of the above lemma is needed.

*Proof.* (1) The closure property of  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA under complementation can be obtained simply by exchanging between  $E_{acc}$  and  $E_{rej}$  since  $\mathcal{O}$  is symmetric.

(2) For each  $i \in \{1, 2\}$ , we take a language  $L_i$  over  $\Sigma$  recognized by a certain  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M_i = (\Sigma, \{\clubsuit, \$\}, V_i, \{B_{i,\sigma}\}_{\sigma \in \check{\Sigma}}, v_{i,0}, E_{i,acc}, E_{i,rej})$ . Let  $L = L_1 \cup L_2$ . For  $(V_1, B_{1,\sigma})$  and  $(V_2, B_{2,\sigma})$ , we consider  $(V, B_\sigma)$  defined by  $V = V_1 \times V_2$  and  $B_\sigma = B_{1,\sigma} \times B_{2,\sigma}$ . Moreover, set  $v_0 = (v_{1,0}, v_{2,0})$ ,  $E_{acc} = (V_1 \times E_{2,acc}) \cup (E_{1,acc} \times V_2)$ , and  $E_{rej} = E_{1,rej} \times E_{2,rej}$ . For any initial segment  $z$  of  $x\$$ ,  $B_{\clubsuit z}(v_0) = (B_{1,\clubsuit z}(v_{1,0}), B_{2,\clubsuit z}(v_{2,0}))$ . It thus follows that (i)  $B_{\clubsuit x\$}(v_0) \in E_{acc}$  if and only if either  $B_{1,\clubsuit x\$}(v_{1,0}) \in E_{1,acc}$  or  $B_{2,\clubsuit x\$}(v_{2,0}) \in E_{2,acc}$  and (ii)  $B_{\clubsuit x\$}(v_0) \in E_{rej}$  if and only if both  $B_{1,\clubsuit x\$}(v_{1,0}) \in E_{1,rej}$  and  $B_{2,\clubsuit x\$}(v_{2,0}) \in E_{2,rej}$ . Finally, we define the desired  $N$  as  $(\Sigma, \{\clubsuit, \$\}, V, \{B_\sigma\}_{\sigma \in \check{\Sigma}}, v_0, E_{acc}, E_{rej})$ .

(3) Similar to (1) in principle, but we need to exchange the roles of “acc” and “rej”.  $\square$

### 3.3. Computational Power Endowed by the Trivial and Discrete Topologies

We briefly discuss the language recognition power endowed to 1dta’s by the trivial topology as well as the discrete topology. In fact, while the trivial topology makes 1dta’s recognize only trivial languages, the discrete topology makes them powerful enough to recognize all languages. This latter fact, in particular, assures us to be able to characterize any language family by an appropriate choice of topologies for 1dfa’s.

**Proposition 3.4** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be a reasonable automata base has the trivial topology  $T_{trivial}(V)$  for each  $V \in \mathcal{V}$ . For any  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M$  with an alphabet  $\Sigma$ ,  $L(M)$  is either  $\emptyset$  or  $\Sigma^*$ .*

*Proof.* Given an automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  in the lemma, let us consider any  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M = (\Sigma, \{\clubsuit, \$\}, V, \{B_\sigma\}_{\sigma \in \check{\Sigma}}, v_0, E_{acc}, E_{rej})$ . Since  $E_{acc}$  is clopen with respect to  $T_{trivial}(V)$ , it must be either  $\emptyset$  or  $V$ . The same holds for  $E_{rej}$ . Hence,  $M$  either accepts all strings or rejects all strings. Thus,  $L(M)$  is either  $\Sigma^*$  or  $\emptyset$ .  $\square$

In contrast, the discrete topology provides underlying automata with enormous computational power, as shown below.

**Proposition 3.5** *There is a reasonable automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  with the discrete topology for each  $V \in \mathcal{V}$  such that, for any language  $L$ , there is a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta that recognizes  $L$ . This is true for the 1dta model with or without endmarkers.*

### 3.4. Slender Topological Automata

Let us consider a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M = (\Sigma, \{\dagger, \$\}, V, \{B_\sigma\}_{\sigma \in \Sigma}, v_0, E_{acc}, E_{rej})$  for a given automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ . There is a case where configurations generated (or visited) by  $M$  starting with  $v_0$  may not cover all points in  $V$ . In this case,  $V$  itself does not seem to characterize the actual behavior of  $M$ , because the points that cannot be visited by  $M$  may satisfy a completely different property from the rest of the points. Therefore, when we discuss the true power of topologies used to define 1dta's, it must be desirable to leave out all points that are not visited by  $M$  and to be focused on the set of all points that  $M$  can reach. This conclusion makes us introduce a new notion of *slender 1dta's*, which can visit all points in  $V$ . Formally, we say that the 1dta  $M$  is *slender* if, for every point  $v \in V$ , there exists a string  $x \in \Sigma^*$  for which either  $B_{\dagger x}(v_0) = v$  or  $B_{\dagger x \$}(v_0) = v$  holds.

