RAIRO-Inf. Theor. Appl. **41** (2007) 157–175
DOI: 10.1051/ita:2007013

SEQUENTIAL MONOTONICITY FOR RESTARTING AUTOMATA *

Tomasz Jurdziński[1] and Friedrich Otto[2]

Abstract. As already 2-monotone R-automata accept NP-complete languages, we introduce a restricted variant of j-monotonicity for restarting automata, called *sequential j-monotonicity*. For restarting automata without auxiliary symbols, this restricted variant still yields infinite hierarchies. However, for restarting automata with auxiliary symbols, all degrees of sequential monotonicity collapse to the first level, implying that RLWW-automata that are sequentially monotone of degree j for any $j \geq 1$ only accept context-free languages.

Mathematics Subject Classification. 68Q45.

1. Introduction

Analysis by reduction is a technique used in linguistics to analyse sentences of natural languages. It consists of a stepwise simplification of a given sentence so that the (in)correctness of the sentence is not affected. Restarting automata were introduced by Jancar *et al.* as a theoretical model for the analysis by reduction [5]. These automata can do a bottom-up syntactic analysis for natural languages as well as for formal languages. The notions developed during the study of restarting automata give a rich taxonomy of constraints for various models of analysers [12]. Already several programs used in Czech and German (corpus) linguistics are based on the idea of restarting automata [11, 16].

Keywords and phrases. Restarting automaton, sequential monotonicity, hierarchies.

* *This work was supported by a grant from the Deutsche Forschungsgemeinschaft. It was mainly performed while Tomasz Jurdziński was visiting the University of Kassel. The results have been announced at ITAT 2004, which was organized by the Department of Computer Science of Pavol Jozef Šafárik University, Košice, at Popradské Pleso, September 2004.*

[1] Institute of Computer Science, University of Wrocław, 51-151 Wrocław, Poland; tju@ii.uni.wroc.pl

[2] Fachbereich Elektrotechnik/Informatik, Universität Kassel, 34109 Kassel, Germany; otto@theory.informatik.uni-kassel.de

A (two-way) restarting automaton, RLWW-automaton for short, is a device M that consists of a finite-state control, a flexible tape containing a word delimited by the sentinels ¢ and \$, and a read/write window of a fixed size. This window is moved along the tape by performing move-right and move-left instructions until the control decides (nondeterministically) that the content of the window should be rewritten by some shorter string, in this way shortening the tape. In general, through a rewrite operation auxiliary (that is, non-input) symbols may be introduced into the tape content. After a rewrite, M can continue to move its window until it either halts and accepts, or halts and rejects, or restarts, that is, it places its window over the left end of the tape, reenters the initial state, and continues with the computation. Thus, each computation of M can be described through a sequence of cycles.

In addition to this general model, various restricted versions of the restarting automaton have been considered. First of all there is the RRWW-automaton, which is the one-way variant of the RLWW-automaton, as it does not use the move-left instruction. Then there is the RWW-automaton, which is an RRWW-automaton that is required to perform a restart immediately after executing a rewrite operation. Thus, in each cycle an RWW-automaton only sees the part of the tape between the left sentinel ¢ and the position where the rewrite step is performed. Then there is the RLW-automaton, which only uses the letters of the input alphabet in its rewrite operations. A further restriction leads to the RL-automaton, where each rewrite operation is actually just a delete operation, that is, during each rewrite operation some letters from the content of the read/write window are simply deleted. Obviously the restrictions on the restart operation and the restrictions on the rewrite operation can be combined, which yields the R(R)W-automaton and the R(R)-automaton.

Also a *monotonicity* property was introduced for the various types of restarting automata which is based on the idea that from one cycle to the next in a computation, the actual place where a rewrite operation is performed must not increase its distance from the *right* end of the tape. Monotone restarting automata essentially model bottom-up one-pass parsers. It was shown that monotone RWW-, RRWW-, and RLWW-automata characterize the class CFL of context-free languages, and that the monotone version of the deterministic R(R)(W)(W)-automaton characterizes the class DCFL of deterministic context-free languages [6]. Thus, monotone restarting automata do not have sufficient expressive power to capture all aspects of the analysis by reduction of natural languages. On the other hand, general RLWW-automata even accept some NP-complete languages [7, 10], which means that they cannot be implemented efficiently.

