Henning Bordihn, Rudolf Freund, Benedek Nagy, and György Vaszil (eds.)

Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016)

Short Papers

Druck: Druckerei Riegelnik
1080 Wien, Piaristengasse 19

ISBN 978-3-200-04725-9

Preface

The *Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016)* has been organized to provide an opportunity for researchers who work on different aspects of non-classical and classical models of automata and grammars to exchange and develop novel ideas. Many non-classical models of automata and grammar-like structures are the natural objects of theoretical computer science. They are studied from different points of view in various areas, both as theoretical concepts and as formal models for applications. The purpose of the NCMA workshop series is to promote a deeper and interdisciplinary coverage of this particular area and in this way to foster new insights and substantial progress in computer science as a whole.

The first workshop on Non-Classical Models of Automata and Applications, NCMA 2009, was held in Wroclaw, Poland, in 2009 as a satellite event of the 17th International Symposium on Fundamentals of Computation Theory (FCT 2009). It was sponsored by the AutoMathA project of the European Science Foundation (ESF). The second workshop, NCMA 2010, was held in Jena, Germany, as an associated workshop of the 11th International Conference on Membrane Computing (CMC 11); the third workshop, NCMA 2011, was organized at the University of Milan, Italy, in close proximity to the 15th Conference on Developments in Language Theory (DLT 2011). In contrast to the previous workshops, the next NCMA workshops were organized as stand-alone events; NCMA 2012 was held in Fribourg, Switzerland, in August 2012, NCMA 2013 took place in Umeå, Sweden, in August 2013, NCMA 2014 was held in Kassel, Germany, in July 2014, and NCMA 2015 took place in Porto, Portugal, August 31st – September 1st, 2015.

Now the Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016) is held in Debrecen, Hungary, in the period of August 29th – 30th, 2016. It is organized by the Department of Computer Science of the Faculty of Informatics at the University of Debrecen. We expect NCMA 2016 to be again a scientifically valuable event with interesting presentations, exciting results, and stimulating discussions. We hope that the friendly atmosphere and the nice surroundings will also help to make this workshop a worthwhile event that will lead to new insights and possibly new cooperations.

In addition to the four invited talks and the 15 regular contributions, NCMA 2016 also features seven short presentations to emphasize its workshop character, each of them also having been evaluated by at least two members of the program committee. The extended abstracts of these short presentations are contained in this volume.

We are grateful to the Department of Computer Science and the Faculty of Informatics of the University of Debrecen for the local organization and for the financial support of NCMA 2016, and we would also like to thank the Institute of Computer Languages of the TU Wien for covering the production costs of the proceedings and this collection of short papers.

August 2016

Rudolf Freund, Wien
Henning Bordihn, Potsdam
Benedek Nagy, Debrecen and Famagusta
György Vaszil, Debrecen

Table of Contents

Short Papers

ON REGULAR LANGUAGES ACCEPTED BY ONE-WAY JUMPING FINITE AUTOMATA 7

Szilárd Zsolt Fazekas and Akihiro Yamamura

COMPLEXITY ON UNARY UNION-FREE AND UNARY STAR-FREE LANGUAGES 15

Michal Hospodár

JUMPING AND PUMPING LEMMAS AND THEIR APPLICATIONS 25

Grzegorz Madejski

EXTENDED UNIFORMLY LIMITED T0L LANGUAGES AND MILD CONTEXT-SENSITIVITY 35

Suna Bensch, Martin Kutrib, and Andreas Malcher

ORDERED RESTARTING AUTOMATA: RECENT RESULTS AND OPEN PROBLEMS 47

Kent Kwee and Friedrich Otto

REGULAR ARRAY GRAMMARS AND BOUSTROPHEDON FINITE AUTOMATA 55

Henning Fernau, Meenakshi Paramasivan, and D. Gnanaraj Thomas

ON LANGUAGES ACCEPTED BY WEIGHTED RESTARTING AUTOMATA 65

Qichao Wang

Author Index 73

ON REGULAR LANGUAGES ACCEPTED BY ONE-WAY JUMPING FINITE AUTOMATA

Szilárd Zsolt Fazekas Akihiro Yamamura

Graduate School of Engineering Science
Department of Mathematical Science and Electrical-Electronic-Computer Engineering
Akita University
1-1 Tegata Gakuen-machi, Akita 010-8502, Japan
{szilard.fazekas,yamamura}@ie.akita-u.ac.jp

Abstract
The recently introduced one-way jumping finite automata (OWJFA) model processes the input in a determnistic but non-contiguous manner, by making jumps whenever the current state has no defined transitions for the upcoming input letter and continuing from the start upon reaching the end. The class of languages accepted by (right-)OWJFA is a strict superset of the class of regular languages. In this paper we explore sufficient conditions for the accepted language to be regular by analyzing the number of times the automaton jumps over a position.

1. Introduction

Most classical computer science methods process information in a contiguous way. For instance, in the case of deterministic finite automata[3], the read head starts at the beginning of the input word, it moves in a left-to-right direction, and it reads the input symbol-by-symbol. In certain cases, however, the processing is conceptually simpler if the letters of the input are rearranged. Take as an easy example, checking whether a binary word has an equal number of both letters of its alphabet. If the input were processed by always jumping to a letter different than the last one read, then a finite state machine with the same graph as a DFA accepting $\{ab\}^*$ could perform the required computation. However, the language of words having the same number of a's and b's is one of the textbook examples of a non-regular language.

In recent years new automata models were introduced, which diverge from the usual sequence in which the input letters are read. Several of these models work by "processing" the input non-contiguously, i.e., reading and deleting/marking a letter and then moving to a position which is not necessarily the next one to the right. Restarting automata process the input by looking at a constant sized window of it and moving the window to the beginning of the input after a letter is read and deleted [4]. Input revolving automata are equipped with an extra transition function, based on which in every step the remaining part of the input can be shifted cyclically before reading it [1]. Jumping finite automata [5] can jump over a part of the input word after reading and deleting a symbol and continue processing from there.

To restrict the inherent non-determinism in jumping finite automata, a variant of it, the one-way jumping finite automata has been proposed recently [2]. The moves are similar to jumping finite automata, but with some changes leading to a deterministic behavior. The read head moves in one direction only and starts at the beginning of the input word. It moves from left to right (and possibly jumps over parts of the input) and upon reaching the end of the input word it is returned to the beginning of the input, continuing the computation until all the letters are read or the automaton is stuck in a state in which it cannot read any letter of the remaining input. If a transition is defined for the current state and the next letter to be read, then the automaton reads and marks the symbol as read. If not, but in the remaining input there are letters for which a transition is defined from the current state, the read head jumps to the nearest such letter to the right.

After briefly presenting a few definitions and an example we go on to discuss a sufficient condition for the language accepted by a right one-way jumping finite automaton to be regular. We conjecture that the condition is necessary, too.

An *alphabet* Σ is a finite, non-empty set and its elements are called *letters.* A word is a sequence of letters and the set of all words formed by concatenating (0 or more) elements of Σ is Σ^*. The *empty word*, i.e., the word containing no letters is ε.

A *(nondeterministic) finite automaton*, a NFA for short, is a quintuple $M = (Q, \Sigma, R, \mathbf{s}, F)$, where Q is the finite set of states, Σ is the finite input alphabet, $\Sigma \cap Q = \emptyset$, $R \subseteq Q \times \{\Sigma \cup \varepsilon\} \times Q$, $\mathbf{s} \in Q$ is the start state, and $F \subseteq Q$ is the set of final states. Elements of R are referred to as rules of M and we write $\mathbf{p}y \to \mathbf{q} \in R$ instead of $(\mathbf{p}, y, \mathbf{q}) \in R$. A configuration of M is a string in $Q \times \Sigma^*$. M is an ε-free FA if $\mathbf{p}y \to \mathbf{q} \in R$ implies $|y| = 1$. M is a *deterministic finite automaton*, a DFA for short, if (1) it is an ε-free FA and (2) for each $\mathbf{p} \in Q$ and each $a \in \Sigma$, there is no more than one $\mathbf{q} \in Q$ such that $\mathbf{p}a \to \mathbf{q} \in R$. A NFA or DFA makes a transition from configuration $\mathbf{p}w$ to configuration $\mathbf{q}w'$ if $w = aw'$ and $\mathbf{p}a \to \mathbf{q} \in R$, where $\mathbf{p}, \mathbf{q} \in Q$, $w, w' \in \Sigma^*$ and $a \in \Sigma \cup \{\varepsilon\}$. We denote this by $\mathbf{p}w \Rightarrow \mathbf{q}w'$ and the reflexive and transitive closure of the relation $\Rightarrow$ by $\Rightarrow^*$. A word w is *accepted* by a FA (or DFA) M if there exists $\mathbf{f} \in F$, such that $\mathbf{s}w \Rightarrow^* \mathbf{f}$. Then, the language accepted by M is $L(M) = \{w \in \Sigma^* \mid \exists \mathbf{f} \in F : \mathbf{s}w \Rightarrow^* \mathbf{f}\}$

The *cardinality* of Q, denoted by $|Q|$, is the number of elements of Q. We extend this notation to words $w \in \Sigma^*$, where $|w|$ is the *length* of the word w, the number of all occurrences of all letters of Σ in w. Further extending the notation, $|w|_a$ denotes the number of occurrences of the letter a in w.

2. One-Way Jumping Finite Automata

A right one-way jumping finite automaton M is based on a deterministic finite automaton. The read head of M starts at the leftmost letter of the input word and it moves rightward. M can jump over a part of the word after reading a symbol. If the read head of M reaches the right end of the word, then it continues from the the left end, again.

A *right one-way jumping finite automaton*, ROWJFA for short, is a quintuple $M = (Q, \Sigma, R, s, F)$, where Q, Σ, R, $\mathbf{s}$ and F are defined as in a DFA. By analogy with a DFA, members of R are referred to as rules of M and we write $\mathbf{p}a \to \mathbf{q} \in R$ instead of $(\mathbf{p}, a, \mathbf{q}) \in R$. A configuration of M is any string in $Q\Sigma^*$.

The *right one-way jumping relation*, symbolically denoted by $\circlearrowright$, over $Q\Sigma^*$, is defined as follows. Suppose that x and y belong to Σ^*, a belongs to Σ, $\mathbf{p}$ and $\mathbf{q}$ are states in Q and $\mathbf{p}a \to \mathbf{q} \in R$. Then the ROWJFA M makes a jump from the configuration $\mathbf{p}xay$ to the configuration $\mathbf{q}yx$, symbolically written as

$$\mathbf{p}xay \circlearrowright \mathbf{q}yx$$

if x belongs to $\{\Sigma \setminus \Sigma_\mathbf{p}\}^*$ where $\Sigma_\mathbf{p} = \{b \in \Sigma \mid (\mathbf{p}, b, \mathbf{p}') \in R \text{ for some } \mathbf{p}' \in Q\}$.

In the standard manner, we extend $\circlearrowright$ to $\circlearrowright^m$, where $m \geq 0$. Let $\circlearrowright^*$ denote the transitive-reflexive closure of $\circlearrowright$.

The language accepted by the ROWJFA M, denoted by $L(M)$, is defined to be

$$L(M) = \{w \in \Sigma^* \mid \mathbf{s}w \circlearrowright^* \mathbf{f}, \mathbf{f} \in F\}$$

We say that M accepts a string w in Σ^* if w belongs to $L(M)$, and M rejects w otherwise.

Example 2.1 *Let M be a ROWJFA given by*

$$M = (\{\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2\}, \{a, b, c\}, R, \mathbf{q}_0, \{\mathbf{q}_0\}),$$

where R consists of the rules $\mathbf{q}_0 a \to \mathbf{q}_1$, $\mathbf{q}_1 b \to \mathbf{q}_2$ and $\mathbf{q}_2 c \to \mathbf{q}_0$. Starting from $\mathbf{q}_0$, M has to read some a, some b and some c entering again the start (and also the final) state $\mathbf{q}_0$. All these occurrences of a, b and c can appear anywhere in the input word. Therefore, the accepted language is the non-context-free language $\{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$

We denote the families of languages accepted by right one-way jumping finite automata by **ROWJ**.

Through some languages commonly used for distinguishing classes of the Chomsky hierarchy and a pumping lemma for ROWJFA, one can easily determine [2] the relations between **ROWJ** and the classes of the Chomsky hierarchy, **REG**, **CF** and **CS**, the families of regular languages, context-free languages and context-sensitive languages, respectively.

- **REG** $\subsetneq$ **ROWJ**.
- **CF** and **ROWJ** are incomparable.
- **ROWJ** $\subsetneq$ **CS**.

Furthermore, the class **ROWJ** is not closed under any of the usual operations on languages, i.e., intersection, concatenation, reversal, intersection with regular languages, concatenation with regular languages, substitution and Kleene star or Kleene plus.

3. Regularity of ROWJ Languages

To put the sufficient regularity condition in context, let us recall first another type of automata. In [1] the authors define *input revolving automata* and describe the way they work as follows (we recall only the description of right-revolving automata here).

Definition 3.1 *[1] An extended finite automaton is a 6-tuple* $(Q, \Sigma, \delta, \Delta, q_0, F)$*, where* Q *is the (finite) set of states,* Σ *is the input alphabet,* δ *and* Δ *are mappings from* $Q \times (\Sigma \cup \{\lambda\})$ *to* 2^Q*, where* δ *is called the transition function, and* Δ *is called the input operation function,* $q_0 \in Q$ *is the initial state, and* $F \subseteq Q$ *is the set of accepting states.*

The different operations on the input are formally distinguished by different interpretations of the mapping Δ. Consider configurations of extended finite automata to be tuples (q, w), where $q \in Q$ is the current state, and $w \in \Sigma^*$ is the yet unread part of the input. The transition of a configuration into a successor configuration can be induced by either δ or Δ. In the case that both are applicable, the automaton chooses a transition nondeterministically.

- Let a be in $\Sigma \cup \{\lambda\}$ and $w \in \Sigma^*$. If p is in $\delta(q, a)$, then $(q, aw) \vdash_A (p, w)$.
- An input operation is performed by applying the mapping Δ. For $a \in \Sigma \cup \{\lambda\}$, $b \in \Sigma$, $w \in \Sigma^*$, and $p \in \Delta(q, a)$, a right-revolving transition is defined by $(q, aw) \vdash_A (p, wa)$, if $a \in \Sigma$, and $(q, bw) \vdash_A (p, wb)$ and $(q, \lambda) \vdash_A (p, \lambda)$, if $a = \lambda$. This means that in the latter case, $a = \lambda$, the next input letter is shifted to the end regardless of what it is.

So, right-revolving automata may skip certain letters of the input depending on the current state. This is exactly what happens when a ROWJFA reads the input word and finds a letter for which there is no transition defined from the current state.

Proposition 3.2 *[2] For any ROWJFA* $M = (Q, \Sigma, R, s, F)$ *there exists a right-revolving automaton* $M' = (Q', \Sigma, \delta, \Delta, q_0, F')$*, such that* $L(M) = L(M')$*.*

We note that not all languages accepted by right revolving input automata are in **ROWJ**. Take, for instance, the right-revolving automaton $M = (\{q_0, q_1, q_a, q_b\}, \{a, b\}, \delta, \Delta, q_0, \{q_0, q_1\})$, with transitions given by $\delta(q_0, a) = q_a$, $\delta(q_0, b) = q_b$, $\delta(q_1, b) = q_b$ and $\Delta(q_a, \lambda) = q_0$, $\Delta(q_b, \lambda) = q_1$. It is easy to see that for a word $w \in L(M)$, if $|w|_a = |w|_b$, then $w = (ab)^n$, whereas $|w|_a = k|w|_b$ implies $w = (a^k b)^n$, etc. In general, if the first b occurs at position i, then between every two consecutive b's there are at least $i - 1$ occurrences of a. There exists no ROWJFA, which accepts $L(M)$.

In the case of revolving automata (having also left revolving operations), putting a constant bound on the number of input revolving operations results in these machines accepting only regular languages[1] and, in fact, constant-bound revolving characterizes regular languages. This, together with Proposition 3.2 implies that ROWJFA which only jump over a constant-bounded number of letters, accept regular languages. However, in the case of ROWJFA, this condition does not characterize regular languages, as can be seen, for instance, from the automaton in Example 3.3 accepting input ad^ncbcba for any n, by jumping over d^n in the first sweeps.

To prove regularity conditions of languages accepted by ROWJFA, we need to look at the case when the automaton jumps over the same position multiple times, or equivalently, the number of times the automaton reaches the last letter of the original input word. We say that states for which there is no transition labeled by the upcoming input letter are *deficient*, i.e., $\mathbf{p} \in Q$ is S-deficient if for all $a \in S \subset \Sigma$ we have $\mathbf{p}a \to \mathbf{q} \notin R$ for any $\mathbf{q} \in Q$.

If a position is jumped over, then the input symbol in that position will be processed in another *sweep.* The number of sweeps needed to process the whole input is the number of times the automaton reaches the last letter of the original input word or, equivalently, one more than the maximum number of times any position is jumped over.

To formalize the notion of sweeps, we introduce an alternative definition for the way a ROWJFA works. Let ROWJFA $\mathcal{A}$ be defined as before by a tuple (Q, Σ, R, q_0, F). We will use a counter $i \in \mathbb{N}$ and a binary status for each input letter, which can be *read* or *unread.*. When starting to read the input word w, we start the counter i from 0, the status of all input symbols is *unread* and $\mathcal{A}$ is in state q_0. In each step the transition from some state q works as follows:

- i is incremented by 1;
- if the status of $w[i \mod |w|]$ is *unread* and there exists a state p such that $qw[i \mod |w|] \to p \in R$ then the current state changes to p and $w[i \mod |w|]$ is marked read; otherwise, the current state and the status of $w[i \mod |w|]$ does not change.

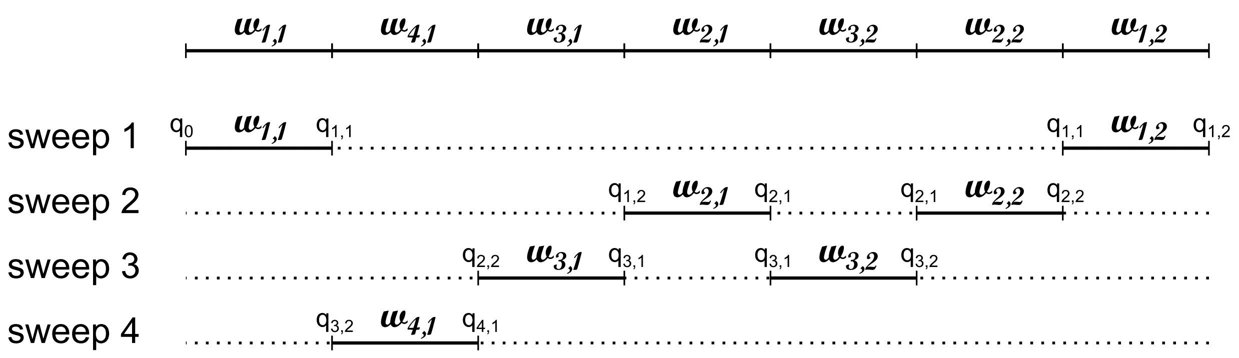
The computation ends when either

1. the status of all positions in w is *read*, or
2. when the current state is q and there exists no pair $i \in \mathbb{N}$ and $p \in Q$ such that $w[i \mod |w|]$ is unread and $qw[i \mod |w|] \to p \in R$.

In case 1, if the current state is a final state, then the input is accepted, in all other cases it is rejected.

At the end of the computation the number of sweeps is $\lfloor i/|w| \rfloor$.

Example 3.3 *Consider the computation in Figure 1 on input* $w_{1,1}w_{4,1}w_{3,1}w_{2,1}w_{3,2}w_{2,2}w_{1,2}$, *where* $w_{i,j} \in \Sigma^+$. *The ROWJFA needs* 4 *sweeps to process it. The order in which the parts of the input are processed is given by the sweeps, first the factors* $w_{1,1}$ *followed by* $w_{1,2}$ *in the first sweep, then the ones in the second sweep, etc., so the order in which the factors are read is* $w_{1,1}w_{1,2}w_{2,1}w_{2,2}w_{3,1}w_{3,2}w_{4,1}$. *The computation starts in the initial state* q_0. *After reading* $w_{i,j}$ *the ROWJFA is in state* $q_{i,j}$. *State* $q_{1,1}$ *is* S*-deficient with* $S = \{a \in \Sigma \mid |w_{4,1}w_{3,1}w_{2,1}w_{3,2}w_{2,2}|_a > 0\}$, *state* $q_{2,1}$ *is* T*-deficient with* $T = \{a \in \Sigma \mid |w_{3,2}|_a > 0\}$, *etc. The input is accepted if* $q_{4,1}$ *is a final state.*



$w_{1,1}$ $w_{4,1}$ $w_{3,1}$ $w_{2,1}$ $w_{3,2}$ $w_{2,2}$ $w_{1,2}$
sweep 1
sweep 2
sweep 3
sweep 4
q_0 $w_{1,1}$ $q_{1,1}$ $q_{1,1}$ $w_{1,2}$ $q_{1,2}$
$q_{1,2}$ $w_{2,1}$ $q_{2,1}$ $q_{2,1}$ $w_{2,2}$ $q_{2,2}$
$q_{2,2}$ $w_{3,1}$ $q_{3,1}$ $q_{3,1}$ $w_{3,2}$ $q_{3,2}$
$q_{3,2}$ $w_{4,1}$ $q_{4,1}$

Figure 1: The input word $w_{1,1}w_{4,1}w_{3,1}w_{2,1}w_{3,2}w_{2,2}w_{1,2}$ processed by a ROWJFA in 4 sweeps.

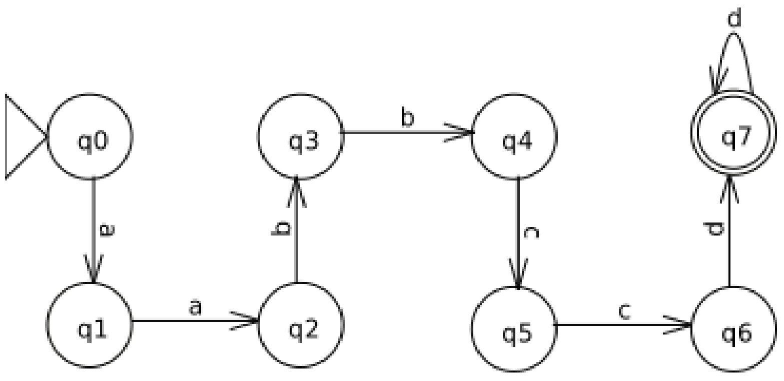


Figure 2: A ROWJFA accepting words w with $|w|_a = |w|_b = |w|_c = 2$ and $|w|_d \geq 1\}$.

The ROWJFA jumps over the letters in $w_{4,1}$ three times before processing them, hence the number of sweeps is four. As an example for a ROWJFA performing such an accepting computation consider the automaton in Figure 2 and the input adcbcba, processed in the order aabbccd.

Theorem 3.4 *Let $\mathcal{A} = (Q, \Sigma, R, q_0, F)$ be a ROWJFA. If there exists a constant k, such that for any word $w \in L(\mathcal{A})$ the number of sweeps needed by $\mathcal{A}$ to process w is at most k, then the language $L(\mathcal{A})$ is regular.*

Proof. We will prove the statement by constructing a NFA, which simulates accepting runs of $\mathcal{A}$. Let our NFA be $\mathcal{B} = (Q', \Sigma, R', q_0', F')$. The set of states is

$$Q' = \{q_{s_1,c_1,f_1,\ldots,s_k,c_k,f_k} \mid s_i, c_i \in Q, 1 \leq i \leq k\}.$$

The indices of the states can be interpreted as follows:

- the index of a state is a series of pairs of the form s_i, c_i, meaning
- $i \leq k$ - the number of the sweep
- $s_i \in Q$ - in which state we started sweep i
- $c_i \in Q$ - current state in sweep i
- $f_i \in \{0, 1\}$ - a flag indicating whether sweep i was needed to process the input: 0 if there have been less than i sweeps, 1 if any letter has been read in a sweep $j \geq i$.

The letters which are in sweep i will all be processed after the ones in sweep $i-1$ and before the ones in sweep $i+1$ and the number of sweeps does not exceed a previously fixed constant. Hence, we can keep track of the computation performed in sweep i by tracking the change of state of $\mathcal{A}$ in the components s_i, c_i, f_i of the states of the NFA.