We show how to build, for any given 1dta, its equivalent slender 1dta. The *normalization* of  $M$  is defined to be a  $(\hat{\mathcal{V}}_M, \hat{\mathcal{B}}_M, \hat{\mathcal{O}}_M)$ -1dta, denoted by  $M_{norm}$ , which is obtained from  $M$  in the following way. First, we define  $\mathcal{B}'$  to be the union of  $\{B_\dagger, B_\$\}$  and the closure of  $\{B_x \mid x \in \Sigma^*\}$  under functional composition. We define  $V_M = \{v_0, B_\dagger(v_0), BB_\dagger(v_0), B_\$BB_\dagger(v_0) \mid B \in \mathcal{B}' - \{B_\dagger, B_\$\}\}$  with its *subspace topology*  $T_{V_M}$  induced by  $T_V$  (namely,  $T_{V_M} = \{A \cap V_M \mid A \in T_V\}$ ). Notice that  $(V_M, T_{V_M})$  and  $(V, T_V)$  may be quite different in nature. We further set  $\hat{\mathcal{V}}_M = \{V_M\}$ . To define  $\hat{\mathcal{B}}_M$ , we need to restrict the domain of each operator  $B \in \mathcal{B}'$  onto  $V_M$ . We write the obtained map as  $\hat{B}$ . The desired  $\hat{\mathcal{B}}_M$  is set to be the family of all operators  $\hat{B}$  induced from operators  $B$  in  $\mathcal{B}'$ . Finally, we set  $\hat{\mathcal{O}}_M$  to be  $\{(E_{acc} \cap V_M, E_{rej} \cap V_M)\}$ . It thus follows that all points of  $V_M$  are visited by  $M$  while reading certain input strings over the alphabet  $\Sigma$ .

**Lemma 3.6** (1)  $M_{norm}$  is slender. (2)  $M_{norm}$  is computationally equivalent to  $M$ .

*Proof.* (1) We first claim that  $M_{norm}$  is slender. Let  $v \in V_M$ . We then obtain  $v = v_0$ ,  $v = BB_\dagger(v_0)$ , or  $v = B_\$BB_\dagger(v_0)$  for a certain  $B \in \mathcal{B}' - \{B_\dagger, B_\$\}$ . By the definition of  $\mathcal{B}'$ , there is an  $x \in \Sigma^*$  such that  $B = B_x$  for a certain  $x \in \Sigma^*$ . We then conclude that  $v = v_0$ ,  $v = B_{\dagger x}(v_0)$ , or  $v = B_{\dagger x \$}(v_0)$ .

(2) Next, we want to show by induction on  $n \in \mathbb{N}$  that  $\hat{B}_{\dagger x}(v_0) = B_{\dagger x}(v_0)$  and  $\hat{B}_{\dagger x \$}(v_0) = B_{\dagger x \$}(v_0)$  for any  $x \in \Sigma^n$ . This yields the computational equivalence between  $M_{norm}$  and  $M$ . Let  $x \in \Sigma^n$  and consider  $B_{\dagger x}(v_0)$ . Note that  $v_0 \in V_M$ . Assume that  $\hat{B}_{\dagger x}(v_0) = B_{\dagger x}(v_0) \in V_M$ . Take any  $\sigma \in \Sigma \cup \{\$\}$ . Since  $\hat{B}_\sigma(w) = B_\sigma(w)$  for any  $w \in V_M$ , it follows by the induction hypothesis that  $\hat{B}_{\dagger x \sigma}(v_0) = \hat{B}_\sigma(\hat{B}_{\dagger x}(v_0)) = \hat{B}_\sigma(B_{\dagger x}(v_0)) = B_\sigma(B_{\dagger x}(v_0)) = B_{\dagger x \sigma}(v_0)$ . In particular, we obtain  $\hat{B}_{\dagger x \$}(v_0) = B_{\dagger x \$}(v_0)$ .  $\square$

Notice that, for any slender 1dta, since  $\Sigma^*$  is countable, its associated topological space is also countable.

**Lemma 3.7** *If  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M = (\Sigma, \{\$, \#\}, V, \{B_\sigma\}_{\sigma \in \Sigma}, v_0, E_{acc}, E_{rej})$  is slender and  $\mathcal{B}$  contains  $\mathcal{B}'$  (which is defined earlier), then  $M_{norm}$  is also a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta.*

*Proof.* Since  $M$  is slender, we obtain  $V_M = V$  and thus  $T_{V_M} = T_V$ . For any  $\hat{B} \in \hat{\mathcal{B}}_M$ , since  $\mathcal{B}$  contains  $\mathcal{B}'$ , there is another operator  $B' \in \mathcal{B}'$  such that  $\hat{B}$  equals  $B'$  restricted to  $V_M$ . Since  $V_M = V$ , we obtain  $\hat{B} = B'$ . Thus,  $\hat{\mathcal{O}}_M = \{(E_{acc}, E_{rej})\}$ .  $\square$

## 4. Computational Strengths of Properties on Topological Spaces

Since topological spaces give fundamental grounds to topological automata, we want to compare the strengths of two different “properties” of the topological spaces by comparing the computational power of the corresponding topological automata. For instance, the *Hausdorff separation axiom* is one of those properties.

For our purpose, we further restrict our attention on slender 1dta’s. Given a property  $P$ , we say that an automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  *meets*  $P$  if every slender  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M$  satisfies  $P$ . Let  $P_1$  and  $P_2$  be two properties on point sets. We say that  $P_2$  is *at least as computationally strong as*  $P_1$ , denoted by  $P_1 \leq_{\text{comp}} P_2$ , if, for any reasonable automata base  $(\mathcal{V}_1, \mathcal{B}_1, \mathcal{O}_1)$  that meets  $P_1$ , there exists another reasonable automata base  $(\mathcal{V}_2, \mathcal{B}_2, \mathcal{O}_2)$  meeting  $P_2$  such that every slender  $(\mathcal{V}_1, \mathcal{B}_1, \mathcal{O}_1)$ -1dta has a computationally equivalent slender  $(\mathcal{V}_2, \mathcal{B}_2, \mathcal{O}_2)$ -1dta. Notice that  $P_1$  is always at least as computationally strong as  $P_1$  itself. Moreover,  $P_2$  is said to be *computationally stronger than*  $P_1$  if  $P_1 \leq_{\text{comp}} P_2$  and  $P_2 \not\leq_{\text{comp}} P_1$ .