Therefore, the notion of *j-monotonicity* ($j \geq 1$) was introduced in [14, 15] as a generalization of the notion of monotonicity. This notion models the generalization from bottom-up one-pass parsers to bottom-up multi-pass parsers, and it allows us to measure the level of non-monotonicity of a language. Further, this new notion seems to be much better suited to the real task of modelling analysis by reduction. A restarting automaton is called *j-monotone* for an integer $j \geq 1$ if, for each of its computations, the corresponding sequence of cycles can be partitioned into at

most j (in general interleaving) subsequences that are each monotone (see Sect. 2 for exact definitions). It is shown in [15] that the expressive power of j-monotone RRW-automata increases with the value of the parameter j.

Unfortunately, it turned out that already 2-monotone R-automata accept NP-complete languages [8]. This fact means in particular that one should not expect that there exist efficient general algorithms for recognizing languages defined by j-monotone restarting automata. Therefore a different generalization of the notion of monotonicity is needed to capture the phenomena of natural languages.

Here we introduce an alternative generalization of monotonicity, called *sequential j-monotonicity*, which is a restricted variant of the notion of j-monotonicity. It differs from the 'classical' notion of j-monotonicity in that the $(i+1)$-st monotone subsequence only starts after the last cycle of the i-th monotone subsequence. In other words, it is not allowed that the j monotone subsequences interleave. Sequential j-monotonicity is a much more natural generalization of monotonicity than j-monotonicity, and as such it is an interesting notion.

After giving the necessary definitions in Section 2, we study the expressive power of sequentially j-monotone automata without auxiliary symbols. We show in Section 3 that there exist strict infinite hierarchies with respect to the level of sequential monotonicity for all variants of nondeterministic restarting automata without auxiliary symbols. Then we investigate restarting automata with auxiliary symbols in Section 4. We show that all degrees of sequential monotonicity collapse to the first level in this case, implying that such automata accept only context-free languages. On the one hand this fact ensures the existence of polynomial time recognition algorithms for all languages defined by sequentially j-monotone automata. On the other hand this result is somewhat 'frustrating', as it shows that the expressive power of sequentially j-monotone restarting automata is severely limited. However, this result can be seen as another example of the 'expressibility' of the family of languages defined by context-free grammars, as sequentially j-monotone restarting automata are in some respects a much less restricted machine model than pushdown automata. In fact, sequential j-monotonicity may be expressed intuitively as a possibility to 'reuse' the pushdown store j times.

In Section 5 we consider sequential j-monotonicity for deterministic restarting automata. For automata that are not allowed to perform move-left transitions, all levels of sequential j-monotonicity collapse to the first level, which is an immediate consequence of the corresponding result for the 'classical' notion of j-monotonicity. On the other hand, we obtain an infinite hierarchy for deterministic restarting automata without auxiliary symbols that are allowed to perform move-left transitions.

2. Definitions and notation

We start by restating in short the definition of the various models of the restarting automaton that will be considered in this paper. For more details concerning the notions introduced we refer to [12, 13].

A *two-way restarting automaton*, RLWW-automaton for short, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, and Γ is a finite tape alphabet containing Σ, where the letters in $\Gamma \setminus \Sigma$ are called *auxiliary symbols*. Further, the symbols $¢, \$ \notin \Gamma$ serve as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and δ is the *transition relation* that assigns a finite set of transitions to each pair (q, u) consisting of a state $q \in Q$ and a possible content u of the read/write window. There are five different types of transition steps:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q' \in Q$ and $u \neq \$$. 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 just the symbol \$, then no shift to the right is possible.
2. A *move-left step* is of the form $(q', \mathsf{MVL}) \in \delta(q, u)$, where $q' \in Q$ and u does not start with the symbol $¢$. It causes M to shift the read/write window one position to the left and to enter state q'.
3. A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q' \in Q$, $u \neq \$$, and v is a string such that $|v| < |u|$. It causes M to replace the content u of the read/write window by the string v, thereby shortening the tape, 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 can new occurrences of these markers be 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.
4. A *restart step* is of the form $\mathsf{Restart} \in \delta(q, u)$. It causes M to place its read/write window over the left end of the tape, so that the first symbol it sees is the left border marker $¢$, and to reenter the initial state q_0.
5. An *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$. It causes M to halt and accept.