The construction could be done directly by DFA, but the number of state indices needed to keep track of sweeps, starting and current states of the sweeps, is too large and the construction would be more difficult to follow.

The NFA has the following transitions

$$q_{s_1,c_1,f_1,\dots,s_i,c_i,f_i,\dots s_k,c_k,f_k} a \rightarrow q_{s_1,c_1,f'_1,\dots,s_i,p,f'_i,\dots,s_k,c_k,f'_k} \tag{1}$$

where

- $c_1, \dots, c_{i-1}$ are $\{a\}$-deficient,
- $c_i a \rightarrow p \in R$,
- after the transition the flags for sweeps 1 to i are set, i.e., $f'_1 = \dots = f'_i = 1$,
- the rest of the flags remain unchanged, $f'_j = f_j, \forall i < j \leq k$.

The initial state is q'_0. In addition to the transitions described above, the automaton has the following transitions:

$$q'_0 \lambda \rightarrow q_{q_0,q_0,1,s_2,s_2,0,\dots,s_k,s_k,0}, \tag{2}$$

for all possible values $s_2, \dots, s_k \in Q$.

For all $\ell \leq k$ there are final states of the form

$$q_{q_0,c_1,1,s_2,c_2,f_2,\dots,s_k,c_k,f_k},$$

with $c_{i-1} = s_i$, $f_i = 1$ and $c_\ell \in F$, $\forall i \in \{2, \dots, \ell\}$ and $f_j = 0, \forall j \in \{\ell+1, \dots, k\}$.

The NFA defined above has an accepting path for input w if and only if $w \in L(\mathcal{A})$, because of the following.

- $\mathcal{B}$ can only change at most one of the c_i in any transition of type (1) and does so according to the transition table of $\mathcal{A}$.
- A letter $a \in \Sigma$ can only be processed in sweep i if the sweeps before jumped over it, ensured by the condition $c_1, \dots, c_{i-1}$ are $\{a\}$-deficient.
- If a letter is processed in sweep i by $\mathcal{A}$, then the flags of all sweeps up to i of the NFA state are set to 1, meaning that we know which sweeps were activated.
- The condition $c_{i-1} = s_i$ in the final states ensures that the computation continues between sweeps according to the transition table of $\mathcal{A}$. It might be the case that sweep i was activated but did not process any letters. However, this is not a problem, as in that case the condition $c_{i-1} = s_i = c_i = s_{i+1}$ is checked by the final states of the NFA.

- In the states reachable through transitions (2) we set the flags to 0 for all sweeps but the first and each transition changes the flags of the current sweep processing some input letter and the flags of the sweeps before it; hence, we know that the greatest i with $f_i = 1$ is the last sweep, and we know that at least one letter was processed in that sweep.
- the conditions $f_i = 1, \forall i \in \{1, \ldots, \ell\}$ and $f_j = 0, \forall j \in \{\ell + 1, \ldots, k\}$ on the final states means that ℓ is the last sweep and $c_\ell \in F$ means the last sweep ends in a final state.

□

We conjecture that the condition in the theorem above is both necessary and sufficient, but we have no proof of the opposite implication yet. The reason for the conjecture is that, while processing a long enough input, if the ROWJFA jumps over a certain input position more than a constant-bounded number of times being in some S-deficient state, it will have to jump over at least another position "roughly" the same number of times being in some T-deficient state, thereby "matching the number of $\Sigma \setminus S$ letters to the number of $\Sigma \setminus T$ ones.

The goal of this line of study is to establish an algorithm which decides whether the language accepted by a given ROWJFA is regular. Therefore, even if the condition above is necessary and sufficient, one needs to translate it into a property of the ROWJFA which can be checked in finitely many steps. A likely candidate for this property is that the number of sweeps is always less than a constant bound if and only there is no deficient state which occurs in a cycle in the graph of the automaton.

References

[1] S. BENSCH, H. BORDIHN, M. HOLZER, M. KUTRIB, On input-revolving deterministic and nondeterministic finite automata. *Information and Computation* 207 (2009) 11, 1140–1155.

[2] H. CHIGAHARA, S. FAZEKAS, A. YAMAMURA, One-Way Jumping Finite Automata. *International Journal of Foundations of Computer Science* 27 (2016) 3, 391–405.

[3] J. HOPCROFT, J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, M.A., 1979.

[4] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting Automata. In: *Proc. FCT95, Dresden, Germany, August 1995*. LNCS 965, Springer, 1995, 283–292.

[5] A. MEDUNA, P. ZEMEK, Jumping finite automata. *International Journal of Foundations of Computer Science* 23 (2012) 7, 1555–1578.

COMPLEXITY ON UNARY UNION-FREE AND UNARY STAR-FREE LANGUAGES

Michal Hospodár

Mathematical Institute, Slovak Academy of Sciences
Grešákova 6, 040 11 Košice
hosmich@gmail.com

Abstract
We examine the nondeterministic and deterministic state complexity of the following operations on unary star-free and union-free regular languages: intersection, union, concatenation, squaring, positive closure, Kleene closure, and complementation. The results are compared to the complexity of these operations on unary regular languages. We prove that all lower bounds for regular languages except for complementation hold also for union-free languages for both nondeterministic and deterministic complexity. We also point out that nondeterminism does not accelerate any operation on unary star-free languages but complementation and closures.

1. Introduction

The state complexity of a regular language L, $\mathrm{sc}(L)$, is the smallest number of states in any deterministic finite automaton (DFA) recognizing the language L. Similarly, the nondeterministic state complexity of a regular language L, $\mathrm{nsc}(L)$, is the smallest number of states in any nondeterministic finite automaton (NFA) with a single initial state recognizing the language L. The state complexity of a regular operation is defined as the maximal state complexity of languages resulting from the operation, considered as a function of state complexities of the operands. Recently, state complexities of operations on certain subclasses of regular languages came into interest. The complexities of operations on star-free languages are determined in [1] and [5]. The complexities of operations on union-free languages has been investigated in [6]. These papers left some cases unresolved, namely the complexities on unary star-free and unary union-free languages. Here we provide lower bounds of complexities on these classes.

2. Preliminaries

A language is union-free if it can be represented by a regular expression without $+$, the sign of union. In 2006, Nagy [9] proved that a language is union-free if and only if it can be represented

Research supported by VEGA grant 2/0084/15 and grant APVV-15-0091.

by an automaton which has exactly one cycle-free accepting path from any state. Such automata have single final state since two reachable final states would result in two accepting paths.

Star-free languages are the smallest class containing the finite languages and closed under boolean operations and concatenation. In 1965, Schützenberger [11] proved that a language is star-free if and only if its syntactic monoid is group-free, that is, has only trivial subgroups. An equivalent condition is that the minimal DFA of a star-free language is permutation-free, that is, it has no non-trivial permutation on its states.

The nondeterministic complexities of intersection, union, concatenation, squaring, positive closure, Kleene closure, reversal, and complementation are known to be $(mn, m+n+1, m+n, 2n, n, n+1, n+1, 2^n)$ (see [4]). All these bounds are met by star-free languages, all but reversal by union-free languages. The deterministic complexities of intersection, union, concatenation, squaring, Kleene closure, reversal, and complementation are known to be $(mn, mn, m2^n - 2^{n-1}, n2^n - 2^{n-1}, 2^{n-1} + 2^{n-2}, 2^n, n)$ (see [12]). Since complementation on DFAs does not change the number of states, we do not deal with it. The complexity of positive closure on DFAs is not of interest since the resulting language differs from Kleene closure in at most one string, ε, so the DFAs only differ in at most one state. Therefore we do not mention the deterministic complexity of positive closure.

3. Complexities on arbitrary alphabet

The complexity of operations on union-free languages was investigated by Jirásková and Masopust [6]. The complexity of operations on star-free languages were investigated by Brzozowski and Liu [1] in the deterministic case and by Holzer, Kutrib, and Meckel [5] in the nondeterminsitic case. Here are their results.

Theorem 3.1 (cf. [6], Theorem 6) *Let $m, n \geq 2$. Let K, L be union-free languages over an alphabet Σ with $\operatorname{nsc}(K) = m$ and $\operatorname{nsc}(L) = n$. Then*

(a) $\operatorname{nsc}(K \cap L) \leq mn$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(b) $\operatorname{nsc}(K \cup L) \leq m + n + 1$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(c) $\operatorname{nsc}(KL) \leq m + n$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(d) $\operatorname{nsc}(L^2) \leq 2n$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(e) $\operatorname{nsc}(L^+) \leq n$, *and this bound is tight if* $|\Sigma| \geq 1$*;*
(f) $\operatorname{nsc}(L^*) \leq n + 1$, *and this bound is tight if* $|\Sigma| \geq 1$*;*
(g) $\operatorname{nsc}(L^R) \leq n$, *and this bound is tight if* $|\Sigma| \geq 1$*;*
(h) $\operatorname{nsc}(L^c) \leq 2^n$, *and this bound is tight if* $|\Sigma| \geq 3$.

Theorem 3.2 (cf. [6], Theorem 8) *Let $m, n \geq 2$. Let K, L be union-free languages over an alphabet Σ with $\operatorname{sc}(K) = m$ and $\operatorname{sc}(L) = n$. Then*

(a) $\operatorname{sc}(K \cap L) \leq mn$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(b) $\operatorname{sc}(K \cup L) \leq mn$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(c) $\operatorname{sc}(KL) \leq m2^n - 2^{n-1}$, *and this bound is tight if* $|\Sigma| \geq 2$*;*

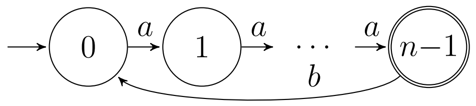


Figure 1: The automaton for the union-free and star-free language $a^{n-1}(ba^{n-1})^*$.

(d) $\mathrm{sc}(L^2) \leq n2^n - 2^{n-1}$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(e) $\mathrm{sc}(L^*) \leq 2^{n-1} + 2^{n-2}$, *and this bound is tight if* $|\Sigma| \geq 2$*;*
(f) $\mathrm{sc}(L^R) \leq 2^n$, *and this bound is tight if* $|\Sigma| \geq 2$ *([8, Theorem 5]).*

Theorem 3.3 *Let* $m, n \geq 2$. *Let* K, L *be star-free languages over an alphabet* Σ *with* $\mathrm{nsc}(K)=m$ *and* $\mathrm{nsc}(L) = n$. *Then*

(a) $\mathrm{nsc}(K \cap L) \leq mn$, *and this bound is tight if* $|\Sigma| \geq 2$ *([5, Theorem 3]);*
(b) $\mathrm{nsc}(K \cup L) \leq m + n + 1$, *and this bound is tight if* $|\Sigma| \geq 2$ *([5, Theorem 2]);*
(c) $\mathrm{nsc}(KL) \leq m + n$, *and this bound is tight if* $|\Sigma| \geq 2$ *([5, Theorem 6]);*
(d) $\mathrm{nsc}(L^2) \leq 2n$, *and this bound is tight if* $|\Sigma| \geq 2$ *(cf. [2, Theorem 3]);*
(e) $\mathrm{nsc}(L^+) \leq n$, *and this bound is tight if* $|\Sigma| \geq 1$*;*
(f) $\mathrm{nsc}(L^*) \leq n + 1$, *and this bound is tight if* $|\Sigma| \geq 2$ *([5, Theorem 7]);*
(g) $\mathrm{nsc}(L^R) \leq n + 1$, *and this bound is tight if* $|\Sigma| \geq 2$ *([5, Theorem 8]);*
(h) $\mathrm{nsc}(L^c) \leq 2^n$, *and this bound is tight if* $|\Sigma| \geq 2$ *([5, Theorem 5]).*

Proof. All upper bounds are the same as for regular languages. The lower bound for squaring was proven in [2, Theorem 3] by the star-free language $L = a^{n-1}(ba^{n-1})^*$ shown in Fig. 1. The lower bound for positive closure is met by the star-free language $L = a^{n-1}a^*$ since $L = L^+$. □

Theorem 3.4 *Let* $m, n \geq 2$. *Let* K, L *be star-free languages over an alphabet* Σ *with* $\mathrm{sc}(K) = m$ *and* $\mathrm{sc}(L) = n$. *Then*

(a) $\mathrm{sc}(K \cap L) \leq mn$, *and this bound is tight if* $|\Sigma| \geq 2$ *([1, Theorem 1]);*
(b) $\mathrm{sc}(K \cup L) \leq mn$, *and this bound is tight if* $|\Sigma| \geq 2$ *([1, Theorem 1]);*
(c) $\mathrm{sc}(KL) \leq m2^n - 2^{n-1}$, *and this bound is tight if* $|\Sigma| \geq 4$ *([1, Theorem 2]);*
(d) $\mathrm{sc}(L^2) \leq n2^n - 2^{n-1}$*;*
(e) $\mathrm{sc}(L^*) \leq 2^{n-1} + 2^{n-2}$, *and this bound is tight if* $|\Sigma| \geq 4$ *([1, Theorem 4]);*
(f) $\mathrm{sc}(L^R) \leq 2^n - 1$, *and this bound is tight if* $|\Sigma| \geq n - 1$ *for* $n \geq 3$ *([1, Theorem 5]).*

We let the lower bound for the complexity of squaring on star-free languages as an open problem.

4. Complexities on unary alphabet

In the unary case, different upper bounds hold for some operations. For example, if L is unary, then $\mathrm{nsc}(L^R) = \mathrm{nsc}(L)$ since $L^R = L$. We prove the lower bounds for union-free and star-free languages in both nondeterministic and deterministic case. First we prove that the lower bound for concatenation $m+n-1$ ([4, Theorem 8]) cannot be exceeded by some classes of languages.

Lemma 4.1 *Let K, L be unary regular languages with* $\text{nsc}(K) = m$ *and* $\text{nsc}(L) = n$ *such that any NFA for K has single final state or L is finite or co-finite. Then* $\text{nsc}(KL) \leq m + n - 1$ *and* $\text{nsc}(L^2) \leq 2n - 1$.

Proof. Let $A = (Q_A, \{a\}, \delta_A, q_s, F_A)$ be an NFA for K and $B = (Q_B, \{a\}, \delta_B, s, F_B)$ be an NFA for L. We construct the NFA C for KL by some of these (not disjoint) cases.

(1) If $|F_A| = 1$, then let $F_A = \{q_f\}$. We define $C = (Q, \{a\}, \delta, q_s, F_B)$ as follows: $Q = Q_A \cup Q_B \setminus \{s\}$; $\delta(q, a) = \delta_A(q, a)$ if $q \in Q_A \setminus F_A$; $\delta(q_f, a) = \delta_A(q, a) \cup \delta_B(s, a)$; $\delta(p, a) = \delta_B(p, a)$ if $p \in Q_B$ and $\delta_B(p, a) \neq s$; $\delta(p, a) = q_f$ if $p \in Q_B$ and $\delta_B(p, a) = s$.
(2) If L is finite, then the NFA B has no cycles, so no transition goes to s in B. We define $C = (Q, \{a\}, \delta, q_s, F_B)$ as follows: $Q = Q_A \cup Q_B \setminus \{s\}$; $\delta(q, a) = \delta_A(q, a) \cup \delta_B(s, a)$ for every $q \in F_A$, in other cases the transition function δ works like δ_A and δ_B.
(3) If L is co-finite, then the NFA B has single cycle, the loop on f, so no transition goes to s in B unless $L = a^*$. We define C in the same way as in (2), or recall that $\text{nsc}(Ka^*) \leq \text{nsc}(K)$.

The computation on C thus can walk between the states of Q_A and the states of $Q_B \setminus \{s\}$ through F_B. Since every unary word is equal to its permutatuon, we have $L(C) = KL$. □

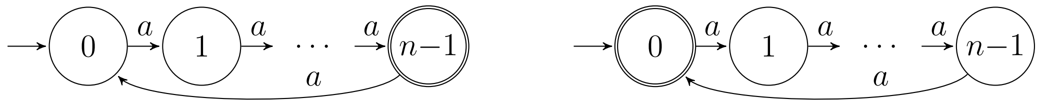


Figure 2: The automata for the union-free languages $a^{n-1}(a^n)^*$ (left) and $(a^n)^*$ (right).

Now look at the class of unary union-free languages. The upper bounds for unary regular languages from [4] are met with the exception of concatenation and squaring.

Theorem 4.2 *Let $m, n \geq 2$, $m \geq n$. Let K, L be unary union-free languages with* $\text{nsc}(K) = m$ *and* $\text{nsc}(L) = n$. *Then*

(a) $\text{nsc}(K \cap L) \leq mn$, *and this bound is tight if* $\gcd(m, n) = 1$;
(b) $\text{nsc}(K \cup L) \leq m + n + 1$, *and this bound is tight if $m \neq kn$ for any natural number k;*
(c) $\text{nsc}(KL) \leq m + n - 1$, *and this bound is tight;*
(d) $\text{nsc}(L^2) \leq 2n - 1$, *and this bound is tight;*
(e) $\text{nsc}(L^+) \leq n$, *and this bound is tight;*
(f) $\text{nsc}(L^*) \leq n + 1$, *and this bound is tight;*
(g) $\text{nsc}(L^c) \leq (n - 1)^2$, *and this bound is tight.*

Proof.

(a) The upper bound mn is met by the union-free languages $K = a^{m-1}(a^m)^*$ and $L = a^{n-1}(a^n)^*$ (see Fig. 2, left), as is proven in [4, Theorem 4].
(b) These languages K and L meet the upper bound $m+n+1$, as is shown in [4, Theorem 2].
(c) By Lemma 4.1, the upper bound is $m+n-1$. The unary union-free languages $K = a^{m-1}a^*$ and $L = a^{n-1}a^*$ meet this bound since NFA for $KL = a^{m+n-2}a^*$ needs $m + n - 1$ states.
(d) By Lemma 4.1, the upper bound is $2n - 1$. The language $L = a^{n-1}a^*$ meets this bound.

(e) The upper bound n is met by the union-free language $L = a^{n-1}(a^n)^*$ from [4, Theorem 9].

(f) The upper bound $n+1$ is met by the same union-free language $L = a^{n-1}(a^n)^*$.

(g) Every unary union-free language has the form $L = a^t(a^{x_1})^*(a^{x_2})^* \dots (a^{x_k})^*$. If $k = 0$, then $L = a^t$ is finite. If $k \geq 2$ and $\gcd(x_i, x_j) = 1$ for some i, j, then L is co-finite. To maximize $\operatorname{nsc}(L^c)$, we look for the longest string possible in L^c. Since for $1 \leq i \leq k$ holds $x_i \leq n$, and the smaller is x_i, the more strings it adds to L, we have $k = 2$, $x_1 = n-1$ and $x_2 = n$. The greatest number that does not equal $i(n-1) + jn$ for $i \geq 1, j \geq 0$ is $(n-1)^2 - 1$. It follows that $\operatorname{nsc}(L^c) \leq (n-1)^2$. For tightness, let A be the unary NFA with n states shown in Fig. 3. Then $L(A) = a^{n-1}(a^{n-1})^*(a^n)^*$. Hence the complement $L(A)^c$ is finite with the longest string $a^{(n-1)^2-1}$, and the minimal NFA for $L(A)^c$ needs $(n-1)^2$ states. □

Now consider the deterministic case. Every unary one-cycle-free-path DFA has single cycle and single final state. Let $A = (\{0, 1, \dots, n-1\}, \{a\}, \delta, 0, \{f\})$ be a unary one-cycle-free-path DFA where $\delta(i, a) = i + 1$ for $0 \leq i \leq n-2$ and $\delta(n-1, a) = t$; $t \leq f$. Then the number of states in the cycle is $c = n - t$. Hence $L(A) = a^f(a^c)^*$. Any language of this form is union-free.

Theorem 4.3 *Let $m, n \geq 2$. Let K, L be unary union-free languages with $\operatorname{sc}(K) = m$ and $\operatorname{sc}(L) = n$. Then*

(a) $\operatorname{sc}(K \cap L) \leq mn$, *and this bound is tight if* $\gcd(m, n) = 1$;

(b) $\operatorname{sc}(K \cup L) \leq mn$, *and this bound is tight if* $\gcd(m, n) = 1$;

(c) $\operatorname{sc}(KL) \leq mn$, *and this bound is tight if* $m \neq n$;

(d) $\operatorname{sc}(L^2) \leq 2n - 1$, *and this bound is tight;*

(e) $\operatorname{sc}(L^*) \leq (n-1)^2 + 1$, *and this bound is tight;*

Proof.

(a) The upper bound mn is met by the union-free languages $K = (a^m)^*$ and $L = (a^n)^*$ since $K \cap L = (a^{mn})^*$.

(b) The upper bound mn is met by the languages $K = (a^m)^*$ and $L = (a^n)^*$ also for union.

(c) The upper bound mn is proven in [12, Theorem 5.5] and met by the union-free languages $K = a^{m-1}(a^m)^*$ and $L = a^{n-1}(a^n)^*$, as is proven in [12, Theorem 5.4].

(d) The upper bound $2n - 1$ is proven in [10, Theorem 3] and met by the union-free language $L = a^{n-1}a^*$ since the longest string not in L^2 is a^{2n-3}.

(e) The tight bound $(n-1)^2 + 1$ is proven in [12, Theorem 5.3] by the union-free language $L = a^{n-1}(a^n)^*$, since the longest string not in L^* is $a^{n(n-2)}$. □

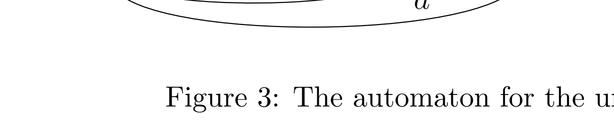


Figure 3: The automaton for the union-free language $a^{n-1}(a^{n-1})^*(a^n)^*$.

Now we look on the star-free languages. Every finite language and every co-finite language is star-free. Since every minimal DFA for a star-free language is permutation-free, and every unary DFA with cycle of length at least two is not permutation-free, unary star-free languages are accepted only by DFAs whose only cycle is a loop. Such automata represent either finite or co-finite languages, so every unary star-free language is either finite or co-finite. On the other hand, the language $\{a, b\}^*a$ is star-free, but it is neither finite nor co-finite.

The unary star-free languages were not considered in [5], so we prove some results here. Every finite language L has the longest accepted string a^ℓ. Then $\ell \leq \operatorname{nsc}(L) - 1$ since the NFA needs $\ell + 1$ states to read the string a^ℓ. Similarly, every co-finite language L has the longest rejected string a^ℓ. Then $\ell \leq \operatorname{nsc}(L) - 2$ since the NFA needs $\ell + 1$ states to reach a nonfinal state by the string a^ℓ and one more final state to accept all longer strings.

Theorem 4.4 *Let $m, n \geq 1$. Let K, L be unary star-free languages with $\operatorname{nsc}(K) = m$ and $\operatorname{nsc}(L) = n$. Then*

(a) $\operatorname{nsc}(K \cap L) \leq \max\{m, n\}$*, and this bound is tight;*
(b) $\operatorname{nsc}(K \cup L) \leq \max\{m, n\}$*, and this bound is tight;*
(c) $\operatorname{nsc}(KL) \leq m + n - 1$*, and this bound is tight;*
(d) $\operatorname{nsc}(L^2) \leq 2n - 1$*, and this bound is tight;*
(e) $\operatorname{nsc}(L^+) \leq n$*, and this bound is tight;*
(f) $\operatorname{nsc}(L^*) \leq n$*, and this bound is tight;*
(g) $\operatorname{nsc}(L^c) \in \Theta(n^2)$.

Proof.