For two properties  $P_1$  and  $P_2$ , we say that  $P_2$  *supersedes*  $P_1$ , denoted by  $P_1 \sqsubseteq P_2$ , exactly when, for every automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ , if it meets  $P_1$ , then it also meets  $P_2$ . The following lemma is trivial.

**Lemma 4.1** *For two properties  $P_1$  and  $P_2$ , if  $P_1 \sqsubseteq P_2$ , then  $P_1 \leq_{\text{comp}} P_2$ .*

In what follows, we present two results concerning topological indistinguishability. Let  $(V, T_V)$  be any topological space. Two points  $x$  and  $y$  of  $V$  are *topologically distinguishable* if there exists an open set  $N \in T_V$  such that either (i)  $x \in N$  and  $y \notin N$  or (ii)  $x \notin N$  and  $y \in N$ . Otherwise, they are *topologically indistinguishable*. The *Kolmogorov separation axiom* (or simply, the *Kolmogorov condition*) dictates that any pair of distinct points of  $V$  are topologically distinguishable. Any space that satisfies the Kolmogorov condition is called a *Kolmogorov space*. For example, let us consider  $(V, T_V)$  with  $V = \{1, 2, 3\}$  and  $T_V = \{\emptyset, \{1, 2\}, \{2\}, \{2, 3\}, \{1, 2, 3\}\}$ . This  $(V, T_V)$  is a Kolmogorov space but it does not have the discrete topology on  $V$ . While the discrete topology always satisfies the Kolmogorov condition, the trivial topology violates the same condition.

The next proposition shows a clear difference between the trivial topology and any topology that violates the Kolmogorov condition.



**Theorem 4.2** *There is a topology not satisfying the Kolmogorov separation axiom and it is computationally stronger than the trivial topology.*

*Proof.* Consider the language  $ZERO = \{0^n \mid n \in \mathbb{N}\}$  over the binary alphabet  $\Sigma = \{0, 1\}$ . By Proposition 3.4,  $ZERO$  cannot be recognized by any  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta's having the binary alphabet for any reasonable automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  with every  $V$  in  $\mathcal{V}$  having the trivial topology. Here, we want to prove the existence of an automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  and a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M = (\Sigma, \{\clubsuit, \$\}, V, \{B_\sigma\}_{\sigma \in \check{\Sigma}}, v_0, E_{acc}, E_{rej})$  satisfying that (i)  $\mathcal{V}$  consists of finite topological spaces violating the Kolmogorov condition and (ii)  $M$  recognizes  $ZERO$ . Since the trivial topology provides only  $\{\emptyset, \Sigma^*\}$ , the topology on  $\mathcal{V}$  is computationally stronger.

Let us define the desired 1dta  $M$  as follows. Let  $V = \{0, 1, 2\}$  and  $T_V = \{\emptyset, V, \{0\}, \{1, 2\}\}$  so that  $(V, T_V)$  cannot satisfy the Kolmogorov condition. We define  $B_\clubsuit = B_\$ = I$  and  $B_0(n) = n$  and  $B_1(n) = \min\{n+1, 2\}$  for any  $n \in V$ . Clearly,  $B_\sigma$  is continuous for each  $\sigma \in \check{\Sigma}$ . Moreover, we set  $v_0 = 0$ ,  $E_{acc} = \{0\}$ , and  $E_{rej} = \{1, 2\}$ . It is immediate that  $M$  accepts all strings of the form  $0^n$  for  $n \in \mathbb{N}$  and rejects all the strings containing 1. Therefore,  $M$  recognizes  $ZERO$ . Finally, we set  $\mathcal{V} = \{V\}$ ,  $\mathcal{B} = \{B_\sigma \mid \sigma \in \check{\Sigma}\}$ , and  $\mathcal{O} = \{(E_{acc}, E_{rej})\}$ .  $\square$

In certain cases, we cannot reduce the size of  $E_{acc}$  and  $E_{rej}$  down to one.

**Lemma 4.3** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be any reasonable automata base. Consider any slender  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M$  with  $V$ ,  $E_{acc}$ , and  $E_{rej}$ . If  $V$  satisfies the Kolmogorov condition but does not have the discrete topology, then neither  $E_{acc}$  nor  $E_{rej}$  can be a singleton.*

In the proof of Theorem 4.2, we have used *finite topologies*, each of which is composed of a finite number of open sets. We argue that any finite topology provides topological automata with no more recognition power than 1dfa's.

**Proposition 4.4** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be any reasonable automata base with finite topologies. Any language recognized by a certain  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta is a regular language.*

*Proof.* Let  $M = (\Sigma, \{\clubsuit, \$\}, V, \{B_\sigma\}_{\sigma \in \check{\Sigma}}, v_0, E_{acc}, E_{rej})$  be any  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta with a finite topology  $T_V$ . We want to convert  $M$  into another equivalent 1dfa  $N$ . For any two points  $v, w \in V$ , we define a binary relation  $\equiv$  as:  $v \equiv w$  if and only if  $v$  and  $w$  are topologically indistinguishable. We first show that this relation  $\equiv$  is an equivalence relation on  $V$ . Clearly,  $v \equiv v$  holds. If  $v \equiv w$ , then  $w \equiv v$  holds. Assume that  $v \equiv w$  and  $w \equiv z$ . If  $v \not\equiv z$ , then there is an open set  $A \in T_V$  such that either  $(v \in A \text{ and } z \notin A)$  or  $(v \notin A \text{ and } z \in A)$ . Without loss of generality, we assume that  $v \in A$  and  $z \notin A$ . Since  $v \equiv w$ , we obtain  $w \in A$ . This means that  $z \not\equiv w$ , a contradiction.