If $\delta(q, u) = \emptyset$ for a pair (q, u), then M necessarily halts, and we say that M *rejects* in this situation. In addition, the transition relation must satisfy the requirement that, ignoring move-right and move-left steps for the moment, rewrite operations and restart steps alternate in every computation, with a rewrite operation coming first.

A *configuration* of M is a string $\alpha q \beta$, where $q \in Q \cup \{\mathsf{Accept}\}$, and either $\alpha = \varepsilon$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state (or the fact that M has accepted), $\alpha\beta$ is the current content of the tape, and it is understood that the read/write window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 ¢ w \$$, where

$w \in \Gamma^*$; if $w \in \Sigma^*$, then $q_0 ¢ w \$$ is an *initial configuration*. A *halting configuration* is a configuration $\alpha q \beta$ in which M cannot perform any transition; it is called *accepting*, if $q = \mathsf{Accept}$, otherwise it is called *rejecting*.

In general, the automaton M is *nondeterministic*, that is, there can be two or more instructions with the same left-hand side (q, u), and thus, there can be more than one computation for an input word. If that is not the case, the automaton is *deterministic*. We will use the prefix det- to denote classes of deterministic restarting automata.

We observe that any finite computation of a two-way restarting automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the head moves along the tape performing MVR operations, MVL 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, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. As each cycle contains an application of a Rewrite operation, each new phase starts on a shorter word than the previous one. We use the notation $u \vdash_M^c v$ to denote a cycle of M that begins with the restarting configuration $q_0 ¢ u \$$ and ends with the restarting configuration $q_0 ¢ v \$$; the relation $\vdash_M^{c^*}$ is the reflexive and transitive closure of $\vdash_M^c$. Thus, $\vdash_M^c$ can be seen as the *single-step rewrite relation* induced by M, and $\vdash_M^{c^*}$ is the corresponding *rewrite relation*.

An input *word* $w \in \Sigma^*$ *is accepted by* M, if there is a computation which, starting with the initial configuration $q_0 ¢ w \$$, finishes by executing an Accept instruction. By $L(M)$ we denote the language consisting of all words accepted by M; we say that M *accepts (recognizes) the language* $L(M)$.

We will also consider some restricted types of restarting automata. An RRWW-automaton is an RLWW-automaton that does not use any MVL operations, and an RWW-automaton is an RRWW-automaton for which each Rewrite step is immediately followed by a Restart step. Further, an RLW-automaton is an RLWW-automaton without auxiliary symbols, and an RL-automaton is an RLW-automaton for which each rewrite operation $(q', v) \in \delta(q, u)$ satisfies the restriction that v is a (scattered) subword of u, that is, u is rewritten into v by simply deleting some of the letters of u. Obviously, the restrictions on the Rewrite operations and those on the movement of the read/write window can be combined, which yields the RRW-, RW-, RR-, and R-automaton.

Each cycle C of a computation of a restarting automaton contains a unique configuration $\alpha q \beta$ in which a Rewrite instruction is applied. Then $|\beta|$ is the *right distance* of C, denoted by $D_r(C)$, and $|\alpha|$ is the *left distance* of C, denoted by $D_l(C)$.

We say that a *sequence of cycles* $Sq = (C_1, C_2, \cdots, C_n)$ is *monotone* (or *right-monotone*) if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$. A *computation* is *monotone* if the corresponding sequence of cycles is monotone. Observe that the tail of the computation does not play any role here. An RLWW-automaton is called *monotone* if all its computations that start with an initial configuration are monotone. The prefix mon- will be used to denote the corresponding classes of restarting automata.