(a) Let K and L be unary co-finite languages, with longest string not in K be a^k and longest string not in L be a^ℓ. Let $k \leq \ell$. Then the longest string not in $K \cap L$ is a^ℓ, so $\operatorname{nsc}(K \cap L) = \max\{m, n\}$. The witness languages meeting this bound are $K = a^{m-1}a^*$ and $L = a^{n-1}a^*$.
(b) Let K and L be unary finite languages, with longest string in K be a^k and longest string in L be a^ℓ. Let $k \leq \ell$. Then the longest string in $K \cup L$ is a^ℓ, so $\operatorname{nsc}(K \cup L) = \max\{m, n\}$. The witness languages meeting this bound are $K = a^{m-1}$ and $L = a^{n-1}$.
(c) By Lemma 4.1, the upper bound is $m + n - 1$. It is met by the co-finite languages $K = a^{m-1}a^*$ and $L = a^{n-1}a^*$ since $KL = a^{m+n-2}a^*$.
(d) By Lemma 4.1, the upper bound is $2n - 1$. It is met by the co-finite language $L = a^{n-1}a^*$.
(e) The upper bound n for positive closure is met by the co-finite language $L = a^{n-1}a^*$.
(f) The upper bound for Kleene closure is n since $L^* \setminus L$ is finite for every co-finite L. It is met by the unary co-finite language $L = a^{n-1}a^*$.
(g) If we transform a unary NFA with n states to the Chrobak normal form, we get a tail with at most $n^2 - 2$ states and disjoint cycles with at most $n - 1$ states ([3, Theorem 3.5]). Since every minimal NFA in Chrobak normal form for a co-finite language has single cycle of length one (a loop), total number of states is $n^2 - 1$. While this NFA is also a complete DFA, it gives us the upper bound $O(n^2)$ on complement of co-finite languages.
Let $L = a^* \setminus \{a^n\}$. As shown in [7], $\operatorname{nsc}(L)$ is in $\Theta(\sqrt{n})$. Next, we have $L^c = a^n$, so $\operatorname{nsc}(L^c) = n + 1$. This gives the lower bound $\Omega(n^2)$. Hence the bound $\Theta(n^2)$ is asymptotically tight.

□

Lemma 4.5 (cf. [1], Theorem 6) *Let $m, n \geq 1$. Let K, L be unary star-free languages with $\operatorname{sc}(K) = m$ and $\operatorname{sc}(L) = n$. Then*

(a) $\operatorname{sc}(K \cap L) \le \max(m, n)$;

(b) $\operatorname{sc}(K \cup L) \le \max(m, n)$;

(c) $\operatorname{sc}(KL) \le m + n - 1$;

(d) $\operatorname{sc}(L^2) \le 2n - 1$;

(e) $\operatorname{sc}(L^*) \le 2$ *if* $n = 1$; $\operatorname{sc}(L^*) \le n$ *if* $2 \le n \le 5$; $\operatorname{sc}(L^*) \le n^2 - 7n + 13$ *if* $n \ge 6$.

All these bounds are tight.

Proof. The bounds for boolean operations, concatenation, and Kleene closure are proven in [1]. For squaring, the upper bound proven in [10] is met by the star-free language $L = a^{n-1}a^*$. □

5. Conclusions

We investigated the nondeterministic and deterministic state complexity of basic operations on union-free and star-free languages. Since most results were known for arbitrary alphabet, we focused on the unary case. The tight upper bounds for intersection, union, positive closure, and Kleene closure on unary union-free languages are the same as for unary regular languages in both nondeterministic and deterministic case. For unary star-free languages, the upper bounds are different, especially for boolean operations, but they are the same in both deterministic and nondeterministic case except for complementation and closures. The following table provides the complexities of operations on the examined classes of unary languages. The second table shows the tight bounds for complexities of operations on examined classes; the witness languages are on an alphabet of size $|\Sigma|$. The lower bound of deterministic complexity of squaring on star-free languages remains an open problem.

	Unary union-free	Unary star-free	Unary regular
$\operatorname{nsc}(K \cap L)$	mn ; $\gcd(m,n) = 1$	$\max\{m,n\}$	mn ; $\gcd(m,n) = 1$
$\operatorname{nsc}(K \cup L)$	$m+n+1$; $m \ne kn$	$\max\{m,n\}$	$m+n+1$; $m \ne kn$
$\operatorname{nsc}(KL)$	$m+n-1$	$m+n-1$	$\ge m+n-1$
$\operatorname{nsc}(L^2)$	$2n-1$	$2n-1$	$\ge 2n-1$
$\operatorname{nsc}(L^+)$	n	n	n
$\operatorname{nsc}(L^*)$	$n+1$	n	$n+1$
$\operatorname{nsc}(L^c)$	$(n-1)^2$	$\Theta(n^2)$	$2^{\Theta(\sqrt{n \log n})}$
$\operatorname{sc}(K \cap L)$	mn; $\gcd(m,n) = 1$	$\max\{m,n\}$	mn ; $\gcd(m,n) = 1$
$\operatorname{sc}(K \cup L)$	mn; $\gcd(m,n) = 1$	$\max\{m,n\}$	mn ; $\gcd(m,n) = 1$
$\operatorname{sc}(KL)$	mn ; $\gcd(m,n) = 1$	$m+n-1$	mn ; $\gcd(m,n) = 1$
$\operatorname{sc}(L^2)$	$2n-1$	$2n-1$	$2n-1$
$\operatorname{sc}(L^*)$	$(n-1)^2+1$	$n^2-7n+13$; $n \ge 6$ n ; $2 \le n \le 5$ 2 ; $n = 1$	$(n-1)^2+1$

	Union-free	$\|\Sigma\|$	Star-free	$\|\Sigma\|$	Regular	$\|\Sigma\|$
$\mathrm{nsc}(K \cap L)$	mn	2	mn	2	mn	2
$\mathrm{nsc}(K \cup L)$	$m+n+1$	2	$m+n+1$	2	$m+n+1$	2
$\mathrm{nsc}(KL)$	$m+n$	2	$m+n$	2	$m+n$	2
$\mathrm{nsc}(L^2)$	$2n$	2	$2n$	2	$2n$	2
$\mathrm{nsc}(L^+)$	n	1	n	1	n	1
$\mathrm{nsc}(L^*)$	$n+1$	1	$n+1$	2	$n+1$	1
$\mathrm{nsc}(L^R)$	n	1	$n+1$	2	$n+1$	2
$\mathrm{nsc}(L^c)$	2^n	3	2^n	2	2^n	2
$\mathrm{sc}(K \cap L)$	mn	2	mn	2	mn	2
$\mathrm{sc}(K \cup L)$	mn	2	mn	2	mn	2
$\mathrm{sc}(KL)$	$m2^n - 2^{n-1}$	2	$m2^n - 2^{n-1}$; $n \geq 3$	4	$m2^n - 2^{n-1}$	2
$\mathrm{sc}(L^2)$	$n2^n - 2^{n-1}$	2	?	?	$n2^n - 2^{n-1}$	2
$\mathrm{sc}(L^*)$	$2^{n-1} + 2^{n-2}$	2	$2^{n-1} + 2^{n-2}$	4	$2^{n-1} + 2^{n-2}$	2
$\mathrm{sc}(L^R)$	2^n	2	$2^n - 1$	$n-1$; $n \geq 3$ n ; $n \leq 2$	2^n	2

References

[1] J. A. BRZOZOWSKI, B. LIU, Quotient complexity of star-free languages. *Int. J. Found. Comput. Sci.* 23 (2012) 6, 1261–1276.
`http://dx.doi.org/10.1142/S0129054112400515`

[2] M. DOMARATZKI, A. OKHOTIN, State complexity of power. *Theoret. Comput. Sci.* 410 (2009) 24-25, 2377–2392.
`http://dx.doi.org/10.1016/j.tcs.2009.02.025`

[3] V. GEFFERT, Magic numbers in the state hierarchy of finite automata. *Inf. Comput.* 205 (2007) 11, 1652–1670.
`http://dx.doi.org/10.1016/j.ic.2007.07.001`

[4] M. HOLZER, M. KUTRIB, Nondeterministic descriptional complexity of regular languages. *Int. J. Found. Comput. Sci.* 14 (2003) 6, 1087–1102.
`http://dx.doi.org/10.1142/S0129054103002199`

[5] M. HOLZER, M. KUTRIB, K. MECKEL, Nondeterministic state complexity of star-free languages. In: *CIAA 2011, Blois, France, 2011. Proceedings*. 2011, 178–189.
`http://dx.doi.org/10.1007/978-3-642-22256-6_17`

[6] G. JIRÁSKOVÁ, T. MASOPUST, Complexity in union-free regular languages. *Int. J. Found. Comput. Sci.* 22 (2011) 7, 1639–1653.
`http://dx.doi.org/10.1142/S0129054111008933`

[7] G. JIRÁSKOVÁ, P. MLYNÁRČIK, Complement on prefix-free, suffix-free, and non-returning NFA Languages. In: *DCFS 2014, Turku, Finland, 2014. Proceedings*. 2014, 222–233.
`http://dx.doi.org/10.1007/978-3-319-09704-6_20`

[8] G. JIRÁSKOVÁ, J. ŠEBEJ, Reversal of binary regular languages. *Theoret. Comput. Sci.* 449 (2012), 85–92.
`http://dx.doi.org/10.1016/j.tcs.2012.05.008`

[9] B. NAGY, Union-free regular languages and 1-cycle-free-path automata. *Publ. Math. Debrecen* 68 (2006) 1-2, 183–197.

[10] N. RAMPERSAD, The state complexity of L^2 and L^k. *Inf. Process. Lett.* 98 (2006) 6, 231–234.
`http://dx.doi.org/10.1016/j.ipl.2005.06.011`

[11] M. P. SCHÜTZENBERGER, On finite monoids having only trivial subgroups. *Information and Control* 8 (1965) 2, 190–194.
`http://dx.doi.org/10.1016/S0019-9958(65)90108-7`

[12] S. YU, Q. ZHUANG, K. SALOMAA, The state complexities of some basic operations on regular languages. *Theoret. Comput. Sci.* 125 (1994) 2, 315–328.
`http://dx.doi.org/10.1016/0304-3975(92)00011-F`

JUMPING AND PUMPING LEMMAS AND THEIR APPLICATIONS

Grzegorz Madejski

Institute of Informatics,
Faculty of Mathematics, Physics and Informatics,
University of Gdańsk,
80-308 Gdańsk, Poland
gmadejsk@inf.ug.edu.pl

Abstract
Jumping grammars, as opposed to the well known classical grammars, use a jumping derivation mode which works in a discontinuous way. By applying a rule $\alpha \to \beta$, we remove α from the sentential form and place β in an arbitrary position in the string. This differs from the classical approach where the position of α and β is the same.

This paper presents some initial results of the author's research on the generative power of such grammars. With the aid of lemmas utilising some jumping and pumping techniques, an infinite hierarchy of jumping languages with respect to the length of the rules is constructed. It is also shown that jumping linear languages are strictly contained within the class of jumping context-free languages.

1. Introduction

Consider a piece of information encoded as a string, consisting of substrings that could be read in an arbitrary order. To process such information our algorithm should parse all possible permutations of the substrings. The task is even more challenging if the strings can be inserted into one another or even shuffled together.

There are many areas of science where such data is expected to be found: information processing, concurrency theory, natural language processing, bioinformatics (such as DNA computing).

Formal grammars with an adequate modification should handle this task. Many attempts were made last years. For example, context-free grammars extended with permutation rules were considered in [6, 7, 10], where their potential to generate languages with relatively free word order was shown. Recently, jumping grammars were introduced [5]. Instead of reordering symbols, these allow to derive the word in a discontinuous way by means of jumping. By applying a rule $\alpha \to \beta$, we remove α from the sentential form and place β in an arbitrary position in the string. This differs from the classical approach where the position of α and β is the same.

Jumping grammars stem from the research on jumping finite automata introduced in [8] and later investigated in [2, 3, 9, 11]. Indeed, jumping grammars are a generalisation of jumping automata, the same way as classical grammars generalise finite automata.

This paper is organised as follows. In Section 2, we provide all the necessary preliminaries. In Section 3, we present the results on the generative power. In Subsection 3.1, an infinite hierarchy of jumping monotonous languages with respect to the length of the rules (defined as *span*) is constructed. In Subsection 3.2, it is shown that jumping linear languages are strictly contained within the class of jumping context-free languages (it was stated as an open problem in [5]). In Section 4, some final remarks and prospects for future research are given.

2. Preliminaries

We assume that the reader is familiar with the basic concepts of the formal language theory, in particular the classes of the Chomsky hierarchy (see [4]). Let $u \in T^*$ be a word and $a \in T$ a letter. We denote $|u|$ as the length of u or the number of all letters, $|u|_a$ as the number of occurrences of letter a in u.

Before we define jumping grammars let us present a notion of a jumping automaton [2, 3].

Definition 2.1 *A* general jumping finite automaton *(GJFA for short) is a quintuple $A = (Q, T, R, q_0, F)$ where Q is a finite set of states, T is a set of terminal symbols (letters) or alphabet, $R \subseteq Q \times T^* \times Q$ is a finite relation, $q_0 \in Q$ is a start state, $F \subseteq Q$ is a set of final states. The elements of R of the form (q, w, p) are called rules and are denoted as $qw \to p$.*

A GJFA, as opposed to classical finite automata, uses a different computation step relation. Let $x, y, x', z' \in T^*$, $q, p \in Q$. $xqy \curvearrowright x'pz'$ if and only if there exist $qw \to p \in R$ and $z \in T^*$ such that $y = wz$, $xz = x'z'$. The language accepted by A is $L(A) = \{w \in T^* : \exists_{u,v \in T^*} \exists_{f \in F}\ w = uv \wedge uq_0v \curvearrowright^* f\}$, where $\curvearrowright^*$ is the transitive-reflexive closure of $\curvearrowright$. A class of all such languages is denoted by $\mathcal{GJFA}$.

We now present a few definitions regarding grammars. Similarly to GJFAs, jumping grammars will be deriving words with a jumping derivation mode [5].

Definition 2.2 *A* jumping grammar *(JG for short) is a quadruple $G = (N, T, P, S)$ where N is a set of non-terminal symbols, T is a set of terminal symbols (letters) or alphabet, $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ is a finite relation and $S \in N$ is a start symbol. The elements of P of the form (x, y) are called rules and are denoted as $x \to y$.*

A jumping grammar uses a jumping derivation step relation to derive words. Let $u, v \in (N \cup T)^*$, $u \notin T^*$. $u \Rightarrow v$ if and only if there exists $x \to y \in P$, $w, z, w', z' \in (N \cup T)^*$ such that $u = wxz, v = w'yz'$ and $wz = w'z'$.

Definition 2.3 *Let $G = (N, T, P, S)$ be a JG. A language generated by G is a set*

$$L(G) = \{x \in T^* : S \Rightarrow^* x\}$$

where $\Rightarrow^*$ is the transitive-reflexive closure of $\Rightarrow$. We say that $L(G)$ is a jumping language *and we denote the class of all jumping languages by $\mathfrak{JL}$.*

Definition 2.4 *A rule is:*

- monotonous *if it is of the form $x \to y$ where $|x| \leq |y|$,*
- context-sensitive *if it is of the form $uXw \to uvw$ where $u, v, w \in (N \cup T)^*$, $v \neq \lambda$, $X \in N$,*
- context-free *if it is of the form $X \to w$ where $.w \in (N \cup T)^+$, $X \in N$,*
- linear *if it is of the form $X \to uYv \mid w$ where $u, v \in T^*$, $w \in T^+$, $X, Y \in N$,*
- right-linear *if it is of the form $X \to uY \mid w$ where $u \in T^*$, $w \in T^+$, $X, Y \in N$,*
- regular *if it is of the form $X \to aY \mid a$, where $a \in T$, $X, Y \in N$.*

If a jumping grammar has rules only of one type, then its name is changed accordingly. Therefore, we have jumping monotonous grammars, jumping context-sensitive grammars and so on. For short, we denote them as JMGs, JCSGs, JCFGs, JLGs, JRLGs, JRGs. They generate jumping monotonous languages, jumping context-sensitive languages, etc. The corresponding classes of languages are denoted as $\mathfrak{JML}$, $\mathfrak{JCSL}$, $\mathfrak{JCFL}$, $\mathfrak{JLL}$, $\mathfrak{JRLL}$, $\mathfrak{JRL}$.

Definition 2.5 *We say that a jumping grammar $G = (N, T, P, S)$ is of span m iff every rule $x \to y \in P$ satisfies $|x| \leq m$, $|y| \leq m$. The language $L(G)$ is then called a language of span m and all such languages denoted as $\mathfrak{JL}(m)$.*

The definition can be extended to all the subclasses of $\mathfrak{JL}$. Therefore, we can speak of jumping context-free languages of span m ($\mathfrak{JCFL}(m)$) and so on.

In [5], the following hierarchy was established.

$$\mathfrak{JRL} \subsetneq \mathfrak{GJFA} = \mathfrak{JRLL} = \mathfrak{JLL} \subseteq \mathfrak{JCFL} \subsetneq \mathfrak{JCSL} \subseteq \mathfrak{JML} \subsetneq \mathfrak{JL}.$$

We see the lack of two strictness results, one of which is presented in Subsection 3.2.

3. Results

3.1. Infinite Hierarchy of Jumping Monotonous Languages

In this subsection, we formulate a jumping lemma and use it to establish an infinite hierarchy of jumping monotonous languages.

Simply put, this lemma shows that if one word can be derived using the rules of grammar G, then other words can be derived too, provided we jump into a different position. The last rule is the best to study, as it always has a string of terminals on the right-hand side.

It should be noted that the lemma generalises some already known proving techniques [5, 8]. However, for the sake of more advanced lemmas and proofs in the next subsection, it is a good idea to study it first.

Lemma 3.1 (Jumping Lemma for $\mathcal{JML}(m)$) *Let $L \in \mathcal{JML}(m)$. For an arbitrary word $w \in L$, $|w| > m$, there exists a partitioning $w = xyz$, such that:*

- $0 < |y| \leq m$,
- $yxz \in L$ *and* $xzy \in L$.

Proof. Let $L \in \mathcal{JML}(m)$. Then, there exists a JMG $G = (N, T, P, S)$ of span m such that $L = L(G)$. If $w \in L$ and $|w| > m$, then the derivation of w in G must consist of at least two steps. The last derivation step uses a rule that has only terminal symbols on the right-hand side. Suppose this rule is $\alpha \to y$ where $\alpha \in (N \cup T)^+ - T^*$, $y \in T^+$, $|\alpha| \leq |y| \leq m$. Suppose now that $w = xyz$. Then the derivation of w in G is of the form:

$$S \Rightarrow ... \Rightarrow x'\alpha z' \Rightarrow xyz$$

where $x'z' = xz \neq \lambda$, $y \geq 1$. We see that the words yxz and xzy can also be derived in G:

$$S \Rightarrow ... \Rightarrow x'\alpha z' \Rightarrow yxz$$

$$S \Rightarrow ... \Rightarrow x'\alpha z' \Rightarrow xzy$$

The derivation of yxz and xzy is the same as of w except for the last derivation step. The jumping position is different. □

The lemma can be used to prove incomparability results. For example, to show that regular languages are not in $\mathcal{JML}$, we could consider a language a^*b^* and a word $w = a^m b^m$. This result was already established in [5].

The jumping lemma can also be used to establish an infinite hierarchy of monotonous jumping languages with respect to their span. A similar hierarchy was studied in [8], where it was shown that there exist languages accepted by general jumping finite automata of degree m that cannot be accepted by general jumping finite automata of degree $m-1$. A degree of automaton is the maximal number of terminal symbols that can be added in one computation step. Therefore, the span of grammars in this paper is a generalisation of the degree of automata. Since $\mathcal{GJFA} = \mathcal{JRLL}$, our result is also a generalisation of the construction of the hierarchy of GJFAs in [8].

Lemma 3.2 *A one-word language $L = \{a^m b\}$ is in the class $\mathcal{JML}(m+1)$, but not in the class $\mathcal{JML}(m)$.*

Proof. L can be generated with a grammar with only one rule $S \to a^m b$ of span $m+1$. Thus, $L \in \mathcal{JML}(m+1)$.

We assume $L \in \mathcal{JML}(m)$ and take the only possible word in L, which is $w = a^m b$. By Lemma 3.1, there exists a partitioning $w = xyz$, where $0 < |y| \leq m$ and $yxz \in L$, $xzy \in L$. If $y = a^k$, $0 < k \leq m$, then the word $xzy = a^{m-k}ba^k \notin L$. If $y = a^k b$, $0 \leq k < m$, then the word $yxz = a^k ba^{m-k} \notin L$. In both possible cases we reach a contradiction. Therefore $L \notin \mathcal{JML}(m)$. □

Theorem 3.3 *$\mathscr{JML}(m) \subsetneq \mathscr{JML}(m+1)$, where $m \geq 1$.*

Proof. The inclusion of the classes follows from their definition. The strictness of the inclusion follows from Lemma 3.2. □

3.2. Jumping Right-Linear Languages versus Jumping Context-Free Languages

It was shown in [5] that $\mathscr{JLL} = \mathscr{JRLL} = \mathscr{GJFA}$. By definition, they are subsets of $\mathscr{JCFL}$, but it was an open problem, whether the inclusion is proper. In this subsection, we formulate a second jumping lemma and use it to show that $\mathscr{JRLL} \subsetneq \mathscr{JCFL}$.

For convenience, we show that we can eliminate rules of the form $X \to Y$ (X, Y are non-terminals) from any JRLG. We shall call them unit rules.

Lemma 3.4 *Let $G = (N, T, P, S)$ be a JRLG. There exists a JRLG $G' = (N, T, P', S)$ such that $L(G) = L(G')$ and P' has no unit rules.*

Proof. The construction of grammar G' based on G is done in the same as in the algorithm for eliminating unit rules, used to acquire grammars in Chomsky normal form (see [4]). Despite having a different derivation mode, proving that $L(G) = L(G')$ is also analogous. □

Example 3.5 *Let $G = (\{S, X, Y, Z\}, \{a, b, c, d\}, P, S)$ be a JRLG of span $m = 3$, where P contains the rules:*

$$S \to aaX, \quad X \to aaY, \quad Y \to bbZ, \quad Z \to ccY, \quad Z \to ddd.$$

We derive a word $w = bbaabbaaccddd$ in G:

$$S \Rightarrow aaX \Rightarrow aaaaY \Rightarrow aabbZaa \Rightarrow aabbaaccY \Rightarrow bbZaabbaacc \Rightarrow bbaabbaaccddd = w,$$

It is easy to see that if the number of derivation steps is greater than the number of non-terminals, then there are two sentential forms containing the same non-terminal symbol. In our example, we have $n = |\{S, X, Y, Z\}| = 4$. Our derivation of w consists of 7 steps and contains two sentential forms with Y: $aaaaY$, $aabbaaccY$. There are two rules used between them $Y \to bbZ, Z \to ccY$, that added subwords $w_1 = bb$, $w_2 = cc$ to the derived word. Similarly to the pumping lemma for regular languages, we can copy the words w_1, w_2 any number of times. The jumping derivation mode lets us put them in any position in the derived string, for example at the end:

$$S \Rightarrow aaX \Rightarrow aaaaY \Rightarrow aabbZaa \Rightarrow aabbaaccY \Rightarrow aabbaacc\underline{bb}Z \Rightarrow aabbaacc\underline{bbcc}Y \Rightarrow$$

$$\Rightarrow bbZaabbaacc\underline{bbcc} \Rightarrow bbaabbaaccddd\underline{bbcc} = ww_1w_2,$$

Pumping for words derived in 4 steps may be impossible, for example:

$$S \Rightarrow aaX \Rightarrow aaaaY \Rightarrow aabbZaa \Rightarrow aabbaaddd.$$

We see that the maximal length of the words generated in 4 steps is $3 \cdot 2 + 3 = 9$. *For an arbitrary grammar of span* m *and* n *non-terminal symbols, this number is* $(n-1)(m-1)+m = n(m-1)+1$. *Words of greater length must contain a loop in derivation.*

We now extend the above investigations to any jumping right-linear languages.

Lemma 3.6 (Jumping and Pumping Lemma for $\mathscr{JRLL}(m)$) *Let* $G = (N, T, P, S)$ *be a JRLG of span* m *without unit rules and* $|N| = n$. *Let* $w \in L(G)$, $|w| > r$ *where* $r = n(m-1)+1$. *Then:*

- *The derivation of* w *in* G *consists of at least* $n+1$ *steps.*
- *Let* $X_0 \to w_1X_1, X_1 \to w_2X_2, ..., X_{n-1} \to w_nX_n, X_n \to w_{n+1}$ *be the rules used in the last* $n+1$ *steps of the derivation of* w *(in the given order), where* $0 < |w_i| < m$ *for all* $1 \le i \le n$ *and* $0 < |w_{n+1}| \le m$. *Then, there exist indices* $1 \le k \le l \le n$ *such that for any permutation*
$$\sigma = \begin{pmatrix} k & k+1 & \cdots & l \\ \sigma(k) & \sigma(k+1) & \cdots & \sigma(l) \end{pmatrix}$$
words $w_1 = w_{\sigma(k)}w_{\sigma(k+1)}\cdots w_{\sigma(l)}w$ *and* $w_2 = ww_{\sigma(k)}w_{\sigma(k+1)}\cdots w_{\sigma(l)}$ *are also in* $L(G)$.