Next, we consider a set  $V/\equiv$  of all equivalence classes. Given a point  $v \in V$ , let  $[v] = \{w \in V \mid v \equiv w\}$ . It follows that  $V/\equiv = \{[v] \mid v \in V\}$ . We claim that  $|V/\equiv|$  is finite. If  $V/\equiv$  is an infinite set, then we can take an infinite subset  $S$  of  $V$  such that any two distinct points are topologically distinguishable. There must be an infinite number of open sets in  $T_V$ . This contradicts the finiteness of  $T_V$ . Thus,  $|V/\equiv|$  must be finite. Let  $m = |V/\equiv|$ .

We choose  $v_0, v_1, \dots, v_{m-1} \in V$  such that  $[v_i] \neq [v_j]$  for any distinct pair  $i, j \in [0, m-1]_{\mathbb{Z}}$ . We define a new ldfa  $N = (Q, \Sigma, \{\clubsuit, \$\}, \delta, v_0, Q_{acc}, Q_{rej})$  as follows. Let  $Q = \{v_0, v_1, \dots, v_{m-1}\}$ . The transition function  $\delta : Q \times \Sigma \rightarrow Q$  is defined as:  $\delta(v_i, \sigma) = v_j$  if and only if there are points  $w_i, w_j \in V$  such that  $[v_i] = [w_i]$ ,  $[v_j] = [w_j]$ , and  $B_\sigma(w_i) = w_j$ . We set  $Q_{acc} = \{v_i \mid [v_i] \cap E_{acc} \neq \emptyset\}$  and  $Q_{rej} = \{v_i \mid [v_i] \cap E_{rej} \neq \emptyset\}$ .

For any initial segment  $z$  of  $x\$$ , we claim that  $B_{\clubsuit z}(v_0) \in E_{acc}$  if and only if  $\delta^*(v_0, \clubsuit z) \in Q_{acc}$ , where  $\delta^*(q, w)$  denotes an inner state obtained just after reading  $w$ , starting in state  $q$ .

Next, we claim that  $[v] = [w]$  implies  $[B_\sigma(v)] = [B_\sigma(w)]$ . Assume that  $[B_\sigma(v)] \neq [B_\sigma(w)]$ . Take a neighborhood  $N$  of  $B_\sigma(v)$  satisfying  $B_\sigma(w) \notin N$ . Choose another neighborhood  $N'$  of  $v$  such that  $B_\sigma(N') \subseteq N$ . Since  $w \in N'$ , we obtain a contradiction.

Finally, we claim that there is no index  $i \in [0, m-1]_{\mathbb{Z}}$  such that  $[v_i] \cap E_{acc} \neq \emptyset$  and  $[v_i] \cap E_{rej} \neq \emptyset$  because, otherwise, there are two distinct points  $w_1, w_2 \in [v_i]$  satisfying that  $w_1 \in E_{acc}$  and  $w_2 \in E_{rej}$ , and thus  $w_1 \neq w_2$ , a contradiction.

Since  $N$  can simulate  $M$ , we conclude that  $L(M) = L(N)$ . □

## 5. Compactness, Equicontinuity, and Regularity

In general topology, the notion of compactness of topological spaces plays an important role. This notion also makes a significant effect on the computational complexity of ldfa's. For a metric space  $V$  and a topological automaton  $M$ , Jeandel claimed in [5, Theorem 3] that, using our notation, the compactness of  $V_M$  and  $\hat{\mathcal{B}}_M$  yields the regularity of the language recognized by  $M$ . In contrast, our topological automata use arbitrary topologies, not limited to metric spaces; therefore, we need to show a more general statement, which gives a necessary and sufficient condition for the regularity of languages.

With an appropriate index set  $I$ , a collection  $\{W_i\}_{i \in I}$  of open subsets of  $V$  is called a *covering* if  $V \subseteq \bigcup_{i \in I} W_i$ . A *subcovering* of  $\{W_i\}_{i \in I}$  is a collection  $\{W_j\}_{j \in J}$  for a certain subset  $J$  of  $I$  satisfying that  $V \subseteq \bigcup_{j \in J} W_j$ . When  $J$  is a finite set, the subcovering is said to be *finite*. A topological space  $(V, T_V)$  is called *compact* if every open covering of  $V$  has a finite subcovering. Recall that the set  $C_{\mathcal{B}}(V)$  is assumed to have a topology, denoted by  $T_{C_{\mathcal{B}}(V)}$ . We say that a sub-automata base  $(\mathcal{V}, \mathcal{B})$  is *compact* if, for any  $V$  in  $\mathcal{V}$ ,  $V$  is compact and  $C_{\mathcal{B}}(V)$  is also compact.