Let j be a positive integer. A sequence of cycles $(C_1, C_2, \ldots, C_n)$ of a restarting automaton is called *j-monotone* if it can be partitioned into j interleaved subsequences that are all monotone [14,15]. Accordingly, a *computation* is *j-monotone* if the corresponding sequence of cycles is j-monotone, and an RLWW-automaton is *j-monotone* if all its computations that start with an initial configuration are j-monotone. It has been shown that with the value of the parameter j the expressive power of j-monotone RRW-automata increases [15]. On the other hand, already 2-monotone R-automata accept NP-complete languages [8], which implies that these automata are already too powerful to admit an efficient implementation. Therefore, we introduce the following weaker notion of j-monotonicity.

Let j be a positive integer. A sequence of cycles $(C_1, C_2, \ldots, C_n)$ of a restarting automaton is called *sequentially j-monotone* if there exist indices $0 = p_0 < p_1 < \cdots < p_j = n$ such that the subsequence $(C_{p_{i-1}+1}, C_{p_{i-1}+2}, \ldots, C_{p_i})$ is monotone for each $i = 1, \ldots, j$. Observe that here the j monotone subsequences follow sequentially one after the other, while for the general notion of j-monotonicity it is allowed that the j subsequences interleave. A restarting automaton is called *sequentially j-monotone* if each of its computations that starts with an initial configuration is sequentially j'-monotone for some $j' \leq j$. We will use the prefix j-s-mon- to denote classes of sequentially j-monotone restarting automata.

For integers i, j, where $0 \leq i \leq j$, we use $[i, j]$ to denote the set of integers $\{i, i+1, \ldots, j\}$, and we take $\sigma^{\geq n} := \{\sigma^i \mid i \geq n\}$, where σ is a word. Also we will identify regular expressions with the regular languages defined by them.

3. Nondeterministic restarting automata without auxiliary symbols

In this section we show that for nondeterministic restarting automata without auxiliary symbols, sequential $(j+1)$-monotonicity is more expressive than sequential j-monotonicity. For proving this result we present a family of example languages L_j $(j \geq 1)$. To define these languages we first introduce a function $\vartheta : (\bigcup_{p \geq 1} \mathbb{N}_+^{2p}) \to \{0, 1\}$ that is defined inductively as follows:

$$\text{For } n, m \in \mathbb{N}_+,\ \vartheta(n, m) := \left\{ \begin{array}{ll} 1 & \text{if } n = m \\ 0 & \text{if } n \neq m \end{array} \right\}, \text{ and for } n, m, r_1, \ldots, r_{2p} \in \mathbb{N}_+,$$

$$\vartheta(n, m, r_1, \ldots, r_{2p}) := \left\{ \begin{array}{ll} 1 & \text{if } n = m \text{ and } \vartheta(r_1, \ldots, r_{2p}) = 1 \\ & \text{or } n > 2m \text{ and } \vartheta(r_1, \ldots, r_{2p}) = 0, \\ 0 & \text{otherwise.} \end{array} \right.$$

The following observations on ϑ will be useful. They follow easily from the above definition.

Lemma 3.1. *Let $j \geq 2$ and $n_1, m_1, \ldots, n_j, m_j \in \mathbb{N}_+$.*

(a) *For $0 < i < j$, if $n_i \neq m_i$ and $n_i \leq 2m_i$, then $\vartheta(n_i, m_i, \ldots, n_j, m_j) = 0$.*

(b) *If $\vartheta(n_i, m_i, \ldots, n_j, m_j) = 0$ for some i satisfying $0 < i \leq j$, and if $n_l \leq 2m_l$ for each $l < i$, then $\vartheta(n_1, m_1, \ldots, n_j, m_j) = 0$.*

Now, for each integer $j \geq 1$, the language L_j on $\Sigma := \{a, b\}$ is defined as

$$L_j := \{\, a^{n_1}b^{m_1} \cdots a^{n_j}b^{m_j} \mid \vartheta(n_1, m_1, \ldots, n_j, m_j) = 1 \,\}.$$

For each $1 \leq i < j$, the value of $\vartheta(n_{i+1}, m_{i+1}, \ldots, n_j, m_j)$ determines which comparison between a^{n_i} and b^{m_i} is to be made. As we will see below, this will force the degree of sequential monotonicity for an RLW-automaton for L_j to be at least j. First, however, we show that the language L_j is accepted by a deterministic RL-automaton that is sequentially j-monotone.