Proof. The first point of the lemma is easy to prove, see Example 3.5. If there are at least $n+1$ steps, then let the rules used in the last $n+1$ steps be the following (in the given order):

$$X_0 \to w_1X_1, X_1 \to w_2X_2, ..., X_{n-1} \to w_nX_n, X_n \to w_{n+1}.$$

Among the $n+1$ non-terminal symbols $X_0, ..., X_n$ there exist two which are the same. Let $X_{k-1} = X_l$ for some $1 \le k \le l \le n$. Then, the rules $X_{k-1} \to w_kX_k, ..., X_{l-1} \to w_lX_{k-1}$ allow us to pump the words $w_k, ..., w_l$. Due to the jumping derivation mode, we can put them at the beginning or the end of the word, in any order represented by a permutation σ (see Figure 1).

$$\begin{tikzcd}
S \arrow[r] & \ldots \arrow[r] & \alpha X_{k-1}\beta \arrow[d, "\text{J\&P}"] & & \alpha\beta w_l X_{k-1} w_{l-1}\ldots w_k \arrow[r] & \ldots \arrow[r] & w w_l w_{l-1}\ldots w_k \\
& & \alpha\beta w_k X_k \arrow[d, "\text{J\&P}"] & & \alpha\beta w_{l-1} X_{l-1} w_{l-2}\ldots w_{k+1} w_k \arrow[u, "\text{J\&P}"'] & & \\
& & \alpha\beta w_{k+1} X_{k+1} w_k \arrow[r] & \ldots \arrow[r, "\text{J\&P}"'] & \alpha\beta w_{l-2} X_{l-2} w_{l-3}\ldots w_{k+1} w_k \arrow[u, "\text{J\&P}"'] & &
\end{tikzcd}$$

Figure 1: An example derivation of a pumped word. The steps where jumping and pumping was used are marked with J&P. A permutation $\sigma(i) = l + k - i$ was used to order the pumped words $w_k, ..., w_l$ at the end of the derived word (after w).

□

The lemma can be generalised. We see that we use pumping only once, but the loop could be repeated any number of times. We also give a big restriction on the jumping, as we only consider jumps to the beginning or the end. However, such a restricted version of the lemma suffices for the purpose of this paper.

We now consider an important language and show that it generates words of a specific form.

Example 3.7 *Let L be a language generated by the following JCFG*

$$G = (\{S, X\}, \{a, b, c, d\}, \{S \to adSX, S \to adX, X \to bc\}, S).$$

By definition $L \in \mathfrak{JCFL}$.

We state and briefly prove some important properties of language L from the above example.

Lemma 3.8 *L has the following properties:*

1. *Words from L have the same number of letters a, b, c and d.*
2. *Words of the form $a^r b^r c^r d^r$ where $r > 0$ are in L.*
3. *Words from L cannot contain any letter a after the last letter d.*
4. *Words from L cannot contain any letter d before the first letter a.*

Proof.

1. We see that in an arbitrary derivation a pair ad always comes with a symbol X, which is later substituted by bc.
2. We use the jumping rule $S \to adSX$ $r-1$-times, always jumping in between a and d. Then we use $S \to adX$, acquiring

$$S \Rightarrow adSX \Rightarrow aadSXdX \Rightarrow ... \Rightarrow a^r(dX)^r.$$

After that, we use the rule $X \to bc$ r-times firstly jumping between a and d, and afterwards between b and c.

$$S \Rightarrow ... \Rightarrow a^r(dX)^r \Rightarrow a^r bcd(dX)^{r-1} \Rightarrow a^r bbccdd(dX)^{r-2} \Rightarrow ... \Rightarrow a^r b^r c^r d^r.$$

3. and 4. This is due to the form of the rules, in which the letter a is accompanied by letter d to the right.

□

We are now ready to prove the following

Proposition 3.9 *$L \notin \mathfrak{JRLL}$.*

Proof. We assume $L \in \mathfrak{JRLL}$. Then, there exists a JRLG $G = (N, T, P, S)$ of span m, with no unit rules, such that $L = L(G)$. Let $n = |N|$ and $r = n(m-1)+1$. From property 2 from Lemma 3.8, we know that $w = a^r b^r c^r d^r \in L$.

Let $w_k, w_{k+1}, ..., w_l$ be the words from Lemma 3.6, from the derivation of w. We consider the following four cases:

- $\forall_{v \in \{w_k, w_{k+1}, ..., w_l\}}\ |v|_a = 0 \wedge |v|_d = 0$.
  In this case, by pumping and jumping we violate the property 1 from Lemma 3.8. The number of b and c is greater than the number of a and d.

- $\exists_{v\in\{w_k,w_{k+1},\dots,w_l\}}\ |v|_a \neq 0 \wedge |v|_d = 0$.
  We put the pumped v at the end of the word, which leads to a contradiction with the property 3 from Lemma 3.8.
- $\exists_{v\in\{w_k,w_{k+1},\dots,w_l\}}\ |v|_a = 0 \wedge |v|_d \neq 0$.
  We violate the property 4 from Lemma 3.8 by putting the pumped v at the beginning of the derived word.
- $\exists_{v\in\{w_k,w_{k+1},\dots,w_l\}}\ |v|_a \neq 0 \wedge |v|_d \neq 0$.
  Note that a should be before d in v, otherwise it is possible to destroy the property 3. Before the application of the rule containing v, no letter b or c is derived. Otherwise, w would contain a subword with b or c appearing either before both a and d, or after. Thus, all $2r$ letters b and c must be derived during and after the application of the rule with v. This is not possible, as in the last $n+1$ steps, only $n(m-1)+m = r+m-1$ letters can be derived and that is strictly smaller than $2r$.

In all cases, we reached a contradiction which is due to a false assumption that $L \in \mathfrak{JRLL}$. □

Theorem 3.10 *$\mathfrak{JRLL} \subsetneq \mathfrak{JCFL}$.*

Proof. The inclusion of the classes follows from their definition. The strictness of the inclusion follows from Example 3.7 and Proposition 3.9. □

4. Conclusions

In this paper, we defined jumping grammars and languages of span m. Similarly to the families of languages of the Chomsky hierarchy, we defined their jumping counterparts. Then, we showed that $\mathfrak{JML}(m) \subsetneq \mathfrak{JML}(m+1)$ for any $m > 0$. We also proved that jumping linear languages are strictly contained in the class of jumping context-free languages.

$$\mathfrak{JRL} \subsetneq \mathfrak{JRLL} = \mathfrak{JLL} \underset{\text{Thm.3.10}}{\subsetneq} \mathfrak{JCFL} \subsetneq \mathfrak{JCSL} \underset{\text{strict?}}{\subseteq} \mathfrak{JML} \subsetneq \mathfrak{JL}.$$

There are several interesting problems to consider. It is not known if the inclusion $\mathfrak{JCSL} \subseteq \mathfrak{JML}$ is strict. A few other jumping derivations modes can be studied (see [5]). Finally, it would be interesting to compare the class of jumping languages or its subclasses with families generated by other models of grammars, for example permutation languages [6, 7, 10].

5. Acknowledgements

The author wishes to thank anonymous referees for useful tips and corrections in the previous version of this paper.

References

[1] F. DREWES (ed.), *Implementation and Application of Automata - 20th International Conference, CIAA 2015, Umeå, Sweden, August 18-21, 2015, Proceedings*. Lecture Notes in Computer Science 9223, Springer, 2015.
`http://dx.doi.org/10.1007/978-3-319-22360-5`

[2] H. FERNAU, M. PARAMASIVAN, M. L. SCHMID, Jumping Finite Automata: Characterizations and Complexity. In: DREWES [1], 2015, 89–101.
`http://dx.doi.org/10.1007/978-3-319-22360-5_8`

[3] H. FERNAU, M. PARAMASIVAN, M. L. SCHMID, V. VOREL, Characterization and Complexity Results on Jumping Finite Automata. *CoRR* abs/1512.00482 (2015).
`http://arxiv.org/abs/1512.00482`

[4] J. E. HOPCROFT, R. MOTWANI, J. D. ULLMAN, *Introduction to automata theory, languages, and computation - (2. ed.)*. Addison-Wesley series in computer science, Addison-Wesley-Longman, 2001.

[5] Z. KRIVKA, A. MEDUNA, Jumping Grammars. *Int. J. Found. Comput. Sci.* 26 (2015) 6, 709–732.
`http://dx.doi.org/10.1142/S0129054115500409`

[6] G. MADEJSKI, Infinite Hierarchy of Permutation Languages. *Fundam. Inform.* 130 (2014) 3, 263–274.
`http://dx.doi.org/10.3233/FI-2014-992`

[7] G. MADEJSKI, The Membership Problem for Linear and Regular Permutation Languages. In: DREWES [1], 2015, 211–223.
`http://dx.doi.org/10.1007/978-3-319-22360-5_18`

[8] A. MEDUNA, P. ZEMEK, Jumping Finite Automata. *Int. J. Found. Comput. Sci.* 23 (2012) 7, 1555–1578.
`http://dx.doi.org/10.1142/S0129054112500244`

[9] A. MEDUNA, P. ZEMEK, *Regulated Grammars and Automata*. Springer, 2014.
`http://dx.doi.org/10.1007/978-1-4939-0369-6`

[10] B. NAGY, Languages Generated by Context-Free Grammars Extended by Type AB -> BA Rules. *Journal of Automata, Languages and Combinatorics* 14 (2009) 2, 175–186.

[11] V. VOREL, On Basic Properties of Jumping Finite Automata. *CoRR* abs/1511.08396 (2015).
`http://arxiv.org/abs/1511.08396`

EXTENDED UNIFORMLY LIMITED T0L LANGUAGES AND MILD CONTEXT-SENSITIVITY

Suna Bensch$^{(A)}$ Martin Kutrib$^{(B)}$
Andreas Malcher$^{(B)}$

$^{(A)}$Department of Computing Science, Umeå University,
90187 Umeå, Sweden
suna@cs.umu.se

$^{(B)}$Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher}@informatik.uni-giessen.de

Abstract
We study the fixed membership problem for k-uniformly-limited and propagating ET0L *systems (k*ulEPT0L *systems). To this end, the algorithm given in [7] is applied. It follows that* $k$ulEPT0L *languages are parsable in polynomial time. Since* $k$ulEPT0L *languages are semi-linear [1] and* $k$ulEPT0L *systems generate certain non-context-free languages, which capture the non-context-free phenomena occurring in natural languages, this is the last building block to show that* $k$ulEPT0L *languages, for $k \geq 2$, belong to the family of mildly context-sensitive languages.*

1. Introduction

Context-free languages are parsable in polynomial time and there are several membership algorithms based on the concept of dynamic programming for context-free languages [6, 15]. For context-sensitive languages, on the other hand, there are no known polynomial time algorithms for the membership problem. Extended Lindenmayer systems without interaction (E0L systems) can be considered as parallel counterparts of context-free grammars and E0L languages are known to be parsable in polynomial time as well [10]. However, for ET0L languages, that is, languages generated by tabled E0L systems, the membership complexity is NP-complete [12].

Research has studied many language families that lie between "context-free" (or E0L) and "context-sensitive" (or ET0L), (see, for instance [4]). In [7], for example, the authors study the membership complexity for context-free languages generated by context-free grammars which are extended so that context-free productions are applied to a fixed number k of symbols at each derivation step. The authors use results from scheduling theory and dynamic programming to show that the membership for these extended context-free languages is decidable in polynomial time. The authors in [14], for instance, introduced a restricted version of ET0L systems, namely k-uniformly-limited ET0L systems (abbreviated $k$ulET0L systems). In these systems the parallel rewriting mechanism of Lindenmayer systems is limited such that not all symbols in a word w have to be rewritten, but $\min\{k, |w|\}$ symbols, where k is a positive integer. Note that, if $k = 1$, we have a context-free grammar (see [14]). The crucial differences between these two grammar formalisms are, that (i) the extended context-free grammars in [7] are extended with respect to their sequential rewriting mechanisms (that is, from rewriting one symbol to rewriting k symbols) and $k$ulET0L systems are limited with respect that their parallel substitution mechanisms (that is, from substituting all symbols to substituting k symbols), and (ii) in [7] the lengths of all sentential forms (after rewriting the start symbol) are at least k.

In mathematical linguistics, researchers also have been investigating language families that lie between the context-free language family and the context-sensitive language family in the Chomsky hierarchy. A concept that captures such language families is *mild context-sensitivity*. The notion of mild context-sensitivity was first mentioned in [8], where the author proposed that a class of grammars (and their associated languages) modeling the syntax of natural languages should have the following three characteristics. First, it should describe certain non-context-free structures in order to capture the non-context-free phenomena occurring in natural languages. Second, it should have the constant growth property (the constant growth property is obeyed by every semilinear language) and, third, it should be parsable in polynomial time. Note, that the concept mildly context-sensitive captures a *family of families of languages*, not a single language family. For a formal definition of mildly context-sensitive grammar formalisms see [2]. In the literature there have been many investigations of mildly context-sensitive *sequential* grammar formalisms and their languages (see, for instance, [9, 13]). There have been less investigations of mildly context-sensitive *parallel* grammar formalisms. In [1] the author investigates some restricted versions of limited parallel Lindenmayer systems with respect to their mild context-sensitivity.

In this paper, we apply the fixed membership algorithm given in [7] to propagating $k$ulET0L languages (abbreviated $k$ulEPT0L languages). It follows that $k$ulEPT0L languages are parsable in polynomial time. Moreover, it is known that $k$ulEPT0L languages are semilinear [1]. Additionally, $k$ulEPT0L systems generate non-context-free languages, such as $\{\, a^n b^n c^n \mid n \geq 1 \,\}$, $\{\, a^n b^m c^n d^m \mid n, m \geq 1 \,\}$, and $\{\, ww \mid w \in \{a, b\}^+ \,\}$, which capture the non-context-free phenomena occurring in natural languages. From all this, we conclude that $k$ulEPT0L languages, for $k \geq 2$, belong to the family of mildly context-sensitive languages.

2. Definition and Preliminaries

We assume the reader to be familiar with the basic notions of Lindenmayer systems without interaction such as in [11]. In general we have the following conventions: The set of positive integers is denoted by $\mathbb{N}$ and if we want to include 0, we write $\mathbb{N}_0$. The cardinality of a set A is denoted by $\#A$. Let $V = \{a_1, a_2, \ldots, a_n\}$ be some alphabet, where the order of the symbols is fix. By V^+ we denote the set of nonempty words; if the empty word λ is included, then we use the notation V^*. The length of a word w in V^* is the number of letters in w and written as $|w|$. The length of the empty word λ is 0.

An ET0L system is a quadruple $G = (\Sigma, H, \psi, \Delta)$, where Σ is an alphabet, $\psi \in \Sigma^*$ is the axiom, $\Delta \subseteq \Sigma$ is the terminal alphabet, and H is a finite set of finite substitutions from Σ into Σ^*. A substitution h in H is called a *table*. For x in Σ we write $x \to y$ if $y \in h(x)$. By Maxr(G) we denote the length of the longest right-hand side of a production in G. In general, we write $u \underset{G}{\Longrightarrow} w$ if and only if $w \in h(u)$, for u and w in Σ^* and some h in H. If the table should be noted explicitly, we write $u \underset{h}{\Longrightarrow} w$. The reflexive transitive closure of the derivation relation $\underset{G}{\Longrightarrow}$ is denoted by $\overset{*}{\underset{G}{\Longrightarrow}}$.

The language generated by G is $L(G) = \{\, w \in \Delta^* \mid \psi \overset{*}{\underset{G}{\Longrightarrow}} w \,\}$.

An ET0L system G is called *propagating* (EPT0L system, for short) if for all substitutions h in H and all $x \in \Sigma$, we have $\lambda \notin h(x)$.

In a derivation of a $k$ulET0L system at each step of the rewriting process exactly $\min\{k, |w|\}$ symbols of the word w considered have to be rewritten. That is, if $|w| < k$ then all symbols have to be rewritten, but if $|w| \geq k$ then there are $\binom{|w|}{k}$ possibilities to rewrite the word w.

Formally, a *k-uniformly-limited* ET0L *system* G ($k$ulET0L system, for short) ([14]) is a quintuple $G = (\Sigma, H, \psi, \Delta, k)$, where $k \in \mathbb{N}$ and $(\Sigma, H, \psi, \Delta)$ is an ET0L system.

The derivation relation $\underset{G}{\Longrightarrow}$ of a $k$ulET0L system is defined as follows. Let $u, w \in \Sigma^*$ and $h \in H$.

1. If $|u| \geq k$ then $u \underset{G}{\Longrightarrow} w$ if we can write
$$u = v_1 x_1 v_2 x_2 \cdots x_k v_{k+1} \quad\text{and}\quad w = v_1 z_1 v_2 z_2 \cdots z_k v_{k+1}$$
with $x_i \in \Sigma, v_j \in \Sigma^*, z_i \in h(x_i), i = 1, \ldots, k, j = 1, \ldots, k+1$.
2. If $|u| < k$ then $u \underset{G}{\Longrightarrow} w$ if $w \in h(u)$.

A *sentential form* of a $k$ulET0L system $G = (\Sigma, H, \psi, \Delta, k)$ is a word $w \in \Sigma^*$ with $\psi \overset{*}{\underset{G}{\Longrightarrow}} w$. A propagating $k$ulET0L ($k$ulEPT0L, for short) is defined the same way as for ET0L systems.

The authors in [14] introduced the notion of *pseudo-synchronization* for $k$ulET0L grammars. A $k$ulET0L system G is called pseudo-synchronized if for every $a \in \Delta$ and for every $w \in \Sigma^*$ the following holds: if $a \Longrightarrow w$ then $w \notin \Delta^*$; that is, there are no productions of the form $a \to w$, for $a \in \Delta$ and $w \in \Delta^*$.

Theorem 2.1 ([14], Theorem 3.1) *To every kulE(P)T0L system $G = (\Sigma, H, \psi, \Delta, k)$, there exists an equivalent pseudo-synchronized kulE(P)T0L system $G' = (\Sigma', H', S, \Delta, k)$ such that $L(G) = L(G')$.*

In the following we assume a $k$ulEPT0L system to be pseudo-synchronized without explicitly mentioning.

The reader is assumed to be familiar with the basic notions of *trees*, *forests*, *root*, *parent*, *child*, *ancestor*, *descendant*, *internal node* and *leaf*. Let v be a node. The depth of v is its distance from the root of its tree, plus 1. The height of v is the distance to its furthest descendant. Leaves have height zero and roots have depth one. Let F be a forest. The height of F is the maximal height of its roots. By $|F|$ we denote the number of nodes in F and by $\#F$ we denote the number of trees in F. The *bare forest* of F is obtained by deleting all leaves in F; the bare forest of F is denoted by $\mathrm{Bare}(F)$. The *child forest* of F is obtained by deleting all roots from F.

To each derivation of a $k$ulEPT0L system one can associate a *derivation tree* t in a similar fashion as it is done for context-free grammars. Moreover, if a node is labeled with a symbol $a \in \Sigma$, and the label of its children (from left to right) form the word w, then $a \to w$ is a production in a table of the $k$ulEPT0L system. A *derivation forest* is a forest containing a tree for each symbol of the axiom. The roots of the trees are labeled by these symbols. We illustrate some of these notions by the following example.

Example 2.2 (derivation forest) *Let $G = (\Sigma, H, \psi, \Delta, k)$ be the $k$ulEPT0L system with $\Sigma = \{A, B, a, b\}$, $\Delta = \{a, b\}$, $\psi = AB$, $k = 3$, $H = \{h_1, h_2\}$, where the set of tables is given by*

$$h_1 = \{A \to AA, B \to BB, a \to a, b \to b\},$$

$$h_2 = \{A \to a, B \to b, a \to a, b \to b\}.$$

Consider the following derivation, in which we mark the symbols which are rewritten with a dot. Table h_1 is used in the first two derivation steps and then Table h_2 is applied to terminate the derivation:

$$\dot{A}\dot{B} \Longrightarrow \dot{A}\dot{A}\dot{B}B \Longrightarrow AAA\dot{A}\dot{B}B\dot{B} \Longrightarrow \dot{A}\dot{A}\dot{A}abBb \Longrightarrow aa\dot{a}a\dot{b}\dot{B}b \Longrightarrow aaaabbb.$$

The derivation forest for $aaaabbb$ is illustrated in Figure 1. ■

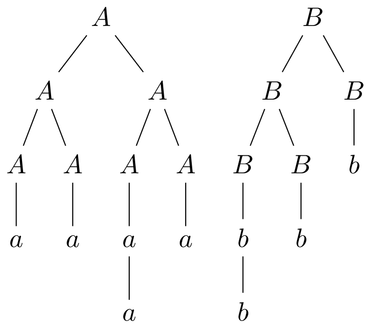


Figure 1: Derivation forest of $aaaabbb$ derived by the $k$ulEPT0L system given in Example 2.2.

In the following we will divide the derivations of a $k$ulEPT0L system into two phases. The sentential forms w in a derivation of a $k$ulEPT0L can be divided into a phase in which each $|w| < k$ and into the phase in which each $|w| \geq k$. In the first phase all symbols have to be rewritten and in the second phase exactly k symbols have to be rewritten. The derivation steps in the first phase correspond to derivation steps in an EPT0L derivation and the second phase is called a *k-derivation* of a $k$ulEPT0L system. Note that in a propagating $k$ulET0L the lengths of the sentential forms do not decrease. In the following we define a k-derivation of a $k$ulEPT0L system and its k-derivation forest.

Definition 2.3 (k-derivation forest) *Let $G = (\Sigma, H, \psi, \Delta, k)$ be a k*ulEPT0L *system. A k-derivation in G is a sequence of words $w_1, \ldots, w_{l+1}$, such that, for all $1 \leq i \leq l$, $w_i \underset{G}{\Longrightarrow} w_{i+1}$ and $|w_i| \geq k$, $1 \leq i \leq l+1$.*

The length of the derivation $w_1, \ldots, w_{l+1}$ is l and the ith step is $w_i \Longrightarrow w_{i+1}$.

*A k-*derivation forest *is a derivation forest that corresponds to a k-derivation.*

Example 2.4 (k-derivation forest) *Note that the derivation forest given in Example 2.2 is not a k-derivation forest, since $|\psi| < k$. Figure 2 illustrates a k-derivation forest for the k-derivation $\dot{A}\dot{A}\dot{B}B \Longrightarrow AAA\dot{A}\dot{B}B\dot{B} \Longrightarrow \dot{A}\dot{A}\dot{A}abBb \Longrightarrow aa\dot{a}a\dot{b}\dot{B}b \Longrightarrow aaaabbb$ as derived by the* $k$ulEPT0L *system given in Example 2.2.* ■

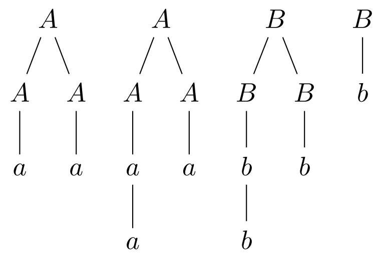


Figure 2: k-derivation forest of $aaaabbb$ derived by the $k$ulEPT0L system given in Example 2.2.

Next we turn to define *schedules* on forests following [7]. Let F be a forest. We assume that there are k processors that correspond to rewriting k symbols in each derivation step. Every node in F is interpreted as a *task*. We assume that all tasks are unit-length (that is, they take the same time to be executed). The parent-child relation in F specifies the precedence constraints (that is, the parent nodes are scheduled before its children nodes). A *k-schedule* is then a sequence of *slots*, where each slot contains up to k tasks. Each slot has a corresponding time unit. Each slot indicates which of the at most k tasks are to be scheduled in the corresponding time unit. Each slot is filled with symbols that occur on the left-hand side of a production in a table $h \in H$ of a given $k$ulEPT0L system $G = (\Sigma, H, \psi, \Delta, k)$. Figure 3 illustrates these notions.

Definition 2.5 (k-schedule [7]) *Let F be a forest and let $k \geq 1$. A k-schedule of F is a function σ mapping the nodes of F onto the set $\{1, \ldots, l\}$, for some $l \leq |F|$, such that*

time $\rightarrow$	1	2	$\dots$	l
processor 1 $\rightarrow$				
processor 2 $\rightarrow$				
$\vdots$				
processor k $\rightarrow$				
	slot 1	slot 2		slot l

Figure 3: An illustration of a k-schedule of length l with k processors. Each cell will be filled with one symbol from an alphabet.