A *uniform structure* on  $V$  is a collection  $\Phi$  of subsets of  $V \times V$  satisfying that, for any  $U \in \Phi$  and any  $W \subseteq V \times V$ , (i)  $\{(v, v) \mid v \in V\} \subseteq U$ , (ii)  $U \subseteq W$  implies  $W \in \Phi$ , (iii)  $W \in \Phi$  implies  $U \cap W \in \Phi$ , (iv) there exists a set  $W' \in \Phi$  for which  $W' \circ W' \subseteq U$ , and (v)  $U^{-1} \in \Phi$ , where  $W' \circ W' = \{(v, w) \mid \exists z \in V [(v, z), (z, w) \in W']\}$  and  $U^{-1} = \{(w, v) \mid (v, w) \in U\}$ . For the set  $C(V)$  of all continuous maps on  $V$ , a subset  $F$  of  $C(V)$  is *uniformly topologically equicontinuous* if, for any element  $A$  of a uniform structure on  $V$ , the set  $\{(u, v) \in V^2 \mid \forall f \in F [(f(u), f(v)) \in A]\}$  is also an element of the uniform structure. A uniform structure  $\Phi$  on  $V$  is said to be

compatible with a given topology  $T_V$  if  $A \in T_V$  holds for every set  $A \subseteq V$  exactly when, for any  $x \in A$ , there is a set  $U \in \Phi$  satisfying that  $U[x] \subseteq A$ , where  $U[x] = \{y \in V \mid (x, y) \in U\}$ . A topological space  $(V, T_V)$  is *uniformizable* if there exists a uniform structure compatible with the topology  $T_V$ . We say that a sub-automata base  $(\mathcal{V}, \mathcal{B})$  is *uniformly topologically equicontinuous* if, for any  $V \in \mathcal{V}$ ,  $C_{\mathcal{B}}(V)$  is uniformly topologically equicontinuous.

Next, we show one of our main theorems, which gives a necessary and sufficient condition on  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ , ensuring that  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA = REG.

**Theorem 5.1** *For any language  $L$ , the following two statements are logically equivalent. (1)  $L$  is regular. (2) There is a reasonable automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  such that every element in  $\mathcal{V}$  is uniformizable,  $(\mathcal{V}, \mathcal{B})$  is compact and uniformly topologically equicontinuous, and  $L$  is recognized by a certain  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta.*

To prove this theorem, we need the following lemma.

**Lemma 5.2** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be any reasonable automata base such that  $\mathcal{V}$ 's elements are uniformizable. If  $(\mathcal{V}, \mathcal{B})$  is compact and uniformly topologically equicontinuous, then  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA  $\subseteq$  REG.*

*Proof.* Assume that  $(\mathcal{V}, \mathcal{B})$  is compact and uniformly topologically equicontinuous and that  $\mathcal{V}$ 's elements are uniformizable. Let  $L$  be any language in  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1DTA over an alphabet  $\Sigma$ . Take any  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta  $M$  that recognizes  $L$ . We consider the normalization of  $M$ , say, a  $(\hat{\mathcal{V}}_M, \hat{\mathcal{B}}_M, \hat{\mathcal{O}}_M)$ -1dta  $\hat{M}$ , defined in Section 3.4, with  $\hat{\mathcal{V}}_M = \{V_M\}$ . Let  $\hat{T}_M$  indicate the subspace topology, restricted to  $V_M$ , induced from  $T_V$  (i.e.,  $\hat{T}_M = \{P \cap V_M \mid P \in T_V\}$ ). It then follows that, since  $V$  is compact,  $V_M$  must be compact. Let  $\hat{M} = (\Sigma, \{\phi, \$\}, V_M, \{B_\sigma \mid \sigma \in \check{\Sigma}, v_0, E_{acc}, E_{rej}\})$ , provided that  $(E_{acc}, E_{rej}) \in \hat{\mathcal{O}}_M$  and  $B_\sigma \in \hat{\mathcal{B}}_M$  for all  $\sigma \in \check{\Sigma}$ . Hereafter, we want to show that  $L$  is a regular language by converting  $\hat{M}$  into an equivalent 1dfa  $N$ .

By the uniformizability of  $V_M$ , there exists a uniform structure  $\Phi_M$  of  $V_M$  that is compatible with  $\hat{T}_M$ . The uniform topological equicontinuity also implies that (\*) for any  $C \subseteq V \times V$  in  $\Phi_M$ , the set  $R^C = \{(u, v) \in V^2 \mid \forall \sigma [(B_\sigma(u), B_\sigma(v)) \in C]\}$  is a subset of  $C$ .

To eliminate the right-endmarker, we define  $E_r^\$ = \{v \in V_M \mid B_\$(v) \in E_r\}$  for each  $r \in \{acc, rej\}$  and consider  $T^\$ = \{W \in \hat{T}_M \mid \exists r \in \{acc, rej\}[W \cap E_r^\$ = \emptyset]\}$ . Here, we claim that  $V_M = \bigcup_{W \in T^\$} W$ .

By the compatibility of  $\Phi_M$  with  $\hat{T}_M$ , for each  $W \in \hat{T}_M$ , we have a point  $w \in V_M$  and a set  $C \in \Phi_M$  satisfying  $C[w] \subseteq W$ . We define  $P = \{(w, C) \mid w \in V_M, C \in \Phi_M, \exists W \in T^\#[C[w] \subseteq W]\}$ . It follows that  $V_M = \bigcup_{(w, C) \in P} C[w]$ . Hence,  $\{C[w] \mid (w, C) \in P\}$  is a covering of  $V_M$ .