Proposition 3.2. *For each $j \geq 1$, $L_j \in \mathcal{L}$(det-j-s-mon-RL).*

Proof. The language L_j is accepted by a deterministic j-s-mon-RL-automaton M that works as follows. If the word on the tape does not belong to $(a^+b^+)^j$, it is rejected immediately. For a word of the form $(a^+b^+)^j$, we call the factors from a^+b^+ *blocks*. Let $a^{n_1}b^{m_1} \cdots a^{n_j}b^{m_j}$ be the given input. First M determines the last block $a^{n_i}b^{m_i}$ for which $m_i > 1$ and $n_i > 2$ hold. This M can do by first moving its read/write window all the way to the right delimiter \$, and by then moving it back across all those blocks $a^{n_l}b^{m_l}$ for which $m_l = 1$ or $n_l \leq 2$ holds. If $i = j$, then M removes a factor ab from this block and restarts. Otherwise, M can obviously compute the value of $\vartheta(n_{i+1}, m_{i+1}, \ldots, n_j, m_j)$ in its finite control while moving its read/write window to the left.

If $\vartheta(n_{i+1}, m_{i+1}, \ldots, n_j, m_j) = 1$, then M removes a factor ab from the i-th block, otherwise, M removes a factor a^2b from that block. In each case M restarts after performing the rewrite operation.

In this way the tape content is reduced until all blocks satisfy $m_i = 1$ or $n_i \leq 2$. Now M accepts if and only if $\vartheta(n_1, m_1, \ldots, n_j, m_j) = 1$ holds. It follows easily that $L(M) = L_j$. As M processes the input block by block from right to left, each computation of M consists of (at most) j monotone subsequences that follow one after the other. Hence, M is sequentially j-monotone. □

For each RL-automaton, there exists a (nondeterministic) RR-automaton that accepts the same language and that executes exactly the same cycles ([9], Prop. 2.4). Hence, the proposition above has the following immediate consequence.

Corollary 3.3. *For each $j \geq 1$, $L_j \in \mathcal{L}$(j-s-mon-RR).*

Below we will see that, for $j \geq 2$, the language L_j is not accepted by any sequentially $(j-1)$-monotone RLW-automaton. For proving this negative result we need the following pumping lemma for restarting automata (see, *e.g.*, [13]).

Proposition 3.4 (Pumping Lemma)**.** *For each* RLWW*-automaton* $M = (Q, \Sigma, \Gamma, \mathbb{c}, \$, q_0, k, \delta)$*, there exists a constant* $p > \max\{k, |Q|\}$ *such that the following holds. Assume that* $uvw \vdash_M^c uv'w$*, where* $u = u_1u_2u_3$ *and* $|u_2| = p$*. Then there exists a factorization* $u_2 = z_1z_2z_3$ *such that* z_2 *is non-empty, and*

$$u_1z_1(z_2)^iz_3u_3vw \vdash_M^c u_1z_1(z_2)^iz_3u_3v'w$$

holds for all $i > 0$*, that is,* z_2 *is a 'pumping factor' in the above cycle. Similarly, such a pumping factor can be found in any factor of length* p *of* w*. Such a pumping factor can also be found in any factor of length* p *of a word accepted in a tail computation.*

Now we are ready to establish the following result announced above.

Proposition 3.5. *For each* $j \geq 2$*,* $L_j \notin \mathcal{L}((j-1)$-s-mon-RLW$)$*.*

Proof. Let $M = (Q, \Sigma, \Sigma, \mathbb{c}, \$, q_0, k, \delta)$ be an RLW-automaton for L_j, let p be the constant for M from the Pumping Lemma, and let $r := p!$. We say that the i-th block of a word of the form $a^{n_1}b^{m_1}\cdots a^{n_j}b^{m_j}$ is *short* if $n_i < 3r$ or $m_i < 3r$ holds.