1. $1 \leq \#\sigma^{-1}(i) \leq k$ *for all* $1 \leq k \leq l$,
2. *for each pair of nodes* v_1, v_2 *in* F, *if* v_2 *is a successor of* v_1, *then* $\sigma(v_2) > \sigma(v_1)$.

The length of σ is l and $\sigma^{-1}(i)$ is called the ith slot of σ. The tasks of slot i are scheduled at time i (that is, $\#\sigma^{-1}(i)$ out of the k processors are assigned a task at that time). There are $k - \#\sigma^{-1}(i)$ *idle periods* in slot i (that is, periods in which not all k processors are used). A schedule σ has $p(\sigma)$ idle periods, where

$$p(\sigma) = \sum_{i=1}^{l} (k - \#\sigma^{-1}(i)) = l \cdot k - |F|.$$

A schedule σ is *optimal* for F if there is no schedule σ' of F with $p(\sigma') \leq p(\sigma)$. Note that optimal schedules have minimal length. The number of idle periods of F, denoted $p(F)$, is the number of idle periods in an *optimal schedule* for F.

If $p(\sigma) = 0$ in a schedule σ, then σ is called *perfect*.

1	2	3	4	5
A	A	A	A	a
B	A	B	A	b
	B	B	A	B

1	2	3	4
A	A	A	a
A	B	A	b
B	B	A	B

Figure 4: The table on the left side is a k-schedule, $k = 3$, for the derivation forest given in Figure 1; the schedule is optimal and has one idle period. The table on the right side is a k-schedule, $k = 3$, for the k-derivation forest given in Figure 2; it is a perfect schedule.

Observe that the bare forests of k-derivation forests have perfect k-schedules. If $v_1, \dots, v_k$ are symbols that are rewritten in step i during a k-derivation, then $v_1, \dots, v_k$ occur in the ith slot of the corresponding perfect schedule.

Lemma 2.6 ([7]) *A derivation forest is a k-derivation forest if and only if its bare forest has a perfect schedule.*

Lemma 2.7 *(first and second phase) Let* $G = (\Sigma, H, \psi, \Delta, k)$ *be a* $k$ulEPT0L *system and let* $\psi \underset{G}{\Longrightarrow} w_1 \overset{*}{\underset{G}{\Longrightarrow}} \cdots \underset{G}{\Longrightarrow} w_l \overset{*}{\underset{G}{\Longrightarrow}} \cdots \overset{*}{\underset{G}{\Longrightarrow}} w_n = w$ *be a derivation of* w *in* G, *where* $|\psi| < k$, $|w_i| < k$, *for* $1 \leq i \leq l - 1$ *and* $|w_l| \geq k$.

A word w is in $L(G)$ if and only if there exists a derivation $\psi \overset{*}{\underset{G}{\Longrightarrow}} w$, such that

1. *there is a derivation tree t of w_l from ψ, and*
2. *there is a k-derivation tree s of w from w_l, such that $p(\mathrm{Bare}(s)) = 0$.*

We refer to the derivation from ψ to w_l as the first phase and to the derivation from w_l to w as the second phase. If $|\psi| \geq k$, then the derivation has no first phase.

Proof. All slots in a k-schedule for a k-derivation are filled with tasks and there are no idle periods. □

The following algorithm is for obtaining a *Highest Level First* (HLF) k-schedule for a forest. Roughly speaking, the algorithm builds an HLF k-schedule for a forest F by scheduling the nodes on the longest paths in a tree in F.

Algorithm 2.8 ([7])

1. *If F consists of at least k trees, then $\sigma^{-1}(1)$ contains the roots of the k highest trees (for trees of equal height the choice is arbitrary).*
2. *Otherwise, $\sigma^{-1}(1)$ is the set of all the roots.*
3. *The tail of the schedule is constructed similarly, with the nodes in $\sigma^{-1}(1)$ deleted from F.*

The k-schedules in Figure 4 are not HLF k-schedules. An HLF k-schedule is given in Figure 5.

1	2	3	4
A	A	A	A
A	B	A	b
B	B	a	B

Figure 5: An HLF k-schedule for the k-derivation forest given in Figure 2.

Theorem 2.9 ([3, 5]) *Any HLF schedule for a forest is optimal.*

3. Polynomial Membership Algorithm

We divide the membership testing into the two phases of a derivation of a $k$ulEPT0L system (see Lemma 2.7). The first phase ends when the length of the sentential form is at least k. Since there are at most $(|\Sigma| + 1)^{k-1}$ different sentential forms whose length is less than k and the systems are propagating, testing membership in the first phase can be done in constant time.

For the second phase we use the algorithm in [7] for extended context-free grammars, where k nonterminal symbols are replaced in each derivation step. The algorithm in [7] is bottom-up and keeps track of all the derivation trees deriving subwords of the input word, similar to the CYK algorithm. These derivation trees are parameterized and we obtain a polynomial size characterization. There may be a family of derivation forests for a given word. The parameterized trees are called *frames* and the parameters are the root symbol, the start and end position of the subword in w, the height of the subtree and the number of its nodes.

The algorithm computes then the number of idle periods for a frame collection by using the *median*. In [5] the median was used to present a polynomial time scheduling algorithm for forests and other graphs assuming a constant number of processors. Note that the number of idle periods for various frame collections have to be computed. Intuitively, all those k-derivation trees which are higher than the median are "hard" to schedule, whereas all other k-derivation trees are "easy" to schedule. There are only a polynomial number of frame collections that are "hard" to schedule and this achieves a polynomial time algorithm for solving membership for a constant k and a constant $k$ulEPT0L system.

Definition 3.1 (k-median [7]) *The k-median of a forest F is one plus the height of the kth highest tree of F. If F contains less than k trees, then the median is zero. The k-*high forest *of F is the set of all those trees in F which are strictly higher than the median. The k-*low forest *is the set of the remaining trees.*

The k-high forest and k-low forest of a forest F are denoted by $\mathrm{High}_k(F)$ and $\mathrm{Low}_k(F)$, respectively.

Theorem 3.2 ([7]) *Let F be a forest and σ be a k-schedule for $\mathrm{High}_k(F)$ with q idle periods. Then there is a schedule σ' for the whole forest F, such that:*

1. *if $q \geq |\mathrm{Low}_k(F)|$, then the length of σ' is as most as long as the length of σ;*
2. *if $q < |\mathrm{Low}_k(F)|$, then σ' has idle periods only in its last slot.*

Lemma 3.3 ([7]) *Assume that the HLF schedules for $\mathrm{High}_k(F)$ have q idle periods. Then the HLF schedules for F have*

1. *$q - |\mathrm{Low}_k(F)|$ idle periods if $q \geq |\mathrm{Low}_k(F)|$,*
2. *$-|F| \mod k$ idle periods, otherwise.*

The following lemma is a restatement for $k$ulEPT0L derivations and restricts the length of a derivation for a word w in a $k$ulEPT0L language polynomially in n.

Lemma 3.4 *Let $G = (\Sigma, H, \psi, \Delta, k)$ be a $k$ulEPT0L system and let $w \in \Delta^*$ with $|w| = n$. Then $w \in L(G)$ if and only if there exists a derivation tree t of w from ψ, such that the height of t is at most $f(n) = nk^{2k}(\#\Sigma)^{k(k+1)/2}$ and $|t| \leq nf(n)$.*

The bound on the total number of nodes $|t|$ follows from the fact that in each tree level, there can be at most n nodes.

The following definition is a restatement for $k$ulEPT0L systems and its k-derivation trees. The k-derivation trees are parameterized into *frames*.

Definition 3.5 *Let G be a k*ulEPT0L *and let $w = a_1 \cdots a_n$, where $a_1, \ldots, a_n \in \Delta$ and $|w| \geq k$. A frame R (of w) is a quintuple (A, l, r, h, c), such that $A \in \Sigma$ is the root of R, $1 \leq l \leq r \leq n$, and there is a k-derivation tree t of $a_l \cdots a_r$ from A in G, such that its bare tree has height h and c nodes. If the derivation tree is of height zero, that is, A is a terminal symbol, then $c = 0$ and $h = -1$.*

A tree t as above is called a *frame tree* for R. The height of a frame R is h and the size of R, denoted $|R|$, is c.

An ordered set $\mathcal{R}$ of frames (A, l, r, h, c) is called a *frame collection*, where all A in $\mathcal{R}$ occur on the left-hand side of a production in a table $h \in H$ of a given $k$ulEPT0L system. The height of $\mathcal{R}$ is the maximum of the frame heights in $\mathcal{R}$ and the size of $\mathcal{R}$ is the sum of the sizes of the frames in $\mathcal{R}$. If F is a forest, such that the ith tree in F is a frame tree for the ith frame in $\mathcal{R}$, for $1 \leq i \leq \#F = \#\mathcal{R}$, then F is called a *frame forest of* $\mathcal{R}$ (see Example 3.6 for an example).

Example 3.6 *The forest in Figure 2 is a frame forest for the frame collection $\mathcal{R}$:*

$$\mathcal{R} = \{(A, 1, 4, 3, 8), (B, 5, 7, 3, 6)\}.$$

■

The notions of k-median, k-high collection (k-high forest), and k-low collection (k-low forest) are defined similarly for frame collections. In particular, the number of idle periods for a frame collection is given by

$$p(\mathcal{R}) = \min\{p(\mathrm{Bare}(F)) \mid F \text{ is a frame forest of } \mathcal{R}\}.$$

The following is a restatement of Lemma 2.7 for frames.

Lemma 3.7 *Let $G = (\Sigma, H, \psi, \Delta, k)$ be a k*ulEPT0L *system and let $\psi \underset{G}{\Longrightarrow} w_1 \overset{*}{\underset{G}{\Longrightarrow}} \cdots \overset{*}{\underset{G}{\Longrightarrow}} w_l \overset{*}{\underset{G}{\Longrightarrow}} \cdots \overset{*}{\underset{G}{\Longrightarrow}} w_n = w$ be a derivation of w in G, where $|\psi| < k$, $|w_i| < k$, for $1 \leq i \leq l-1$ and $|w_l| \geq k$. Furthermore, let $S \to w_l$ be an additional rule for G, where S is a new symbol in $\Sigma \setminus \Delta$ and $S \to w_l$ a new production in a new table h_{new} in H.*

A word w is in $L(G)$ if and only if there exists a derivation $\psi \overset{*}{\underset{G}{\Longrightarrow}} w$, such that

1. *there is a derivation tree t of w_l from ψ, and*
2. *there exists a frame $R = (S, 1, n, h, c)$, for some h and c, such that $p(R) = 0$.*

If $|\psi| \geq k$, then let $S \to \psi$ be an additional rule in a new table in H of G.

Definition 3.8 (child collection [7]) *Let $R = (A, l, r, h, c)$ be a frame of a word w, and let $\mathcal{R} = \{R_1, \ldots, R_j\}$ be a frame collection of w, where $R_i = (A_i, l_i, r_i, h_i, c_i)$ for all $1 < i \leq j$. We* *say that $\mathcal{R}$ is a child collection of R if: $A \to A_1 \cdots A_j \in H$, $l = l_1$, $r_j = r$, and $l_i = r_{i-1} + 1$ for all $2 \leq i \leq j$, $h = 1 + \max\{h_1, \dots, h_j\}$, and $c = 1 + \sum_{i=1}^{j} c_i$.*

A child collection *of a frame collection $\mathcal{R}$ is obtained by choosing a child collection for each of the frames in $\mathcal{R}$ and taking their union.*

By Lemma 3.4, we only consider those frames $R = (A, l, r, h, c)$ with bounded length. Since there are at most $\#\Sigma$ choices for A, and n choices for each l and r and since $f(n)$ in Lemma 3.4 is a linear function, the following bound is obtained.

Corollary 3.9 ([7]) *There are $\mathcal{O}(n^5)$ frames to be computed while testing membership for a word of length n.*

The following corollary is a restatement of Lemma 3.3 for frames.

Corollary 3.10 ([7]) *Let $\mathcal{R}$ be a frame collection. Then*

$$p(\mathcal{R}) = \begin{cases} p(\mathrm{High}_k(\mathcal{R})) - |\mathrm{Low}_k(\mathcal{R})| \text{ if } p(\mathrm{High}_k(\mathcal{R})) \geq |\mathrm{Low}_k(\mathcal{R})|, \\ -|\mathcal{R}| \mod k, \text{ otherwise.} \end{cases}$$

By definition, a k-high collection consists of at most $k-1$ frames and by Corollary 3.9 at most $\mathcal{O}(n^{5(k-1)})$ idle periods of k-high collections have to be computed.

Lemma 3.11 ([7]) *Let j be the number of frames in a frame collection $\mathcal{I}$, where $\mathcal{I} = \mathrm{High}_k(\mathcal{I})$. Then*

$$p(\mathcal{I}) = \begin{cases} 0 \text{ if } \mathcal{I} \textit{ is empty}, \\ k - j + \min\{\, p(\mathcal{R}') \mid \mathcal{R}' \textit{ is a child collection of } \mathcal{I}\,\}, \text{ otherwise.} \end{cases}$$

The algorithm in [7] first constructs all the frames for the input word w, and then, computes the number of idle periods for each possible high collection. The number of idle periods for each frame of a word w is computable in polynomial time. The number of idle periods of various frame collections is computed, by increasing height, using the recurrences stated in Corollary 3.10 and Lemma 3.11. Finally, it is tested whether there exists a frame that covers all of w, that is, a frame of the form $(S, 1, |w|, h, c)$ and that has $k-1$ idle periods.

The algorithm does not only decide membership but also provides the information necessary to construct a k-derivation for an input word w (see [7]).

Theorem 3.12 ([7]) *Algorithm 1 runs in time polynomial in n, $\mathcal{O}(n^{5(k-1)(\mathrm{Maxr}(G)+1)+1})$, if both k and G are constant.*

Algorithm 1: Adjusted membership algorithm from [7] for the second phase in $k$ulEPT0L derivations

```
Data: A $k$ulEPT0L $G = (\{A_1, \dots, A_m\}, H, S, \Delta, k)$ , where $H$ contains a new table $h_{new}$ with
      $S \to w'$ as its single production, and a word $w = A_{p_1} \cdots A_{p_n} \in \Delta^*$.
Result: accept if $w \in L(G)$, otherwise reject.
begin
    /* test if $w' = w$ and $S$ directly derives $w'$ in one derivation step;   */
    if $S \to w$ then accept /* construct all the frames of $w$, of height $h$;   */
    for $i := 1$ to $n$ do $(A_{p_i}, i, i, -1, 0)$ is a frame for $h := 0$ to $f(n)$ do
        forall the $g \in H$ do
            forall the $A \to B_1 B_2 \cdots B_j \in g$ do
                forall the $1 \leq l_0 \leq \ldots \leq l_j \leq n$ do
                    forall the $h_1, \dots, h_j$ with $\max\{h_1, \dots, h_j\} = h - 1$ do
                        forall the $0 \leq c_1, \dots, c_j \leq nf(n)$ do
                            if $(B_i, l_{i-1}, l_i, h_i, c_i)$ is a frame or $1 \leq i \leq j$ then
                                $(A, l_0, l_j, h, c_1 + c_2 + \ldots + c_j + 1)$ is a frame;

    /* compute the number of idle periods for all collections up to $k - 1$
       frames;                                                                 */
    $p(\{\}) := 0$;
    for $h := -1$ to $f(n)$ do
        forall the frame collections $\mathcal{R}$ of height $h$, consisting of up to $k - 1$ frames, each of
        positive height do
            $q := \infty$;
            forall the child collections $\mathcal{R}'$ of $\mathcal{R}$ do
                forall the $\mathrm{High}_k(\mathcal{R}')$ where all $A$ in the frames occur on the left-hand side of a
                production in a table $g \in H$ do
                    $\mathcal{I} := \mathrm{High}_k(\mathcal{R}')$;
                    /* Since the height of $\mathcal{I}'$ is $h - 1$, we can recur on $p(\mathcal{I}')$;   */
                    if $p(\mathcal{I}') \geq |\mathrm{Low}_k(\mathcal{R}')|$ then
                        $p(\mathcal{R}') := p(\mathcal{I}') - |\mathrm{Low}_k(\mathcal{R}')|$
                    else $p(\mathcal{R}') := -|\mathcal{R}'| \mod k$ $q := \min\{q, p(\mathcal{R}')\}$;
                $p(\mathcal{R}) := k - \#\mathcal{R} + q$;

    /* This is the membership test;                                            */
    for $h := 1$ to $f(n)$ do
        for $c := 2$ to $n \cdot f(n)$ do
            if $(S, 1, n, h, c)$ is a frame and $p((S, 1, n, h, c)) = k - 1$ then
                accept
            else reject
```

References

[1] S. BENSCH, *Parallel systems as mildly context-sensitive grammar formalisms*. Ph.D. thesis, Universität Potsdam, Potsdam, Germany, 2009.

[2] H. BORDIHN, Mildly context-sensitive grammars. In: C. MARTÍN-VIDE, V. MITRANA, G. PĂUN (eds.), *Formal Languages and Application, Studies in Fuzziness and Soft Computing 148*. Springer, Berlin, 2004, 163–173.

[3] J. BRUNO, Deterministic and stochastic scheduling problems with tree-like precedence constraints. *NATO Conference* (1981).

[4] J. DASSOW, G. PĂUN, *Regulated Rewriting in Formal Language Theory*. Springer, 1989.

[5] D. DOLEV, M. WARMUTH, Scheduling flat graphs. *SIAM Journal on Computing* 14 (1985) 3, 638–657.

[6] J. EARLEY, An efficient context-free parsing algorithm. *Communications of the ACM* 13 (1970) 2, 94–102.

[7] J. GONCZAROWSKI, M. K. WARMUTH, Applications of scheduling theory to formal language theory. *Theoretical Computer Science* 37 (1985), 217–243.

[8] A. K. JOSHI, Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions? In: D. R. DOWTY, L. KARTTUNEN, A. M. ZWICKY (eds.), *Natural Language Parsing. Psychological, Computational, and Theoretical Perspectives*. Cambridge University Press, New York, 1985, 206–250.

[9] A. K. JOSHI, K. VIJAY-SHANKER, D. J. WEIR, The convergence of mildly context-sensitive grammar formalisms. In: T. WASOW, P. SELLS (eds.), *The Processing of Linguistic Structure*. MIT Press, 1991, 31–81.

[10] J. OPATRNY, K. CULIK II, Time complexity of recognition and parsing of E0L languages. In: A. LINDENMAYER, G. ROZENBERG (eds.), *Automata, Languages, Development*. North-Holland, Amsterdam, 1976, 243–250.

[11] G. ROZENBERG, A. SALOMAA, *The Mathematical Theory of L-Systems*. Academic Press, New York, 1980.

[12] J. VAN LEEUWEN, The membership question for ET0L languages is polynomially complete. *Information Processing Letters* 3 (1967), 138–143.

[13] K. VIJAY-SHANKER, D. J. WEIR, The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* 27 (1994) 6, 511–545.

[14] D. WÄTJEN, E. UNRUH, On extended k-uniformly limited T0L systems and languages. *Information Processing and Cybernetics EIK 26* 5/6 (1990), 283–299.

[15] D. H. YOUNGER, Recognition and parsing of context-free languages in time n^3. *Information and Control* 10 (1967), 189–208.

ORDERED RESTARTING AUTOMATA: RECENT RESULTS AND OPEN PROBLEMS

Kent Kwee Friedrich Otto

Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
{kwee,otto}@theory.informatik.uni-kassel.de

Abstract
We summarize recent results on ordered restarting automata and present a number of open problems that concern the expressive power of the various types of ordered restarting automata, the closure properties of the defined classes of languages, decision problems, and the descriptional complexity of these types of automata.

1. Introduction

The *restarting automaton* was introduced in [2] as a formal model for the linguistic technique of *analysis by reduction.* Such an automaton has a finite-state control and a flexible tape with endmarkers, on which a read/write window of a fixed finite size operates. A restarting automaton works in cycles. Starting in its initial state with the window at the left end of the tape, it scans the tape from left to right until, at some point, it performs a combined rewrite/restart operation. Such an operation replaces the content of the window by a shorter word, and thereafter it moves the window to the left end of the tape and resets the automaton to its initial state. Since the mid 1990's, many variants of restarting automata have been considered (see, e.g., [13]), for example, the combined rewrite/restart operation has been split into two separate operations [3], or instead of requiring that each rewrite operation is strictly length-reducing, also types of restarting automata have been introduced for which the rewrite operations are just weight-reducing with respect to some predefined weight function [4].

The *ordered restarting automaton* (ORWW-automaton), which was introduced by Mráz and Otto in [11], can be seen as a very special type of the *shrinking* restarting automaton of [4]. It has a window of size three, and during a combined rewrite/restart step, it just replaces the symbol in the middle position of its window by a symbol that is strictly smaller with respect to a predefined partial ordering on the tape alphabet. Of course, instead of the partial ordering, one could use a weight function, but the use of a partial ordering seems to be more intuitive.

In [11], and then later in [10, 12], the deterministic variant of the ORWW-automaton was used as a basic device for several types of two-dimensional restarting automata that accept picture languages. In [14], the study of the descriptional complexity of deterministic ORWW-automata was initiated, which was then continued in [6] and [15]. The nondeterministic ORWW-automaton was studied in detail in [8], further results are to appear in [9]. Finally, by separating the rewrite from the restart operation, we obtain the ORRWW-automaton that is studied in [7].

In the current paper we summarize the main results on ORWW- and ORRWW-automata obtained in these papers and present a number of open problems for future work.

2. Definitions

An *ORWW-automaton* is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, where Q is a finite set of states containing the initial state q_0, Σ is a finite input alphabet, Γ is a finite tape alphabet such that $\Sigma \subseteq \Gamma$, the symbols $\rhd, \lhd \not\in \Gamma$ serve as markers for the left and right border of the work space, respectively,

$$\delta : (Q \times ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\}) \cup \{\rhd\lhd\})) \to 2^{(Q \times \{\mathsf{MVR}\}) \cup \Gamma \cup \{\mathsf{Accept}\}}$$

is the *transition relation*, and $>$ is a *partial ordering* on Γ. The transition relation describes three different types of transition steps:

(1) A *move-right step* has the form $(q', \mathsf{MVR}) \in \delta(q, a_1a_2a_3)$, where $q, q' \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, and $a_2, a_3 \in \Gamma$. It causes M to shift the window one position to the right and to change from state q to state q'. Observe that no move-right step is possible if the window contains the right sentinel $\lhd$.

(2) A *rewrite/restart step* has the form $b \in \delta(q, a_1a_2a_3)$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2, b \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$ such that $a_2 > b$ holds. It causes M to replace the symbol a_2 in the middle of its window by the symbol b and to restart, that is, the window is moved back to the left end of the tape, and M reenters its initial state q_0.

(3) An *accept step* has the form $\mathsf{Accept} \in \delta(q, a_1a_2a_3)$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$. It causes M to halt and accept. In addition, we allow an accept step of the form $\delta(q_0, \rhd\lhd) = \{\mathsf{Accept}\}$.

If $\delta(q, u) = \emptyset$ for some state q and a word u, then M necessarily halts, when it is in state q seeing u in its window, and we say that M *rejects* in this situation. Further, the letters in $\Gamma \smallsetminus \Sigma$ are called *auxiliary symbols*. If $|\delta(q, u)| \leq 1$ for all q and u, then M is a *deterministic ORWW-automaton* (det-ORWW-automaton), and if $Q = \{q_0\}$, then we call M a *stateless ORWW-automaton* (stl-ORWW-automaton) or a *stateless deterministic ORWW-automaton* (stl-det-ORWW-automaton), as in this case the state is actually not needed.

A *configuration* of an ORWW-automaton M is a word $\alpha q \beta$, where $q \in Q$ is the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first three symbols of β. In addition, we admit the configuration $q_0 \rhd \lhd$. A *restarting configuration* has the form $q_0 \rhd w \lhd$; if $w \in \Sigma^*$, then $q_0 \rhd w \lhd$ is also called an *initial configuration*. Further, we use Accept to denote the *accepting configurations*, which are those configurations that M reaches by an accept step.

Any computation of an ORWW-automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the head is moved along the tape by MVR steps until a rewrite/restart step is performed and thus, a new restarting configuration is reached. If no further rewrite operation is performed, any computation necessarily finishes in a halting configuration – such a phase is called a *tail*. By $\vdash_M^c$ we denote the execution of a complete cycle, and $\vdash_M^{c^*}$ is the reflexive transitive closure of this relation. It can be seen as the *rewrite relation* that is realized by M on the set of restarting configurations.