By the compactness of  $V_M$ , we can choose a finite subcovering  $\{P_i\}_{i \in [t]}$  of  $V_M$  for a certain number  $t \in \mathbb{N}^+$ . Next, we define a binary relation  $\equiv$  on  $V_M$  as:  $v \equiv w$  if and only if there exists an  $i \in [t]$  such that  $v, w \in P_i$  and  $v, w \notin P_j$  for any  $j \in [m] - \{i\}$ . We want to show that  $\equiv$  is an equivalence relation on  $V_M$ . Clearly,  $v \equiv v$ . If  $v \equiv w$ , then  $w \equiv v$  holds. If  $v \equiv w$  and  $w \equiv z$ , then we obtain  $v \equiv z$ .

If we define  $[v] = \{w \in V_M \mid v \equiv w\}$  for every  $v \in V_M$ , then the set  $V_M/\equiv$  of all equivalence classes coincides with  $\{[v] \mid v \in V_M\}$ . It is easy to show that  $V_M/\equiv$  is a finite set. Let  $m = |V_M/\equiv|$  and take  $m$  distinct points  $v_0, v_1, v_2, \dots, v_{m-1} \in V_M$  such that  $V_M/\equiv$  equals  $\{[v_0], [v_1], \dots, [v_{m-1}]\}$ .

Finally, we define the desired 1dfa  $N = (Q, \Sigma, \{\$, \#\}, \delta, q_0, Q_{acc}, Q_{rej})$  as follows. Let  $Q = \{v_0, v_1, \dots, v_{m-1}\}$ . Let  $Q_{acc} = \{v_i \in Q \mid v_i \in E_{acc}\}$  and  $Q_{rej} = \{v_i \in Q \mid v_i \in E_{rej}\}$ . Moreover, we define  $\delta$  as:  $\delta(v_i, \sigma) = v_j$  if and only if there are  $w_i, w_j \in V_M$  such that  $v_i \equiv w_i$ ,  $v_j \equiv w_j$ , and  $B_\sigma(w_i) = w_j$ . We claim that  $\delta$  is a well-defined function from  $Q \times \Sigma$  to  $Q$ . Assume that  $\delta(v_i, \sigma) = v_j$  and  $\delta(v_k, \sigma) = v_j$ . There are four points  $w_i, w_j, w'_j, w_k$  such that  $v_i \equiv w_i$ ,  $v_j \equiv w_j \equiv w'_j$ ,  $v_k \equiv w_k$ ,  $B_\sigma(w_i) = w_j$ , and  $B_\sigma(w_k) = w'_j$ . By Statement (\*), for any  $C \in \Phi_M$ ,  $(w_j, w'_j) \in C$  implies  $(w_i, w_k) \in C$ . Since  $\{P_i\}_{i \in [t]}$  is a subset of  $\{C[w] \mid (w, C) \in P\}$ , it follows that  $w_i \equiv w_k$ .

By the definition of  $N$ , we conclude that  $\hat{M}$  accepts (resp., rejects)  $x$  if and only if  $N$  accepts (resp., rejects)  $x$ . Therefore,  $L(\hat{M}) = L(N)$  follows. This completes the proof of the lemma.  $\square$

*Proof Sketch of Theorem 5.1.* As seen in Section 2.2, any 1dfa can be viewed as a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta of the particular form. By the definition of such a 1dta, it follows that  $\mathcal{V}$ 's points are uniformizable and  $(\mathcal{V}, \mathcal{B})$  is compact and uniformly topologically equicontinuous. Combining this with Lemma 5.2, we immediately obtain the desired characterization of regular languages in terms of 1dta's.  $\square$

The condition of compactness in Theorem 5.1 is needed because, without it, 1dta's can recognize non-regular languages.

**Lemma 5.3** *Let  $\mathcal{V} = \{(\mathbb{Z}, \mathcal{P}(\mathbb{Z}))\}$ ,  $\mathcal{B} = \{B_\$, B_\$, B_a, B_b\}$  with  $\Sigma = \{a, b\}$ ,  $B_\$ = B_\$ = I$ ,  $B_a(n) = n + 1$ , and  $B_b(n) = n - 1$  for all  $n \in \mathbb{Z}$ , and  $\mathcal{O} = \{(E_{acc}, E_{rej})\}$  with  $E_{acc} = \{0\}$  and  $E_{rej} = \mathbb{Z} - \{0\}$ . The sub-automata base  $(\mathcal{V}, \mathcal{B})$  is uniformly topologically equicontinuous but not compact. There exists a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta that recognizes the language  $Equal = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$ , where  $\#_a(w)$  indicates the number of all occurrences of symbol  $a$  in string  $w$ .*

## 6. Multi-Valued Operators and Nondeterminism

*Nondeterminism* is a ubiquitous feature, which appears in many fields of computer science. Jeandel [5] considered such a feature for his model of topological automata. Here, we define a nondeterministic version of our  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1dta's, called *1-way nondeterministic topological automata* (or *1nta's*), in such a way that it naturally extends the standard definition of *1-way nondeterministic finite automata* (or *1nfa's*), each of which nondeterministically chooses one next state out of a predetermined set of possible states at every step.

Unlike the previous sections, we now introduce multi-valued operators, which map each element to “multiple” elements. Formally, a *multi-valued operator* is a map from each point  $x$  of a given

topological space  $(V_1, T_{V_1})$  to a certain number of points of another topological space  $(V_2, T_{V_2})$ . Although this operator can be seen as an ordinary map from  $V_1$  to  $\mathcal{P}(V_2)$ , we customarily express such a multi-valued map as  $B : V_1 \rightarrow V_2$ . A multi-valued operator is said to be *continuous* if, for any  $x \in V_1$  and for any neighborhood  $N$  of  $B(x)$ , there exists a neighborhood  $N'$  of  $x$  satisfying  $B(N') \subseteq N$ .

Next, we define an *extended automata base* to be a tuple  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  in which  $\mathcal{B}$  consists of continuous multi-valued operators  $B : V \rightarrow V$  for sets  $V \in \mathcal{V}$ . Given an extended automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ , a  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1nta  $M$  is a tuple  $(\Sigma, \{\dagger, \$\}, V, \{B_{\sigma} \mid \sigma \in \check{\Sigma}, v_0, E_{acc}, E_{rej}\})$  such that  $B_{\sigma}$  is a multi-valued continuous operator from  $V$  to  $V$  for each  $\sigma \in \check{\Sigma}$ . This  $M$  works as follows. On input  $x$  (which is given as  $\dagger x \$$  on an input tape), we apply  $B_{\dagger x \$}$  to  $v_0$ , where  $B_{\sigma} \diamond B_{\tau}(v) = B_{\sigma}(B_{\tau}(v)) (= \bigcup_{w \in B_{\tau}(v)} B_{\sigma}(w))$  and  $B_{\dagger x \$}^{\diamond} = B_{\$} \diamond B_{x_n} \diamond \cdots \diamond B_{x_2} \diamond B_{x_1} \diamond B_{\dagger}$  if  $x = x_1 x_2 \cdots x_n$ . We say that  $M$  *accepts* an input  $x$  if  $B_{\dagger x \$}^{\diamond}(v_0) \cap E_{acc} \neq \emptyset$  and that  $M$  *rejects*  $x$  if  $B_{\dagger x \$}^{\diamond}(v_0) \subseteq E_{rej}$ . Given a class  $T$  of subsets of  $V$ , the notation  $\text{co-}T$  expresses the class of the complements of sets in  $T$  (with respect to  $V$ ).

It is known that nondeterministic finite automata can be simulated by deterministic ones using exponentially more inner states. In the following lemma, for a given topological space  $(V, T_V)$ , we expand  $V$  to  $T_V^+$  so that  $(T_V^+, T^{\circ}(T_V^+))$  forms a topological space for an appropriately chosen topology  $T^{\circ}(T_V^+)$ . Following Michael [7], we here take  $T^{\circ}(T_V^+)$  as the topology on  $T_V^+$  that is generated by the bases  $\{[A]^+, [A]^- \mid A \in T_V\}$ , where  $[A]^+ = \{X \in T_V^+ \mid X \subseteq A\}$  and  $[A]^- = \{X \in T_V^+ \mid X \cap A \neq \emptyset\}$ . This topology is known as the *Vietoris topology*, adapted to  $T_V^+$ .

**Lemma 6.1** *Let  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$  be any extended automata base. There exists an automata base  $(\mathcal{V}', \mathcal{B}', \mathcal{O}')$  with  $\mathcal{V}' = \{(T_V^+, T^{\circ}(T_V^+)) \mid V \in \mathcal{V}\}$  such that, for any  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1nta  $M$  with  $v_0$  and  $V$ , there is an equivalent  $(\mathcal{V}', \mathcal{B}', \mathcal{O}')$ -1nta  $N$ , provided that  $\{v_0\} \in T_V^+$ .*

*Proof.* From a given extended automata base  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ , we define  $\mathcal{B}'$  and  $\mathcal{O}'$  as follows. Let  $\mathcal{B}' = \{B' : T_V^+ \rightarrow T_V^+ \mid B \in \mathcal{B}\}$ , where  $B'(W) = \bigcup_{w \in W} B(w)$  for any element  $W \in T_V^+$ , and let  $\mathcal{O}' = \{(E'_1, E'_2) \in T_V^+ \times T_V^+ \mid V \in \mathcal{V}, E'_1 \cap E'_2 = \emptyset, E'_1, E'_2 \in T^{\circ}(T_V^+) \cap \text{co-}T^{\circ}(T_V^+), \exists (E_1, E_2) \in \mathcal{O} \text{ s.t. } \forall A_1 \in E'_1 [A_1 \cap E_1 \neq \emptyset] \wedge \forall A_2 \in E'_2 [A_2 \subseteq E_2]\}$ .

Next, we argue that  $(\mathcal{V}', \mathcal{B}', \mathcal{O}')$  forms a proper automata base. Let  $B$  be any multi-valued continuous operator in  $\mathcal{B}$  and take its corresponding operator  $B'$ . We want to show that  $B'$  is continuous. Let  $B'(W) = U$  for  $U, W \in T_V^+$  and consider any open set  $S$  in  $T^{\circ}(T_V^+)$  containing  $U$ . Without loss of generality, we assume that  $S$  is either  $[U]^-$  or  $[U]^+$  because  $U$  is an open set in  $T_V^+$ . If  $S = [U]^-$ , then we take  $R = [W]^-$ . For any  $Y \in R$ , it follows that  $B'(Y) \subseteq U$ , and thus  $B'(Y) \in S$ . In contrast, if  $S = [U]^+$ , then we take  $R = [W]^+$ . For any  $Y \in R$ , since  $W \cap Y \neq \emptyset$ , we obtain  $U \cap B'(Y) \neq \emptyset$ ; hence,  $B'(Y) \in S$ .