An input $w \in \Sigma^*$ is accepted by M if there is a computation of M which starts with the initial configuration $q_0 \rhd w \lhd$ and ends with an accept step. The language consisting of all (input) words that are accepted by M is denoted by $L(M)$.

The *ORRWW-automaton* is obtained from the ORWW-automaton by splitting the rewrite/restart operation into separate rewrite and restart operations. However, it is still required that an ORRWW-automaton executes exactly one rewrite operation in each cycle.

As each cycle (of an ORWW- or ORRWW-automaton) M contains a rewrite operation, which replaces a symbol a by a symbol b that is strictly smaller than a with respect to the given ordering $>$, each computation of M on an input of length n consists of at most $(|\Gamma|-1)\cdot n$ cycles. Thus, M can be simulated by a nondeterministic single-tape Turing machine in time $O(n^2)$.

For restarting automata in general, each RR-variant is at least as powerful as the corresponding R-variant, but for stateless automata the situation is not that obvious. The feature of continuing to read the tape after a rewrite step has been executed is problematic for these automata, as they cannot distinguish between the phase of a cycle *before* the rewrite step and the phase *after* the rewrite step. In [5] this problem has been addressed for various types of deterministic restarting automata, and two options for dealing with it have been proposed. Here we follow the second option presented in [5] which distinguishes between two *phases* of each cycle: the first phase, which ends with the execution of a rewrite operation, and the second phase, which starts after the execution of a rewrite operation and ends with either a restart or an accept step. These two phases are realized by providing two separate transition functions. In [5] the corresponding stateless restarting automata are called *two-phase* restarting automata, but as we will only deal with this type of stateless (deterministic) ORRWW-automata, we just call them *stateless* (*deterministic*) *ORRWW-automata* (stl-(det-)ORRWW-automata). Formally, these automata are defined as follows.

Definition 2.1 *A* stl-ORRWW-automaton *is described by a 7-tuple* $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_1, \delta_2, >)$, *where* Σ, Γ, $\rhd$, $\lhd$, *and* $>$ *are defined as for ORWW-automata, and*

$$\delta_1 : ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \cup \{\rhd\lhd\} \to 2^{\Gamma \cup \{\mathsf{MVR},\mathsf{Accept}\}}$$

and

$$\delta_2 : (\Gamma^{\leq 2} \cdot (\Gamma \cup \{\lhd\})) \to 2^{\{\mathsf{MVR},\mathsf{Restart},\mathsf{Accept}\}}$$

are the transition relations. *Here it is required that $b > b'$ holds for each rewrite instruction $b' \in \delta_1(abc)$.*

Given a word $w \in \Sigma^+$ as input, at first the transition relation δ_1 is used until either an accept instruction is executed, a rewrite instruction $b' \in \Gamma$ is executed, or the window contains a word for which δ_1 is undefined. In the first case, M accepts, in the second case the letter in the middle of the window is replaced by the letter b', the window is moved one step to the right, and the computation is continued using the transition relation δ_2. Finally, in the third case M simply halts without accepting. The transition relation δ_2, which is used in the second phase of a cycle *after* the execution of a rewrite step, shifts the window to the right until either an accept instruction is executed, and then M accepts, until a restart instruction is executed, which resets the window to the left end of the tape and starts the next cycle, or until a window content is reached for which δ_2 is undefined. In the latter case M halts without accepting. For $w = \lambda$, there either is no applicable operation for the initial configuration $\rhd\lhd$, or $\delta_1(\rhd\lhd) = \{\mathsf{Accept}\}$.

Finally, for each type X of automaton, we denote by $\mathcal{L}(\mathsf{X})$ the class of languages that are accepted by automata of type X.

3. Language Classes and Their Inclusion Relations

Concerning the expressive power of the various types of ordered restarting automata, the following results have been obtained.

Theorem 3.1

(a) $\mathcal{L}(\mathsf{stl\text{-}det\text{-}ORWW}) = \mathcal{L}(\mathsf{stl\text{-}ORWW}) = \mathcal{L}(\mathsf{det\text{-}ORWW}) = \mathsf{REG}$ ([8, 11, 14]).

(b) $\mathcal{L}(\mathsf{stl\text{-}det\text{-}ORRWW}) = \mathcal{L}(\mathsf{stl\text{-}ORRWW}) = \mathcal{L}(\mathsf{det\text{-}ORRWW}) = \mathsf{REG}$ ([7]).

The result in (a) also holds for stl-det-ORWW-automata that are *reversible* [15], where a stl-det-ORWW-automaton M is called reversible if it has a *reverse transition function* that undoes the cycles of the computations of M. Concerning the expressive power of nondeterministic ORWW-automata, the following results have been obtained.

Theorem 3.2 [8] *The language class* $\mathcal{L}(\mathsf{ORWW})$ *is incomparable to the language classes* DLIN, LIN, DCFL, CFL, CRL, *and* GCSL *with respect to inclusion.*

Here $L'_{\text{copy}} = \{\, w\$u \mid w, u \in \{a,b\}^*, |w|, |u| \geq 2, u \text{ is a scattered subsequence of } w \,\}$ is a language that is accepted by an ORWW-automaton, but that it is not even growing context-sensitive. On the other hand, the deterministic linear language $L_1 = \{\, a^n b^n \mid n \geq 1 \,\}$ is not accepted by any ORWW-automaton due to the following Cut-and-Paste Lemma.

Theorem 3.3 (Cut-and-Paste Lemma) [8]
For each ORWW-automaton M, there exists a constant $N_c(M) > 0$ such that each word $w \in L(M)$, $|w| \geq N_c(M)$, has a factorization $w = xyz$ satisfying all of the following conditions:

$$\text{(a) } |yz| \leq N_c(M), \ \text{(b) } |y| > 0, \ \textit{and } \text{(c) } xz \in L(M).$$

Actually, also a Pumping Lemma has been obtained for ORWW-automata that nicely complements the above lemma.

Theorem 3.4 (Pumping Lemma) [9]
For each ORWW automaton M, there exists a constant $N_p(M) > 0$ such that each word $w \in L(M)$, $|w| \geq N_p(M)$, has a factorization $w = xyz$ satisfying all of the following conditions:

$$\text{(a) } |xy| \leq N_p(M),\ \text{(b) } |y| > 0,\ \textit{and } \text{(c) } xy^m z \in L(M) \textit{ for all } m \geq 1.$$

By Theorem 3.3, a sufficiently long word from $L(M)$ can be cut within a suffix, and by Theorem 3.4, it can be pumped within a prefix. These results have the following consequence.

Theorem 3.5 [9] *If a unary language is accepted by an ORWW-automaton, then it is necessarily a regular language.*

Finally, concerning ORRWW-automata, the following result has been obtained.

Theorem 3.6 [7] *Each context-free language is accepted by some ORRWW-automaton.*

Also it has been observed that ORRWW-automata accept some unary languages that are not semi-linear, that is, not regular. In summary, we have the hierarchy of language classes depicted in Figure 1.

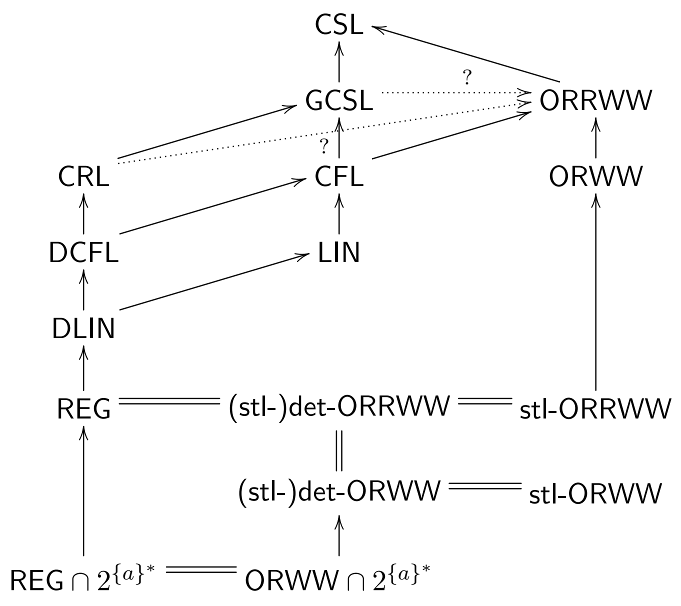


Figure 1: The language classes accepted by the various types of ordered restarting automata in relation to the extended Chomsky hierarchy. The dotted arrows indicate inclusions that are still open.

The following inclusion problems, however, are currently still open.

Open Problem 1 (a) *Is* GCSL *contained in* $\mathcal{L}(\mathsf{ORRWW})$*?*

(b) *Is at least* CRL *contained in* $\mathcal{L}(\mathsf{ORRWW})$*?*

(c) *Is* $\mathcal{L}(\mathsf{ORRWW})$ *a proper subclass of* CSL*? Observe that the equality* $\mathsf{CSL} = \mathcal{L}(\mathsf{ORRWW})$ *would imply that* $\mathsf{NP} = \mathsf{PSPACE}$*, as* CSL *contains* PSPACE*-complete languages, while* $\mathcal{L}(\mathsf{ORRWW})$ *is a subclass of* NP*.*

(d) *Does* $\mathcal{L}(\mathsf{ORWW})$ *only contain languages that have a semi-linear Parikh image? In the unary case this holds by Theorem 3.5, but how about the general case?*

4. Closure Properties of Nondeterministic Classes

Concerning the closure properties for the nondeterministic language classes $\mathcal{L}(\mathsf{ORWW})$ and $\mathcal{L}(\mathsf{ORRWW})$, the following results are known.

Theorem 4.1 [8] *The language class* $\mathcal{L}(\mathsf{ORWW})$ *is closed under union, intersection, product, Kleene star, inverse morphisms, and non-erasing morphisms, but it is not closed under complementation nor under reversal.*

Theorem 4.2 [7] *The language class* $\mathcal{L}(\mathsf{ORRWW})$ *is closed under union, intersection, product, Kleene star, inverse morphisms, non-erasing morphisms, and reversal.*

Here the following questions are still open.

Open Problem 2 (a) *Is* $\mathcal{L}(\mathsf{ORWW})$ *closed under arbitrary morphisms?*

(b) *Is* $\mathcal{L}(\mathsf{ORRWW})$ *closed under complementation? Observe that this class is not closed under arbitrary morphisms, as it is closed under intersection and contains the context-free languages.*

5. Decision Problems for Nondeterministic Classes

Based on the Cut-and-Paste Lemma, the following decidability result has been derived in [8].

Theorem 5.1 *For ORWW-automata, the emptiness problem is decidable.*

In combination with the Pumping Lemma, the following is shown in [9].

Theorem 5.2 *For ORWW-automata, the finiteness problem is decidable.*

On the other hand, as $\mathcal{L}(\mathsf{ORRWW})$ is closed under intersection, and as it contains all context-free languages, the following undecidability results hold for ORRWW-automata.

Theorem 5.3 [7] *For ORRWW-automata, emptiness, finiteness, universality, regularity, inclusion, and equivalence are all undecidable.*

Here the following questions are still open.

Open Problem 3 (a) *Is regularity decidable for ORWW-automata? In fact, given an ORWW-automaton M and a regular language L, is it decidable whether $L \subseteq L(M)$ holds? Observe that the converse inclusion is decidable, as $\mathcal{L}(\mathsf{ORWW})$ is closed under intersection and as emptiness is decidable for ORWW-automata.*

(b) *Are inclusion or equivalence decidable for ORWW-automata?*

(c) *The proof of the decidablility of emptiness for ORWW-automata is based on Higman's Theorem [1]. Accordingly, the method for solving it given in [8] is quite inefficient. Is there a more efficient algorithm for solving this problem, or can a non-trivial lower bound be established? The same questions can be asked for the finiteness problem.*

6. Descriptional Complexity for Stateless Ordered Restarting Automata

Finally, as all stateless and/or deterministic types of ordered restarting automata characterize the regular languages, one can ask about the descriptional complexity of these types of automata, where for stateless types of automata, the cardinality of the tape alphabet is taken as a complexity measure. For stl-det-ORWW-automata, the following results have been obtained.

Theorem 6.1 [6, 14]

(a) *For each DFA $A = (Q, \Sigma, q_0, F, \varphi)$, there is a stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ such that $L(M) = L(A)$ and $|\Gamma| = |Q| + |\Sigma|$.*

(b) *For each stl-det-ORWW-automaton M with an alphabet of size n, there exists an NFA A of size $2^{O(n)}$ such that $L(A) = L(M)$ holds.*

(c) *For each $n \geq 1$, there exists a regular language $B_n \subseteq \{0, 1, \#, \$\}^*$ such that B_n is accepted by a stl-det-ORWW-automaton over an alphabet of size $O(n)$, but each NFA for accepting B_n has at least 2^n states.*

Further, in [8] a lower bound of $2^{O(n)}$ is given for the conversion of a stl-ORWW-automaton into an equivalent stl-det-ORWW-automaton, and from the results of [7], an upper bound of $2^{O(n)}$ can be derived for the conversion of a stl-det-ORRWW-automaton into a stl-det-ORWW-automaton. Here, however, many problems are still open.

Open Problem 4 (a) *How can a stl-ORWW-automaton efficiently be converted into a stl-det-ORWW-automaton? Is the lower bound mentioned above achievable?*

(b) *Find a lower bound for the conversion of a stl-det-ORRWW-automaton into a stl-det-ORWW-automaton!*

(c) *Find lower and upper bounds for the conversion of a stl-ORRWW-automaton into a stl-det-ORWW-automaton!*

(d) *Determine the descriptional complexity of language operations in terms of stl-det-ORWW-automata! A few preliminary results have been presented in [14].*

References

[1] G. HIGMAN, Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society* 2 (1952), 326–336.

[2] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting automata. In: H. REICHEL (ed.), *Proceedings FCT'95*. *LNCS 965*, Springer, Heidelberg, 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* 4 (1999), 287–311.

[4] T. JURDZIŃSKI, F. OTTO, Shrinking restarting automata. *International Journal of Foundations of Computer Science* 18 (2007), 361–385.

[5] M. KUTRIB, H. MESSERSCHMIDT, F. OTTO, On stateless deterministic restarting automata. *Acta Informatica* 47 (2010), 391–412.

[6] K. KWEE, F. OTTO, On some decision problems for stateless deterministic ordered restarting automata. In: J. SHALLIT, A. OKHOTIN (eds.), *Proceedings DCFS 2015*. *LNCS 9118*, Springer, Heidelberg, 2015, 165–176.

[7] K. KWEE, F. OTTO, On ordered RRWW-automata. In: S. BRLEK, C. REUTENAUER (eds.), *Proceedings DLT 2016*. *LNCS 9840*, Springer, Heidelberg, 2016, 268–279.

[8] K. KWEE, F. OTTO, On the effects of nondeterminism on ordered restarting automata. In: R. FREIVALDS, G. ENGELS, B. CATANIA (eds.), *Proceedings SOFSEM 2016*. *LNCS 9587*, Springer, Heidelberg, 2016, 369–380.

[9] K. KWEE, F. OTTO, *A pumping lemma for ordered restarting automata*, 2016. Submitted.

[10] F. MRÁZ, F. OTTO, Extended two-way ordered restarting automata for picture languages. In: A.-H. DEDIU, C. MARTÍN-VIDE, J. SIERRA-RODRÍGUEZ, B. TRUTHE (eds.), *LATA 2014*. *LNCS 8370*, Springer, Heidelberg, 2014, 541–552.

[11] F. MRÁZ, F. OTTO, Ordered restarting automata for picture languages. In: V. GEFFERT, B. PRENEEL, B. ROVAN, J. ŠTULLER, A. MIN TJOA (eds.), *Proceedings SOFSEM 2014*. *LNCS 8327*, Springer, Heidelberg, 2014, 431–442.

[12] F. MRÁZ, F. OTTO, D. PRŮŠA, On a class of rational functions for pictures. 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, Oesterreichische Computer Gesellschaft, Wien, 2015, 159–176.

[13] 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, Heidelberg, 2006, 269–303.

[14] F. OTTO, On the descriptional complexity of deterministic ordered restarting automata. In: H. JÜRGENSEN, J. KARHUMÄKI, A. OKHOTIN (eds.), *Proceedings DCFS 2014*. *LNCS 8614*, Springer, Heidelberg, 2014, 318–329.

[15] F. OTTO, M. WENDLANDT, K. KWEE, Reversible ordered restarting automata. In: J. KREVINE, J. STEFANI (eds.), *Proceedings RC 2015*. *LNCS 9138*, Springer, Heidelberg, 2015, 60–75.

REGULAR ARRAY GRAMMARS AND BOUSTROPHEDON FINITE AUTOMATA

Henning Fernau$^{(A)}$ Meenakshi Paramasivan$^{(A)}$
D. Gnanaraj Thomas$^{(B)}$

$^{(A)}$Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany,
`{fernau,paramasivan}@uni-trier.de`

$^{(B)}$Department of Mathematics, Madras Christian College, Chennai - 600059, India,
`dgthomasmcc@yahoo.com`

Abstract
We give a formal definition of regular array grammars for describing rectangular pictures. By forbidding one specific direction in the rules of isometric regular array grammars, we obtain four special types of families of picture languages that we then relate to boustrophedon automata.

1. Introduction

In the area of formal picture languages, which is about the transfer of ideas from classical Formal Languages (automata, grammars, expressions) to the description of digital images, two main directions can be identified. We will describe this distinction in the terminology of grammars.
(1) In the isometric approach, local modifications are applied to digital images representing, say, sentential forms, in order to derive, for instance, images only containing terminal letters. In particular, nonterminal symbols will always only represent single terminal letters.
(2) In the non-isometric approach, nonterminals can be replaced by whole two-dimensional pictures. The left-hand sides and right-hand sides of rules have different sizes, which explain the name of this approach. The main problem in this idea is that sentential forms have to be stuck together in a way that the sizes fit together. Therefore, mostly rectangular-shaped pictures can be described in this way. It is this latter feature that we employ in the following.

In this paper, we want to have a look at the simplest forms of generalizing what is known from the string case to the two-dimensional case: finite automata and regular grammars. Even here, many different approaches have been proposed in the literature [4]. We are deliberately picking two of them that seem to be among the simplest ones. Interestingly enough, each originated from one of the mentioned two approaches, although both kind of approaches naturally coincide in the setting of regular string languages. However, as we will see in this paper, the difference between describing rectangular-shaped pictures (alone) and the pictures that allow general shapes is quite crucial for several of our results.

Isometric regular array grammars (IRAG) [1, 6] have been introduced as the lowest level of the Chomsky hierarchy of grammars that describe two-dimensional languages. An isometric array consists of (finitely) many occurrences of symbols from Σ placed in the grid points (pixels) of $\mathbb{Z}^2$ (the discretized plane); the points of the plane which are not marked with elements of Σ are supposed to be marked with the blank symbol $\# \notin \Sigma$. Notice that the introduction of a blank symbol allows the description of pictures that are not of rectangular shape. These pictures are usually formalized (more generally) as mappings $\mathbb{Z}^2 \to \Sigma \cup \{\#\}$ with the understanding that symbols from Σ are assigned to at most finitely many positions (grid points). The collection of all such mappings (in other words, of all pictures), is denoted by Σ^{++} in this paper, with the implicit understanding that $\# \notin \Sigma$ is reserved as a *background symbol.*

Sometimes, isometric arrays are considered identical if they can be transferred into each other by shifting. Here, we take the opposite approach, allowing to move the arrays defined by a grammar to an arbitrary position by arbitrarily selecting the position of the start symbol. By way of contrast, non-isometric varieties of picture-description mechanisms necessarily only describe rectangular-shaped pictures. Therefore, these pictures are often described as $m \times n$ matrices (arrays) containing symbols from Σ as entries. In this paper, we are going to denote the set of all rectangular-shaped arrays over the alphabet Σ by Σ^+_+.

Now, we are going to formally relate the isometric arrays with non-isometric arrays. To this end, we identify some $m \times n$ matrix (a_{ij}) with entries from Σ, i.e., some element from Σ^+_+, with the isometric array A with $A(i,j) = a_{ij}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$ and $A(i,j) = \#$ for all other cases. In this sense, we can think of Σ^{++} as being the set of all isometric rectangular arrays whose left upper non-blank entry is at coordinate $(1,1)$.

Interestingly enough, also some sort of reverse embedding is possible. If $A \in \Sigma^{++}$ is an isometric array, then there is some smallest $m \times n$ rectangle such that outside of this rectangle, only background symbols are attached to grid points via A. This (unique) rectangular-shaped array can be viewed as an element of $(\Sigma \cup \{\#\})^+_+$. In other words, we have an embedding $\mathrm{emb}_1 : \Sigma^+_+ \to \Sigma^{++}$ and another embedding $\mathrm{emb}_2 : \Sigma^{++} \to (\Sigma \cup \{\#\})^+_+$ such that

- If $A \in \Sigma^+_+$, then $A = \mathrm{emb}_2(\mathrm{emb}_1(A))$.
- If $A \in \Sigma^{++}$ has a regular shape (concerning the grid points labeled with symbols from Σ) and if the uppermost leftmost grid point of this rectangular shaped array has the coordinate $(1,1)$, then $A = \mathrm{emb}_1(\mathrm{emb}_2(A))$. Conversely, if $A \in \Sigma^{++}$ does not satisfy the mentioned conditions, then $A \neq \mathrm{emb}_1(\mathrm{emb}_2(A))$.

For simplicity, we will identify Σ^+_+ with $\{A \in \Sigma^{++} \mid \mathrm{emb}_1(A) \in \Sigma^+_+\}$.

2. Definitions and Examples

Definition 2.1 *An* isometric regular array grammar *G, or IRAG for short, is described by a quintuple $G = (N, \Sigma, P, S, \#)$, where N is the nonterminal alphabet, Σ is the terminal alphabet, P is the set of rules, $S \in N$ is the start symbol, and $\#$ is the blank symbol. Moreover, every* *rule from P is of the form*

$$\#A \to Ba \quad , \qquad A\# \to aB \quad , \quad \begin{matrix}\# \\ A\end{matrix} \to \begin{matrix}B \\ a\end{matrix} \,, \quad \begin{matrix}A \\ \#\end{matrix} \to \begin{matrix}a \\ B\end{matrix} \,, \text{ or } \quad A \to a \quad ,$$

where $A, B \in N$ and $a \in \Sigma$. The derivation of a grammar proceeds as follows:

- *At the beginning, the whole discretized plane is filled with blank symbols.*
- *Then, the start symbol $S \in N$ is placed on some grid position, replacing $\#$. To be more precise, this means that an initial configuration is described by some mapping $\iota : \mathbb{Z}^2 \to \{S, \#\}$ such that $|\iota^-(S)| = 1$ (here ι^- denotes the inverse mapping of ι).*
- *Intermediately, we find that all non-blank symbols in the plane are terminal symbols but one, say, $A \in N$. Formally, this means that an intermediate configuration can be described by some mapping $\chi : \mathbb{Z}^2 \to N \cup \Sigma \cup \{\#\}$ satisfying $|\chi^-(N)| = 1$; in the case discussed in the following, we assume that $|\chi^-(A)| = 1$ (here χ^- denotes the inverse mapping of χ).*
  - *If the position left to A is blank, then we can apply a rule of the first listed type; this application replaces the blank symbol to the left of A by B and then A by a. This type of rule is therefore called a* left movement.
    Given χ, the successor configuration χ' hence satisfies:

    $$\chi'(x,y) = \begin{cases} \chi(x,y) & \textit{if } \chi(x,y) \neq A \\ a & \textit{if } \chi(x,y) = A \\ B & \textit{if } \chi(x,y) = \# \wedge \chi(x+1,y) = A \end{cases}$$

    The successor configurations χ' for the other types of rules informally described below can be formalized in a similar way.
  - *If the position right to A is blank, we can similarly apply the second type of rule. Applying such a rule implements a* right movement.
  - *If the position above A is blank, we can likewise apply the third type of rule. This means an* upward movement.
  - *If the position below A is blank, we can alternatively apply the fourth type of rule, which yields a* downward movement.
- *In any case, we can (if possible) apply the last type of rule; in that case A is replaced by a, so that none of the previously described rule applications are possible henceforth. That is, a terminal configuration can be seen as a mapping $\tau : \mathbb{Z}^2 \to \Sigma \cup \{\#\}$ satisfying $|\tau^-(\Sigma)| < \infty$.*

Let $L(G)$ collect all isometric terminal arrays that can be derived by a finite sequence of rule applications in the described way. More formally, we write $\chi \Rightarrow \chi'$ if χ' is the successor configuration of χ. Then $L(G) = \{\tau : \mathbb{Z}^2 \to (\Sigma \cup \{\#\}) \mid \iota \Rightarrow^+ \tau\}$.

As the start position is arbitrary, any isometric terminal array $W : \mathbb{Z}^2 \to \Sigma \cup \{\#\} \in L(G)$ carries along an infinite number of other arrays $W' \in L(G)$ that can be obtained by shifting W.

Definition 2.2 $\mathcal{L}_\Sigma(\mathit{IRAG}) = \{L(G) : G \textit{ is an IRAG with terminal alphabet } \Sigma\}$.
$L_{\mathit{Rect}}(G) = \{L(G) \cap \Sigma_+^+ : G \textit{ is an IRAG with terminal alphabet } \Sigma\}$.
$\mathcal{L}_{\mathit{Rect},\Sigma}(\mathit{IRAG}) = \{L_{\mathit{Rect}}(G) : G \textit{ is an IRAG with terminal alphabet } \Sigma\}$.

Let us now define four special types of IRAG that are defined by forbidding certain directions. Correspondingly, we obtain four types of families of picture languages.

Definition 2.3 *Let G be some IRAG. If G contains no upwards (or downwards, or left, or right) movements, we face an $\overline{U}$-IRAG (or $\overline{D}$-IRAG, $\overline{L}$-IRAG, $\overline{R}$-IRAG, respectively).*

Definition 2.4 *Let $X \in \{U, D, L, R\}$. Then,*
$\mathcal{L}_{\Sigma}(\overline{X}\text{-}IRAG) = \{L(G) : G \text{ is a } \overline{X}\text{-}IRAG \text{ with terminal alphabet } \Sigma\}$.
$\mathcal{L}_{Rect,\Sigma}(\overline{X}\text{-}IRAG) = \{L_{Rect}(G) : G \text{ is an } \overline{X}\text{-}IRAG \text{ with terminal alphabet } \Sigma\}$.

Example 2.5 *Let us consider the* set K of tokens `L` *of all sizes and of all proportions:*

$$K = \left\{ \begin{smallmatrix} (\mathtt{x} & (\bullet)^n)_{m-1} \\ \mathtt{x} & \mathtt{x}^n \end{smallmatrix} \;:\; n \geq 1, m \geq 2 \right\}.$$

The $\overline{U}$-IRAG $G = (N, \Sigma, P, S, \#)$ such that $L_{Rect}(G) = K$ is defined as follows:

- $N = \{S, A, B, E, F\}$,
- $\Sigma = \{\mathtt{x}, \bullet\}$, *and*
- $P = \{S\# \rightarrow \mathtt{x}A, A\# \rightarrow \bullet A, \#B \rightarrow B\bullet, E\# \rightarrow \mathtt{x}E, \#F \rightarrow F\mathtt{x}$,
  $\begin{smallmatrix}A\\ \#\end{smallmatrix} \rightarrow \begin{smallmatrix}\bullet\\ B\end{smallmatrix}, \begin{smallmatrix}A\\ \#\end{smallmatrix} \rightarrow \begin{smallmatrix}\bullet\\ F\end{smallmatrix}, \begin{smallmatrix}B\\ \#\end{smallmatrix} \rightarrow \begin{smallmatrix}\mathtt{x}\\ S\end{smallmatrix}, \begin{smallmatrix}B\\ \#\end{smallmatrix} \rightarrow \begin{smallmatrix}\mathtt{x}\\ E\end{smallmatrix}, E \rightarrow \mathtt{x}, F \rightarrow \mathtt{x}\}$.

By way of contrast, the following grammar uses all four types of movements.

Example 2.6 *Consider the array language L the set of all square pictures of diagonal lines from the upper left corner to the lower right corner where the elements in the diagonal are* 1 *and the other elements are* 0. *The IRAG $G = (N, \Sigma, P, S, \#)$ such that $L_{Rect}(G) = L$ is defined as follows: $N = \{S, A, B, C, H, E, F\}$, $\Sigma = \{0, 1\}$, $P = P_1 \cup P_2 \cup P_3$ where P_1 contains the following rules, sequentially numbered for the ease of presentation:*

$$1: \begin{smallmatrix}\#\\ S\end{smallmatrix} \rightarrow \begin{smallmatrix}A\\ 1\end{smallmatrix},\; 2: \begin{smallmatrix}\#\\ A\end{smallmatrix} \rightarrow \begin{smallmatrix}A\\ 0\end{smallmatrix},\; 3: \#A \rightarrow B\,0,\; 4: \#B \rightarrow B\,0,\; 5: \begin{smallmatrix}B\\ \#\end{smallmatrix} \rightarrow \begin{smallmatrix}1\\ F\end{smallmatrix},\; 6: \begin{smallmatrix}C\\ \#\end{smallmatrix} \rightarrow \begin{smallmatrix}0\\ C\end{smallmatrix},$$

$$7: C\# \rightarrow 0\,H,\; 8: H\# \rightarrow 0\,H,\; 9: \begin{smallmatrix}\#\\ H\end{smallmatrix} \rightarrow \begin{smallmatrix}E\\ 0\end{smallmatrix},\; a: \begin{smallmatrix}\#\\ E\end{smallmatrix} \rightarrow \begin{smallmatrix}A\\ 1\end{smallmatrix},\; b: \begin{smallmatrix}F\\ \#\end{smallmatrix} \rightarrow \begin{smallmatrix}0\\ C\end{smallmatrix},\; c: S \rightarrow 1,$$

$P_2 = \{d : F \rightarrow 0\}$ and $P_3 = \{f : E \rightarrow 1\}$.

Note: The rules from $P_1 \cup P_2$ are used to generate square arrays of even rows and columns and the rules from $P_1 \cup P_3$ are used to generate square arrays of odd rows and columns. The elements in L are the set of all pictures with equal number of rows and columns, i.e., $m = n$. Hereafter we only say n and we have two cases for n.

For the sake of presentation, we consider only $n = 5$ and $n = 6$ to illustrate the two cases.

$$n = 5: \quad \begin{array}{ccccccccc}
1 & \leftarrow_4 & 0 & \leftarrow_4 & 0 & \leftarrow_4 & 0 & \leftarrow_3 & 0 \\
\downarrow_5 & & & & & & & & \uparrow_2 \\
0 & & 1 & \leftarrow_4 & 0 & \leftarrow_3 & 0 & & 0 \\
\downarrow_b & & \downarrow_5 & & & & \uparrow_2 & & \uparrow_2 \\
0 & & 0 & & 1_f & & 0 & & 0 \\
\downarrow_6 & & \downarrow_b & & \uparrow_9 & & \uparrow_a & & \uparrow_2 \\
0 & & 0 & \rightarrow_7 & 0 & & 1 & & 0 \\
\downarrow_6 & & & & & & \uparrow_9 & & \uparrow_1 \\
0 & \rightarrow_7 & 0 & \rightarrow_8 & 0 & \rightarrow_8 & 0 & & 1
\end{array}
\qquad
n = 6: \quad \begin{array}{ccccccccccc}
1 & \leftarrow_4 & 0 & \leftarrow_4 & 0 & \leftarrow_4 & 0 & \leftarrow_4 & 0 & \leftarrow_3 & 0 \\
\downarrow_5 & & & & & & & & & & \uparrow_2 \\
0 & & 1 & \leftarrow_4 & 0 & \leftarrow_4 & 0 & \leftarrow_3 & 0 & & 0 \\
\downarrow_b & & \downarrow_5 & & & & & & \uparrow_2 & & \uparrow_2 \\
0 & & 0 & & 1 & \leftarrow_3 & 0 & & 0 & & 0 \\
\downarrow_6 & & \downarrow_b & & \downarrow_5 & & \uparrow_a & & \uparrow_2 & & \uparrow_2 \\
0 & & 0 & & 0_d & & 1 & & 0 & & 0 \\
\downarrow_6 & & \downarrow_6 & & & & \uparrow_9 & & \uparrow_a & & \uparrow_2 \\
0 & & 0 & \rightarrow_7 & 0 & \rightarrow_8 & 0 & & 1 & & 0 \\
\downarrow_6 & & & & & & & & \uparrow_9 & & \uparrow_1 \\
0 & \rightarrow_7 & 0 & \rightarrow_8 & 0 & \rightarrow_8 & 0 & \rightarrow_8 & 0 & & 1
\end{array}$$

We now give the definition of boustrophedon finite automata, or BFA for short, a new automaton model for picture processing, that was introduced in [2]. A more general variant of this automaton model, called general boustrophedon finite automaton, or GBFA for short, was introduced in [3].

Definition 2.7 *A* boustrophedon finite automaton*, or BFA for short, can be specified as a 7-tuple $M = (Q, \Sigma, R, s, F, \#, \Box)$, where Q is a finite set of states, Σ is an input alphabet, $R \subseteq Q \times (\Sigma \cup \{\#\}) \times Q$ is a finite set of rules. A rule $(q, a, p) \in R$ is usually written as $qa \to p$. The special symbol $\# \notin \Sigma$ indicates the border of the rectangular picture that is processed, $s \in Q$ is the initial state, F is the set of final states.*

We are now going to discuss the notions of configurations, valid configurations and an according configuration transition to formalize the work of BFAs, based on snapshots of their work.

Let $\Box$ be a new symbol indicating an erased *position and let $\Sigma_{\#,\Box} := \Sigma \cup \{\#, \Box\}$. Then $C_M := Q \times (\Sigma_{\#,\Box})^+_+ \times \mathbb{N}$ is the set of configurations of M.*

A configuration $(p, A, \mu) \in C_M$ is valid *if $1 \leq \mu \leq |A|_r$ and, for every i, $1 \leq i \leq \mu - 1$, the ith row equals $\# \Box^{|A|_c - 2} \#$, for every j, $\mu + 1 \leq j \leq |A|_r$, the jth row equals $\#w\#$, $w \in \Sigma^{|A|_c - 2}$, and, for some ν, $0 \leq \nu \leq |A|_c - 2$, $w \in \Sigma^{|A|_c - \nu - 2}$, the μ^{th} row equals $\# \Box^\nu w\#$, if μ is odd and $\#w \Box^\nu \#$, if μ is even. Notice that valid configurations model the idea of observable snapshots of the work of the BFA.*

- *If (p, A, μ) and (q, A', μ) are two valid configurations such that A and A' are identical but for one position (i, j), where $A'[i, j] = \Box$ while $A[i, j] \in \Sigma$, then $(p, A, \mu) \vdash_M (q, A', \mu)$ if $pA[i, j] \to q \in R$.*
- *If (p, A, μ) and $(q, A, \mu + 1)$ are two valid configurations, then $(p, A, \mu) \vdash_M (q, A, \mu + 1)$ if the μ^{th} row contains only $\#$ and $\Box$ symbols, and if $p\# \to q \in R$.*

The reflexive transitive closure of the relation $\vdash_M$ is denoted by $\vdash_M^*$. The BFA M is deterministic, or a BDFA for short, if for all $p \in Q$ and $a \in \Sigma \cup \{\#\}$, there is at most one $q \in Q$ with $pa \to q \in R$. The language $L(M)$ accepted by M is then the set of all $m \times n$ pictures W over Σ such that

$$(s, \#_m \varobar W \varobar \#_m, 1) \vdash_M^* (f, \#_m \varobar \Box_m^n \varobar \#_m, m)$$

for some $f \in F$.

The according array language family is denoted by $\mathcal{L}_\Sigma(\text{BFA})$. A BFA looks formally identical to a string-processing finite automaton, with input alphabet $\Sigma \cup \{\#\}$. However, it is meant to process arrays from Σ^+_+ by scanning the first row of the rectangle left to right and changing the reading directions each time when a boundary symbol $\#$ is read (as the ox turns). We reserve a special symbol $\Box$ to fill the area already read by the BFA to define configurations.

We will also use two unary well-known operators on rectangular arrays, called *quarter-turn* (written Q) and *reflection about the base* [5] (written R_b). As usual, we extend Q and R_b to array languages and language families.

3. Characterization Results

Let us introduce a normal form for BFAs [2] that ensures direction-awareness.

Definition 3.1 *Let $M = (Q, \Sigma, R, s, F, \#, \Box)$ be a BFA. M is called* direction-aware*, or a d-BFA for short, if there is a mapping $d : Q \to \{r, \ell\}$ such that $d(q) = d(p)$ for any rule $qa \to p$ with $a \in \Sigma$ and $d(q) \neq d(p)$ for any rule $q\# \to p$. In addition, $d(s) = r$.*

We now prove the *direction-aware normal form lemma*, in short *DANF lemma*, for BFAs.

Lemma 3.2 $\mathcal{L}_\Sigma(\mathrm{BFA}) = \mathcal{L}_\Sigma(\mathrm{d-BFA})$.

Proof. $\mathcal{L}_\Sigma(\mathrm{d-BFA}) \subseteq \mathcal{L}_\Sigma(\mathrm{BFA})$ is trivial. Conversely, let $M = (Q, \Sigma, R, s, F, \#, \Box)$ be a BFA. Define a d-BFA $M_d = (Q_d, \Sigma, R_d, (s, r), F_d, \#, \Box)$, where $Q_d = Q \times \{r, \ell\}$, with $d : Q_d \to \{r, \ell\}$, $(q, x) \mapsto x$ for $x \in \{r, \ell\}$; $F_d = F \times \{r, \ell\}$ and

$$\begin{aligned} R_d = {} & \{(p, x)a \to (q, x) : pa \to q \in R \text{ and } x \in \{r, \ell\}\} \\ & \cup \{(p, x)\# \to (q, y) : p\# \to q \in R \text{ and } x, y \in \{r, \ell\}, x \neq y\}. \end{aligned}$$

The idea of the construction is that the second component in the state allows us to keep track of the direction, depending upon reading odd- or even-numbered rows. $\Box$

As can be seen, the direction of movement is made explicit in a second component of the state alphabet in BFAs in DANF.

Theorem 3.3 $\mathcal{L}_\Sigma(\mathrm{BFA}) = \mathcal{L}_{Rect,\Sigma}(\overline{U}-\mathrm{IRAG}) = \mathcal{L}_{Rect,\Sigma}(\overline{D}-\mathrm{IRAG})$.

Proof. We first show that $\mathcal{L}_\Sigma(\mathrm{BFA}) \subseteq \mathcal{L}_{Rect,\Sigma}(\overline{U}-\mathrm{IRAG})$. Let $M = (Q, \Sigma, R, s, F, \#, \Box)$ be some BFA. W.l.o.g., according to Lemma 3.2, we can assume that M is a BFA in DANF; so, the second component of the state alphabet gives the direction of the movements. We remove useless states from M so that M has only useful states.

Now define an $\overline{U}-$IRAG, $G = (N, \Sigma, P, S, \#)$, with $L(M) = L_{Rect}(G)$, in three steps. As a first step we extract the LEFT, RIGHT movements and terminal rules. As a second step we extract the DOWNWARD movement rules of the target $\overline{U}-$IRAG from the given BFA M by combining the non-# rule with the # rule of the BFA M. After the second step we eliminate the non-terminating nonterminals of the target $\overline{U}-$IRAG from the given BFA M in DANF in the elimination step. As a third step we collect all the rules in steps 1 and 2 and eliminate the non-terminating nonterminals, and we define the $\overline{U}-$IRAG, G such that $L(M) = L_{Rect}(G)$. Let us write the three steps formally as follows:

First Step:

- $\#(p, \ell) \to (q, \ell)a \in P$ for all $(p, \ell)a \to (q, \ell) \in R,\ a \in \Sigma$ (LEFT movement),
- $(p, r)\# \to a(q, r) \in P$ for all $(p, r)a \to (q, r) \in R,\ a \in \Sigma$ (RIGHT movement),
- $(p, d) \to a \in P$ for all $(p, d)a \to (f, d) \in R,\ a \in \Sigma,\ (f, d) \in F$ (terminal rules).

Second Step:

- $\begin{matrix}(p,r)\\ \#\end{matrix} \to \begin{matrix}a\\ (q',\ell)\end{matrix} \in P$ for all $\{(p,r)a \to (q,r), (q,r)\# \to (q',\ell)\} \subseteq R$;
- $\begin{matrix}(p,\ell)\\ \#\end{matrix} \to \begin{matrix}a\\ (q',r)\end{matrix} \in P$ for all $\{(p,\ell)a \to (q,\ell), (q,\ell)\# \to (q',r)\} \subseteq R$.

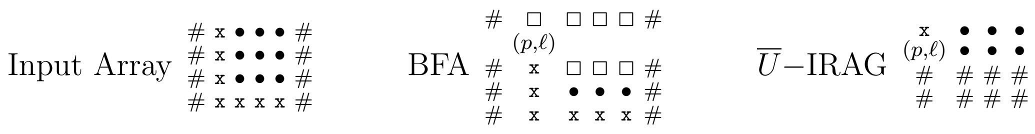
Elimination Step: In this step we eliminate some nonterminals, starting from $N = Q$:

LEFT elimination Delete (p,ℓ) from N if $(p,\ell)a \to (q,\ell) \notin R$ for any $a \in \Sigma$, $(q,\ell) \in Q$.

RIGHT elimination $(p,r)a \to (q,r) \notin R$ for any $a \in \Sigma$, $(q,r) \in Q$.

Third Step: Define the required $\overline{U}-$IRAG $G = (N, \Sigma, P, S, \#)$ where N constructed from Q in the elimination step, $S = (s,r)$, and P collects all rules described in the first and second step.

The idea of the construction is that simulating the $\overline{U}-$IRAG with the given BFA M in DANF that has only useful states. From the second component of the state information of the BFA we use the directions to categorize the rule types left and right for the extraction of the $\overline{U}-$IRAG's production rules. For the rule type downward, the turnings of the BFA are used carefully. So, a step by step computation of the BFA is imitated to be a step by step derivation step of the target grammar $\overline{U}-$IRAG. At any stage of the BFA, the current state of the BFA is matched with the current nonterminal of the simulating $\overline{U}-$IRAG as in the figure below.



The merge of BFA and $\overline{U}-$IRAG in the above picture is the input array, where $\square$ in BFA and $\#$ in $\overline{U}-$IRAG acts like mirror during merging. Hence $\mathcal{L}_\Sigma(\text{BFA}) \subseteq \mathcal{L}_{Rect,\Sigma}(\overline{U}-\text{IRAG})$. Similarly we can prove $\mathcal{L}_\Sigma(\text{BFA}) \supseteq \mathcal{L}_{Rect,\Sigma}(\overline{U}-\text{IRAG})$. Hence $\mathcal{L}_\Sigma(\text{BFA}) = \mathcal{L}_{Rect,\Sigma}(\overline{U}-\text{IRAG})$. According to [3], $\mathcal{L}_\Sigma(\text{BFA}) = R_b(\mathcal{L}_\Sigma(\text{BFA}))$. We can see that $R_b(\mathcal{L}_\Sigma(\text{BFA}))$ works with the RIGHT, UP and LEFT movements starting from the left lower corner of the given picture. With this idea, we can obtain a similar proof for $\mathcal{L}_\Sigma(\text{BFA}) = \mathcal{L}_{Rect,\Sigma}(\overline{D}-\text{IRAG})$. Hence, $\mathcal{L}_{Rect,\Sigma}(\overline{U}-\text{IRAG}) = \mathcal{L}_{Rect,\Sigma}(\overline{D}-\text{IRAG})$. $\square$

As a consequence of Theorem 3.3 and by the definition of quarter-turn Q we obtain:

Corollary 3.4 *For each alphabet Σ,*

$$Q(\mathcal{L}_\Sigma(\text{BFA})) = \mathcal{L}_{Rect,\Sigma}(\overline{R}-\text{IRAG}) = \mathcal{L}_{Rect,\Sigma}(\overline{L}-\text{IRAG}).$$

Theorem 3.5 *For each alphabet Σ with $|\Sigma| > 1$, $\mathcal{L}_\Sigma(\text{BFA}) \cup Q(\mathcal{L}_\Sigma(\text{BFA})) \subsetneq \mathcal{L}_{Rect,\Sigma}(\text{IRAG})$.*

Proof. The inclusion is a consequence of Theorem 3.3 and Corollary 3.4. The strict inclusion can be seen through the array language L given in Example 2.6. We know from Section 6 in [2] that $L \notin \mathcal{L}_{\{0,1\}}(\text{BFA}) \cup Q(\mathcal{L}_{\{0,1\}}(\text{BFA}))$. However, $L \in \mathcal{L}_{Rect,\{0,1\}}(\text{IRAG})$ from Example 2.6. $\square$

$$\begin{tikzcd}
 & \mathcal{L}_{Rect,\Sigma}(\mathrm{IRAG}) \arrow[d, dash] & \\
 & \mathcal{L}_{\Sigma}(\mathrm{BFA}) \cup Q(\mathcal{L}_{\Sigma}(\mathrm{BFA})) \arrow[dl, dash] \arrow[dr, dash] & \\
\mathcal{L}_{\Sigma}(\mathrm{BFA}) = \mathcal{L}_{Rect,\Sigma}(\overline{U}\mathrm{-IRAG}) \arrow[dr, dash] & & Q(\mathcal{L}_{\Sigma}(\mathrm{BFA})) = \mathcal{L}_{Rect,\Sigma}(\overline{R}\mathrm{-IRAG}) \arrow[dl, dash] \\
 & \mathcal{L}_{\Sigma}(\mathrm{BFA}) \cap Q(\mathcal{L}_{\Sigma}(\mathrm{BFA})) &
\end{tikzcd}$$

Figure 1: Relations between language families; see Thm. 3.5 & 3.3 and [2, 3]

Figure 1 shows the inclusion diagram for non-unary array languages, displaying strict inclusions and incomparabilities. For the unary case, we know that all language families coincide except for $\mathcal{L}_{Rect,\{\mathtt{a}\}}(\mathrm{IRAG})$, for which we only **conjecture** equality with $\mathcal{L}_{\{\mathtt{a}\}}(\mathrm{BFA})$.

4. Discussions

We have discussed with the simple example of regular array grammars how classical array language mechanisms introduced to follow the isometric approach to picture languages can be employed to define rectangular-shaped pictures, which is typical for the non-isometric approach.

Conversely, in particular as BFAs process pictures rather in an isometric way, we can use them also to describe non-rectangular arrays. Informally speaking, the processing should start in the uppermost row that contains a symbol from Σ. In this row, the processing starts on the leftmost symbol, moves to the right, until it reaches the background symbol #. Upon sensing it, the automaton moves downwards to the next row and continues processing it by moving to the left, until again a background symbol # is sensed, when the next row is processed, etc.

For comparing these isometric array language classes, it is better to go for looking at equivalence classes of languages under translation, usually denoted by square brackets. So, we arrive at classes like $[\mathcal{L}^{iso}_{\Sigma}(\mathrm{BFA})]$ or $[\mathcal{L}_{\Sigma}(\mathrm{IRAG})]$. The picture of the language families become more intricate here. An illustration can be found in Figure 2.

Theorem 4.1 *For isometric array language classes, the inclusion and incomparability relations stated in Figure 2 holds for any alphabet Σ with $|\Sigma| > 1$.*

Proof. Basically, all arguments can be borrowed from the non-isometric case. Notice, however, that the (unary!) picture $\begin{array}{cc} \mathtt{a} & \\ \mathtt{a} & \mathtt{a} \\ & \mathtt{a} \end{array}$ cannot be described by any BFA-type mechanism, while three directions of movements suffice. $\square$

$$\begin{tikzcd}
 & {[\mathcal{L}_\Sigma(\text{IRAG})]} \arrow[d, dash] & \\
 & {[\mathcal{L}_\Sigma(\overline{U}\text{–IRAG})] \cup [\mathcal{L}_\Sigma(\overline{R}\text{–IRAG})]} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
{[\mathcal{L}_\Sigma(\overline{U}\text{–IRAG})] = [\mathcal{L}_\Sigma(\overline{D}\text{–IRAG})]} \arrow[d, dash] \arrow[dr, dash] & {[\mathcal{L}^{iso}_\Sigma(\text{BFA}) \cup Q(\mathcal{L}^{iso}_\Sigma(\text{BFA}))]} \arrow[dl, dash] \arrow[dr, dash] & {[\mathcal{L}_\Sigma(\overline{R}\text{–IRAG})] = [\mathcal{L}_\Sigma(\overline{L}\text{–IRAG})]} \arrow[dl, dash] \arrow[d, dash] \\
{[\mathcal{L}^{iso}_\Sigma(\text{BFA})]} \arrow[dr, dash] & {[\mathcal{L}_\Sigma(\overline{U}\text{–IRAG})] \cap [\mathcal{L}_\Sigma(\overline{R}\text{–IRAG})]} \arrow[d, dash] & {[Q(\mathcal{L}^{iso}_\Sigma(\text{BFA}))]} \arrow[dl, dash] \\
 & {[\mathcal{L}^{iso}_\Sigma(\text{BFA}) \cap Q(\mathcal{L}^{iso}_\Sigma(\text{BFA}))]} &
\end{tikzcd}$$

Figure 2: Relations between isometric array language families

Acknowledgements

The third author would like to thank the University of Trier for all the support during his visits in May 2015 and 2016, partially using overhead money from a DFG project.