Let  $M = (\Sigma, \{\dagger, \$\}, V, \{B_{\sigma}\}_{\sigma \in \check{\Sigma}}, v_0, E_{acc}, E_{rej})$  be any given  $(\mathcal{V}, \mathcal{B}, \mathcal{O})$ -1nta with  $\{v_0\} \in T_V^+$ . We then define  $N = (\Sigma, \{\dagger, \$\}, T_V^+, \{B'_{\sigma}\}_{\sigma \in \check{\Sigma}}, v'_0, E'_{acc}, E'_{rej})$ , where  $v'_0 = \{v_0\}$ .

By induction on the length of any input string  $w \in \{\dagger\} \cup \dagger \Sigma^* \cup \dagger \Sigma^* \$$ , we want to show that  $B_w^{\diamond}(v_0) = B'_w(v'_0)$ . For the basis case, since  $B_{\dagger}^{\diamond}(v_0) = B_{\dagger}(v_0)$  and  $B'_{\dagger}(v'_0) = \bigcup_{w \in v'_0} B_{\dagger}(w) =$

$B_{\dagger}(v_0)$ ,  $B_{\dagger}^{\diamond}(v_0) = B'_{\dagger}(v'_0)$  follows. In the induction step, we assume that  $B_{\dagger x}^{\diamond}(v_0) = B'_{\dagger x}(v'_0)$ . Consider an input string  $xa$ . Let  $U_x = B_{\dagger x}^{\diamond}(v_0)$ . It then follows that  $B_{\dagger xa}^{\diamond}(v_0) = B_a \diamond B_{\dagger x}^{\diamond}(v_0) = \bigcup_{w \in U_x} B_a(w)$  and that  $B'_{\dagger xa}(v'_0) = \bigcup_{w \in B'_{\dagger x}(v'_0)} B_a(w) = \bigcup_{w \in U_x} B_a(w)$ . Thus, we obtain  $B_{\dagger xa}^{\diamond}(v_0) = B'_{\dagger xa}(v'_0)$ , as requested.

The above fact implies that  $v \in B_{\dagger x \S}^{\diamond}(v_0)$  if and only if  $v \in B'_{\dagger x \S}(v'_0)$ . Let us define  $E'_{acc} = \{E' \in T^{\circ}(T_V^+) \cap \text{co-}T^{\circ}(T_V^+) \mid \forall A \in E'[A \cap E_{acc} \neq \emptyset]\}$ . Note that  $B_{\dagger x \S}^{\diamond} \cap E_{acc} \neq \emptyset$  if and only if  $B'_{\dagger x \S}(v'_0) \cap E'_{acc} \neq \emptyset$ . Therefore,  $x$  is accepted by  $M$  if and only if  $x$  is accepted by  $N$ . Similarly, for the rejection of  $x$ , we define  $E'_{rej} = \{E' \in T^{\circ}(T_V^+) \cap \text{co-}T^{\circ}(T_V^+) \mid \forall A \in E'[A \subseteq E_{rej}]\}$ .  $\square$

## References

- [1] A. AMBAINIS, A. KIKUSTS, M. VALDATS, On the class of languages recognized by 1-way quantum finite automata. In: A. FERREIRA, H. REICHEL (eds.), *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2001)*. Lecture Notes in Computer Science 2010, Springer, 2001, 75–86.
- [2] S. BOZAPALIDIS, Extending stochastic and quantum functions. *Theory of Computing Systems* 36 (2003), 183–197.
- [3] H. EHRIG, W. KÜHNEL, Topological automata. *RAIRO - Theoretical Informatics and Applications - Informatique Thorique et Applications* 91 (1974) R-3, 73–91.
- [4] J. HOPCROFT, R. MOTWANI, J. ULLMAN, *An Introduction to Automata Theory, Languages, and Computation (2nd edition)*. Addison-Wesley, Reading MA, 2001.
- [5] E. JEANDEL, Topological automata. *Theory of Computing Systems* 40 (2007), 397–407.
- [6] A. KONDACS, J. WATROUS, On the power of quantum finite state automata. In: *Proc. of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*. IEEE, 1997, 66–75.
- [7] E. MICHAEL, Topologies on spaces of subsets. *Transactions of the American Mathematical Society* 71 (1951), 152–182.
- [8] C. MOORE, J. CRUTCHFIELD, Quantum automata and quantum languages. *Theoretical Computer Science* 237 (2000), 275–306.
- [9] M. O. RABIN, Probabilistic automata. *Information and Control* 6 (1963), 230–245.
- [10] T. YAMAKAMI, Analysis of quantum functions. *International Journal of Foundations of Computer Science* 14 (2003), 815–852.
- [11] T. YAMAKAMI, One-way reversible and quantum finite automata with advice. *Information and Computation* 239 (2014), 122–148.

## Author Index

Arrighi, Pablo, 31  
Berglund, Martin, 49  
Chouteau, Clément, 31  
Dimitrijevs, Maksims, 65  
Drewes, Frank, 49  
Facchini, Stefano, 31  
Guillon, Bruno, 11  
Holzer, Markus, 83  
Křivka, Zbyněk, 117  
Klíma, Ondřej, 99  
Kocman, Radim, 117  
Kutrib, Martin, 83, 133  
Li, Yongming, 181  
Martiel, Simon, 31  
Meduna, Alexander, 117  
Mráz, František, 149  
Nagy, Benedek, 117  
Otto, Friedrich, 133, 149  
Plátek, Martin, 149  
Polák, Libor, 99  
Sempere, José M., 29  
Truthe, Bianca, 165  
van der Merwe, Brink, 49  
Wang, Qichao, 181  
Yakaryılmaz, Abuzer, 65  
Yamakami, Tomoyuki, 197