References

[1] C. R. COOK, P. S.-P. WANG, A Chomsky hierarchy of isotonic array grammars and languages. *Computer Graphics and Image Processing* 8 (1978), 144–152.

[2] H. FERNAU, M. PARAMASIVAN, M. L. SCHMID, D. G. THOMAS, Scanning Pictures the Boustrophedon Way. In: R. P. BARNEVA, B. B. BHATTACHARYA, V. E. BRIMKOV (eds.), *International Workshop on Combinatorial Image Analysis IWCIA*. LNCS 9448, Springer, 2015, 202–216.

[3] H. FERNAU, M. PARAMASIVAN, D. G. THOMAS, Picture Scanning Automata. In: R. P. BARNEVA, V. E. BRIMKOV, J. M. R. S. TAVARES (eds.), *International Symposium on Computational Modeling of Objects Presented in Images: Fundamentals, Methods, and Applications (CompIMAGE)*. LNCS, Springer, 2016, To appear.

[4] A. ROSENFELD, R. SIROMONEY, Picture Languages – A Survey. *Languages of Design* 1 (1993), 229–245.

[5] G. SIROMONEY, R. SIROMONEY, K. KRITHIVASAN, Picture Languages with Array Rewriting Rules. *Information and Control (now Information and Computation)* 22 (1973) 5, 447–470.

[6] P. S.-P. WANG, Some new results on isotonic array grammars. *Information Processing Letters* 10 (1980), 129–131.

ON LANGUAGES ACCEPTED BY WEIGHTED RESTARTING AUTOMATA

Qichao Wang

Fachbereich Elektrotechnik/Informatik
Universität Kassel, 34109 Kassel, Germany
wang@theory.informatik.uni-kassel.de

Abstract
Originally, weighted restarting automata have been introduced in order to study quantitative aspects of computations of restarting automata. Here we use them to define classes of formal languages by restricting the weight associated to a given input word through an additional requirement. In this way, a weighted restarting automaton can be used as a language acceptor, which may accept more languages than the underlying (unweighted) restarting automaton. Here we consider the tropical semiring $\mathbb{Z}_\infty$ *and its subsemiring* $\mathbb{N}_\infty$*, and the semirings of formal languages* $\mathsf{REG}(\Delta)$ *and* $\mathsf{CFL}(\Delta)$ *over a finite alphabet* Δ*.*

1. Introduction

Analysis by reduction is a linguistic technique that is used to check the correctness of sentences of natural languages through sequences of local simplifications. Restarting automata have been introduced as a formal model for the analysis by reduction [1]. After their introduction, many different types and variants of restarting automata have been introduced and studied. A recent overview on restarting automata is given in [7]. In order to study quantative aspects of computations of restarting automata, *weighted restarting automata* were introduced in [8].

A weighted restarting automaton $\mathcal{M}$ is given by a pair (M, ω), where M is a restarting automaton on some finite input alphabet Σ, and ω is a weight function that assigns a weight from some semiring S to each transition of M. The product (in S) of the weights of all transitions that are used in a computation yields a weight for that computation, and by forming the sum over the weights of all accepting computations for a given input $w \in \Sigma^*$, a value from S is assigned to w. Thus, a partial function $f : \Sigma^* \to S$ is obtained. By looking at different semirings S and different weight functions ω, various quantitative aspects of the behavior of M can be expressed through these functions. For example, for each input $w \in \Sigma^*$, the value $f(w)$ can be the number of accepting computations on the input w, or the minimal number of auxiliary symbols applied during an accepting computation on the input w. Therefore, by placing a condition T on the value $f(w)$, some words from the language $L(M)$ that is accepted by the underlying (unweighted) restarting automaton M can be filtered out. In this way, we can define a sublanguage $L_T(\mathcal{M})$ of the language $L(M)$ by combining the acceptance condition of M with a condition T on the weight $f(w)$ for $w \in L(M)$.

However, the acceptance conditions relative to different semirings are not equally powerful. For example, the *boolean semiring* $(\{0,1\},+,\cdot,0,1)$ can easily be simulated by other semirings. Here we study the case that the semiring S is the *tropical semiring* $\mathbb{Z}_\infty = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, and its subsemiring $\mathbb{N}_\infty = (\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$, and the case that S is the semiring of context-free languages $\mathsf{CFL}(\Delta) = (\mathsf{CFL}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$, and its subsemiring $\mathsf{REG}(\Delta) = (\mathsf{REG}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$ over a given finite alphabet Δ. In the latter case we restrict our attention to the *word-weighted* restarting automata introduced in [9], that is, the weight of each transition is taken to be a singleton. In [9] these automata are studied as transducers, while here we use them as language acceptors.

2. Definitions and Examples

We assume that the reader is familiar with the standard notions and concepts of theoretical computer science, such as monoids, finite automata, and semirings. Throughout the paper we use $|w|$ to denote the length of a word w, $|w|_a$ to denote the number of appearences of the symbol a in w, and λ to denote the empty word. Further, let REG denote the class of regular languages, let CFL denote the class of context-free languages, and let DCFL denote the class of deterministic context-free languages. Next,let $\mathbb{N}$ denote the set of all non-negative integers, and let $\mathbb{Z}$ denote be the set of all integers. Finally, $\mathbb{P}(X)$ denotes the power set of a set X, and $\mathbb{P}_{\text{fin}}(X)$ denotes the set of all finite subsets of X.

A *restarting automaton* (or RRWW-automaton) is a nondeterministic machine with a finite-state control, a flexible tape with endmarkers, and a read/write window [1, 2]. Formally, it is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \text{¢}, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, the symbols $\text{¢}, \$ \notin \Gamma$ are used as 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

$$\delta : Q \times \mathcal{PC}^{(k)} \to \mathbb{P}((Q \times (\{\mathsf{MVR}\} \cup \mathcal{PC}^{\leq(k-1)})) \cup \{\mathsf{Restart}, \mathsf{Accept}\})$$

is the *transition relation.* Here $\mathcal{PC}^{(k)}$ is the set of *possible contents* of the read/write window of M, where, for $i \geq 0$,

$$\mathcal{PC}^{(i)} := (\text{¢} \cdot \Gamma^{i-1}) \cup \Gamma^i \cup (\Gamma^{\leq i-1} \cdot \$) \cup (\text{¢} \cdot \Gamma^{\leq i-2} \cdot \$),$$

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 relation δ describes four different types of transition steps:

(1) A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, 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* is of the form $(q', v) \in \delta(q, u)$, 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 ¢ 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* is of the form $\mathsf{Restart} \in \delta(q, u)$, 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 ¢, and to reenter the initial state q_0.

(4) An *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to halt and accept.

If $\delta(q, u)$ is undefined for some pair (q, u), then M necessarily halts in a corresponding situation, and we say that M *rejects*. Finally, if each rewrite step is combined with a restart step into a joint rewrite/restart operation, then M is called an *RWW-automaton.* An RRWW-automaton is called an *RRW-automaton* if its tape alphabet Γ coincides with its input alphabet Σ, 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. In general, the automaton M is *nondeterministic*, that is, there can be two or more instructions for the same pair (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. A *non-forgetting* restarting automaton M has extended restart steps, which are combined with a change of state just like the move-right and rewrite operations [5, 6]. The prefix nf- is used to denote types of non-forgetting restarting automata. Further, a restarting automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is called *stateless*, if $Q = \{q_0\}$ (see, e.g., [3, 4]). Thus, M can simply be written as $M = (\Sigma, \Gamma, ¢, \$, q_0, k, \delta)$. We use the prefix stl- to denote stateless types of restarting automata.

A *configuration* of M is a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q is the current state, and $\alpha\beta$ is the current content of the tape, where it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 ¢ w \$$. If $w \in \Sigma^*$, then $q_0 ¢ w \$$ is an *initial configuration.*

Any computation of 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 single rewrite operation until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a *tail.* It is required that in each cycle M performs *exactly one* rewrite step. A word $w \in \Sigma^*$ is accepted by M, if there is an accepting computation which starts from the initial configuration $q_0 ¢w\$$. By $L(M)$ we denote the language consisting of all input words that are accepted by M.

Next we come to the notion of *monotonicity.* Let $C := \alpha q \beta$ be a *rewrite configuration* of an RRWW-automaton M, that is, a configuration in which a rewrite step is to be applied. Then $|\beta|$ is called the *right distance* of C, which is denoted by $D_r(C)$. A *sequence of rewrite configurations* $S = (C_1, C_2, \ldots, C_n)$ is called *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$, 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 an RRWW-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 is not taken into account that corresponds to the possible rewrite step that is executed in the tail of the computation considered. Finally, an RRWW-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.

For studying quantitative aspects of computations of restarting automata, the weighted restarting automaton has been introduced in [8]. A *weighted restarting automaton* of type X, a wX-automaton for short, is a pair (M, ω), where M is a restarting automaton of type X, and ω is a *weight function* from the transitions of M into a semiring S, that is, ω assigns an element $\omega(t) \in S$ as a weight to each transition t of M. The product (in S) of the weights of all transitions that are used in a computation then yields a weight for that computation, and the sum over all weights of all accepting computations of M for a given input word $w \in \Sigma^*$ yields a value from S. In this way, a partial function $f_\omega^M : \Sigma^* \to S$ is obtained. Here we use weighted restarting automata to define sublanguages of the language that is accepted by the underlying (unweighted) restarting automaton.

Definition 2.1 *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a restarting automaton, let ω be a weight function from M into a semiring S, and let $\mathcal{M} = (M, \omega)$. For a subset T of S, $L_T(\mathcal{M}) = \{ w \in L(M) \mid f_\omega^M(w) \in T \}$ is the language* accepted by M relative to T*, that is, a word $w \in \Sigma^*$ belongs to the language $L_T(\mathcal{M})$ iff $w \in L(M)$ and $f_\omega^M(w) \in T$.*

Definition 2.2 *Let* X *be a type of restarting automaton, let S be a semiring, and let $\mathbb{H}$ be a family of subsets of S. Then*

$$\mathcal{L}(\mathsf{X}, S, \mathbb{H}) = \{ L_T(\mathcal{M}) \mid \mathcal{M} \text{ is a weighted restarting automaton of type } \mathsf{X} \text{ and } T \in \mathbb{H} \},$$

is the class of languages that are accepted by weighted restarting automata of type X *relative to* $\mathbb{H}$.

For the semirings of regular languages $\mathsf{REG}(\Delta)$ and context-free languages $\mathsf{CFL}(\Delta)$, the weight of a transition of a restarting automaton M can be any regular or context-free language over Δ. However, some more restricted types of weighted restarting automata were introduced in [9]. Here we only consider the so-called *word-weighted restarting automata* that are defined as follows.

Definition 2.3 *A weighted restarting automaton $\mathcal{M} = (M, \omega)$ is called a* word-weighted restarting automaton *of type* X *(a* $\mathsf{w}_{\mathsf{word}}\mathsf{X}$*-automaton for short), if M is a restarting automaton of type* X *and ω is a weight function from M into a semiring of the form $\mathbb{P}_{\text{fin}}(\Delta^*)$ such that the weight $\omega(t)$ of each transition t of M is a singleton set, that is, it is of the form $\omega(t) = \{v\}$ for some $v \in \Delta^*$.*

We continue with an example that illustrates our definitions.

Example 2.4 *Let $M_1 = (Q, \Sigma, \Gamma, \text{¢}, \$, q_0, k, \delta)$ be the* det-mon-R*-automaton that is defined by taking $Q := \{q_0, q_r\}$, $\Gamma := \Sigma := \{a, b, c\}$, and $k := 3$, where δ is defined as follows:*

$$\begin{array}{llll}
t_1: & (q_0, \text{¢}aa) \to (q_0, \mathsf{MVR}), & t_7: & (q_0, \text{¢}ab) \to (q_0, \mathsf{MVR}),\\
t_2: & (q_0, aaa) \to (q_0, \mathsf{MVR}), & t_8: & (q_0, \text{¢}cc) \to (q_0, \mathsf{MVR}),\\
t_3: & (q_0, aab) \to (q_0, \mathsf{MVR}), & t_9: & (q_0, ccc) \to (q_0, \mathsf{MVR}),\\
t_4: & (q_0, abb) \to (q_r, b), & t_{10}: & (q_0, cc\$) \to \mathsf{Accept},\\
t_5: & (q_0, abc) \to (q_r, c), & t_{11}: & (q_0, \text{¢}c\$) \to \mathsf{Accept},\\
t_6: & (q_0, ab\$) \to (q_r, \$), & t_{12}: & (q_0, \text{¢}\$) \to \mathsf{Accept},\\
t_{13,x}: & (q_r, x) \quad \to \mathsf{Restart} \text{ for all admissible } x. & &
\end{array}$$

It is easily seen that $L(M_1) = \{ a^m b^m c^n \mid m, n \geq 0 \}$. Let $\mathbb{Z}_\infty = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$ be the tropical semiring*, and let ω_1 be the weight function from M_1 into the semiring $\mathbb{Z}_\infty$ that is defined as follows*

$$\omega_1(t_i) = \begin{cases} 1, & \text{if } i = \{4, 5, 6\},\\ -1, & \text{if } i \in \{8, 9, 10, 11\},\\ 0, & \text{otherwise.} \end{cases}$$

Let $\mathcal{M}_1 = (M_1, \omega_1)$, and let $T_1 = \{0\}$. Then $f_{\omega_1}^{M_1}(w) \in T_1$ if and only if $w \in L(M_1)$, and w must be of the form $a^n b^n c^n$ for $n \geq 0$, that is,

$$L_{T_1}(\mathcal{M}_1) = \{ a^n b^n c^n \mid n \geq 0 \}.$$

Deterministic monotone R-automata can only recognize deterministic context-free languages, while the language $L_{T_1}(\mathcal{M}_1)$ is not context-free, and it is not even accepted by any RRW-automaton. Hence, we see that the notion of relative acceptance increases the expressive power of a restarting automaton.

3. Results

It is well known that the class of languages accepted by (deterministic) RRWW- or RWW-automata coincides with the class of (deterministic) context-free languages (see [2]). We have seen that by using an acceptance condition relative to a subset of the semiring $\mathbb{Z}_\infty$, the weighted restarting automaton $\mathcal{M}_1$ given in Example 2.4 can accept a language that is not context-free. Hence, the following proper inclusion results can be derived.

Corollary 3.1 *For all* $\mathsf{X} \in \{\lambda, \mathsf{R}\}$ *and* $\mathbb{H}_0 = \{\{0\}\}$,

$$\begin{array}{llll} (a) & \mathsf{CFL} & \subsetneq & \mathcal{L}(\text{mon-XRWW}, \quad \mathbb{Z}_\infty, \mathbb{H}_0). \\ (b) & \mathsf{DCFL} & \subsetneq & \mathcal{L}(\text{det-mon-XRWW}, \mathbb{Z}_\infty, \mathbb{H}_0). \end{array}$$

Now we compare the tropical semiring $\mathbb{Z}_\infty$ and its subsemiring $\mathbb{N}_\infty = (\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$ to the semirings of context-free languages $\mathsf{CFL} = (\mathsf{CFL}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$ and regular languages $\mathsf{REG} = (\mathsf{REG}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$. Here we consider the following families of subsets of these semirings. Let $\mathbb{H}^z_{fin}$ be the family of finite sets of the semiring $\mathbb{Z}_\infty$, and let $\mathbb{H}^N_{fin}$ be the family of finite sets of the semiring $\mathbb{N}_\infty$. Further, let $\mathbb{H}^{cfl(\Delta)}_{fin}$ be the family of sets of finite languages over a finite alphabet Δ, and for each $T \in \mathbb{H}^{cfl(\Delta)}_{fin}$, $\bigcup\limits_{V \in T} V \in \mathsf{CFL}(\Delta)$. In analogy to $\mathbb{H}^{cfl(\Delta)}_{fin}$, let $\mathbb{H}^{reg(\Delta)}_{fin}$ be the family of sets of finite languages over a finite alphabet Δ, and for each $T \in \mathbb{H}^{reg(\Delta)}_{fin}$, $\bigcup\limits_{V \in T} V \in \mathsf{REG}(\Delta)$.

Theorem 3.2 *For any type* X *of restarting automaton,*

$$\begin{array}{llll} (a) & \mathcal{L}(\mathsf{X}, \mathbb{N}_\infty, \mathbb{H}^N_{fin}) & \subseteq & \mathcal{L}(\mathsf{X}, \mathsf{REG}(\Delta), \mathbb{H}^{reg(\Delta)}_{fin}). \\ (b) & \mathcal{L}(\mathsf{X}, \mathbb{Z}_\infty, \mathbb{H}^z_{fin}) & \subseteq & \mathcal{L}(\mathsf{X}, \mathsf{CFL}(\Delta), \mathbb{H}^{cfl(\Delta)}_{fin}). \end{array}$$

Because of the commutativity of the tropical semirings $\mathbb{N}_\infty$ and $\mathbb{Z}_\infty$, the above inclusions are proper for some weak types of restarting automata.

Theorem 3.3

$$\begin{array}{llll} (a) & \mathcal{L}(\text{stl-det-R(1)}, \mathbb{N}_\infty, \mathbb{H}^N_{fin}) & \subsetneq & \mathcal{L}(\text{stl-det-R(1)}, \mathsf{REG}(\Delta), \mathbb{H}^{reg(\Delta)}_{fin}). \\ (b) & \mathcal{L}(\text{stl-det-R(1)}, \mathbb{Z}_\infty, \mathbb{H}^z_{fin}) & \subsetneq & \mathcal{L}(\text{stl-det-R(1)}, \mathsf{CFL}(\Delta), \mathbb{H}^{cfl(\Delta)}_{fin}). \end{array}$$

Further, for word-weighted restarting automata, we define the following weak acceptance.

Definition 3.4 *Let* $\mathcal{M} = (M, \omega)$ *be a* $\mathsf{w}_{\mathsf{word}}\mathsf{X}$*-automaton with input alphabet* Σ*, where* ω *maps the transitions of* M *to singleton sets over* Δ*.*

(a) *For a set* $T \in \mathsf{REG}(\Delta)$*,* $\hat{L}_T(\mathcal{M}) = \{\, w \in L(M) \mid f^M_\omega(w) \cap T \neq \emptyset \,\}$ *is the language* weakly accepted by $\mathcal{M}$ relative to *the set* T*, that is, a word* $w \in \Sigma^*$ *belongs to the language* $\hat{L}_T(\mathcal{M})$ *iff* $w \in L(M)$ *and* $f^M_\omega(w)$ *contains at least one element of* T*.*

(b) *Let* $\mathbb{H}$ *be a family of subsets of* $\mathsf{REG}(\Delta)$*. Then*

$$\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathbb{H}) = \{\, \hat{L}_T(\mathcal{M}) \mid \mathcal{M} \text{ is a } \mathsf{w}_{\mathsf{word}}\mathsf{X}\text{-automaton and } T \in \mathbb{H}\}$$

is the class of languages that are accepted by $\mathsf{w}_{\mathsf{word}}\mathsf{X}$*-automata relative to* $\mathbb{H}$*.*

In [10] it is shown that weighted restarting automata with the weak acceptance relative to $\mathsf{REG}(\Delta)$ are equivalent to the non-forgetting restarting automata of corresponding type, i.e., $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) = \mathcal{L}(\text{nf-}\mathsf{X})$ for each type X of restarting automaton. Based on this result we give new characterizations of some well-known classes of formal languages.

Theorem 3.5 *For all* $\mathsf{X} \in \{\lambda, \mathsf{R}\}$,

$$\begin{array}{llllll}
(a) & \mathsf{REG} & = & \mathcal{L}(\text{det-mon-R(1)}, \mathbb{N}_\infty, \mathbb{H}^N_{fin}) & = & \hat{\mathcal{L}}(\text{(det-)mon-R(1)}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)). \\
(b) & \mathsf{CFL} & = & & & \hat{\mathcal{L}}(\text{mon-XRWW}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)). \\
(c) & \mathsf{DCFL} & = & \mathcal{L}(\text{det-mon-XRWW}, \mathbb{N}_\infty, \mathbb{H}^N_{fin}) & = & \hat{\mathcal{L}}(\text{det-mon-XRWW}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)).
\end{array}$$

4. Conclusions

We have studied the expressive power of weighted restarting automata with various acceptance conditions relative to a subset of different semirings and compared them. However, many problems remain open. In particular, we do not yet have upper bounds of the expressive power of word-weighted restarting automata of some strong types such as RWW and RRWW. Here we have only considered the case of the tropical semirings $\mathbb{Z}_\infty$ and $\mathbb{N}_\infty$ and that of the semirings of formal languages $\mathsf{CFL}(\Delta)$ and $\mathsf{REG}(\Delta)$ over a finite alphabet Δ. In future the acceptance conditions relative to some other semirings will be studied.

References

[1] P. JANCAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting Automata. In: H. REICHEL (ed.), *FCT'95*. Lecture Notes in Computer Science 965, Springer, Heidelberg, 1995, 283–292.

[2] P. JANCAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On Monotonic Automata with a Restart Operation. *J. Auto. Lang. Comb.* 4 (1999) 4, 287–312.

[3] M. KUTRIB, H. MESSERSCHMIDT, F. OTTO, On stateless deterministic restarting automata. *Acta Inf.* 47 (2010) 7-8, 391–412.

[4] M. KUTRIB, H. MESSERSCHMIDT, F. OTTO, On Stateless Two-Pushdown Automata and Restarting Automata. *Int. J. Found. Comput. Sci.* 21 (2010) 5, 781–798.

[5] H. MESSERSCHMIDT, F. OTTO, On Nonforgetting Restarting Automata That Are Deterministic and/or Monotone. In: D. GRIGORIEV, J. HARRISON, E. A. HIRSCH (eds.), *Computer Science - Theory and Applications, First International Computer Science Symposium in Russia, CSR 2006, Proceedings*. Lecture Notes in Computer Science 3967, Springer, Heidelberg, 2006, 247–258.

[6] H. MESSERSCHMIDT, H. STAMER, Restart-Automaten mit mehreren Restart-Zuständen. In: H. BORDIHN (ed.), *Workshop "Formale Methoden in der Linguistik" und 14. Theorietag "Automaten und Formale Sprachen"*. Institut für Informatik, Universität Potsdam, Potsdam, 2004, 111–116.

[7] F. OTTO, Restarting Automata. In: Z. ÉSIK, C. MARTÍN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. 25, Springer, Heidelberg, 2006, 269–303.

[8] F. OTTO, Q. WANG, Weighted Restarting Automata. *Soft Computing* (2016). DOI: 10.1007/s00500-016-2164-4. The results of this paper have been announced at WATA 2014 in Leipzig.

[9] Q. WANG, N. HUNDESHAGEN, F. OTTO, Weighted Restarting Automata and Pushdown Relations. In: A. MALETTI (ed.), *Algebraic Informatics - 6th International Conference, CAI 2015, Proceedings*. Lecture Notes in Computer Science 9270, Springer, Switzerland, 2015, 196–207.

[10] Q. WANG, F. OTTO, Weighted Restarting Automata as Language Acceptors. In: Y.-S. HAN, K.SALOMAA (eds.), *Implementation and Application of Automata - 21st International Conference, CIAA 2016, Proceedings*. Lecture Notes in Computer Science 9705, Springer, Switzerland, 2016, 298–309.

Author Index

B
Bensch, Suna, 35

F
Fazekas, Szilárd Zsolt, 7
Fernau, Henning, 55

H
Hospodár, Michal, 15

K
Kutrib, Martin, 35
Kwee, Kent, 47

M
Madejski, Grzegorz, 25
Malcher, Andreas, 35

O
Otto, Friedrich, 47

P
Paramasivan, Meenakshi, 55

T
Thomas, D. Gnanaraj, 55

W
Wang, Qichao, 65

Y
Yamamura, Akihiro, 7