Claim 1. Let $a^{n_1}b^{m_1}\cdots a^{n_j}b^{m_j} \vdash_M^c a^{n'_1}b^{m'_1}\cdots a^{n'_j}b^{m'_j}$ be a cycle of a computation of the RLW-automaton M. Then, for each $\alpha_1, \ldots, \alpha_j, \beta_1, \ldots, \beta_j \in \mathbb{N}$, the cycle

$$a^{n_1+\alpha_1r}b^{m_1+\beta_1r}\cdots a^{n_j+\alpha_jr}b^{m_j+\beta_jr} \vdash_M^c a^{n'_1+\alpha_1r}b^{m'_1+\beta_1r}\cdots a^{n'_j+\alpha_jr}b^{m'_j+\beta_jr}$$

is also part of a computation of M, provided that $\alpha_i = \beta_i = 0$ for each i for which the i-th block in $a^{n_1}b^{m_1}\cdots a^{n_j}b^{m_j}$ is short.

Proof of Claim 1. We can apply pumping to each a-syllable and each b-syllable that is longer than $2r$. As r is divisible by the length of any resulting pumping factor, the result follows. □

A word $(a^nb^n)^j$ is called a *basic input* if n is a multiple of r and $n \geq 3r$. Note that each basic input belongs to L_j, and that none of its blocks is short.

Claim 2. Let $a^nb^n\cdots a^nb^n \vdash_M^{c^*} a^{n'_1}b^{m'_1}\cdots a^{n'_j}b^{m'_j}$ be an initial segment of an accepting computation of M on the basic input $w = (a^nb^n)^j \in L_j$, and let $i \in [1, j]$. If the i-th block is not short in any of the configurations during the above computation, then

(a) during this computation no rewrite step is applied to any of the first $i-1$ blocks, that is, $n'_t = m'_t = n$ for each $t < i$;
(b) $n'_i = m'_i \geq 3r$, and each rewrite step that is applied during this computation to the i-th block is of the form $a^sb^s \to a^{s-t}b^{s-t}$ for some $k > s \geq t > 0$.

Proof of Claim 2. This claim is proved by contradiction. So assume that it is not true for some $n \geq 3r$ that is a multiple of r and some $i \in [1, j]$, and let $a^{n'_1}b^{m'_1} \cdots a^{n'_j}b^{m'_j}$ be the first configuration that contradicts this claim, that is,

- $n'_l \neq n$ or $m'_l \neq n$ for some $l < i$, or
- $n'_i \neq m'_i$, or
- some rewrite step that changes one of the first i blocks is *not* of the form $a^s b^s \to a^{s-t}b^{s-t}$ for some $k > s \geq t > 0$.

Observe that the third condition implies that also one of the first two conditions is satisfied. Hence, we concentrate only on the first two conditions.

According to our choice of $a^{n'_1}b^{m'_1} \cdots a^{n'_j}b^{m'_j}$, the computation considered ends with a cycle of the form

$$a^{n''_1}b^{m''_1} \cdots a^{n''_j}b^{m''_j} \vdash_M^c a^{n'_1}b^{m'_1} \cdots a^{n'_j}b^{m'_j},$$

where $n''_t = m''_t = n$ for all $t \in [1, i-1]$ and $n''_i = m''_i \geq 3r$. By analysing this cycle in detail, we will derive the intended contradiction.

Let us note for future reference that, for each $s \leq i$,

$$n'_s < 2m'_s \tag{1}$$

holds. Indeed, a rewrite operation can only change a factor of length at most k, and as $k < r$ and $m'_s \geq 3r$, we see that

$$n'_s \leq n''_s + k = m''_s + k \leq (m'_s + k) + k < 2m'_s.$$

To complete the proof of Claim 2, we now distinguish two cases.

Case 1. $n'_t \neq m'_t$ for some $t \leq i$. Then from Lemma 3.1 and condition (1) we see that $a^{n'_1}b^{m'_1} \cdots a^{n'_j}b^{m'_j} \notin L_j$. This contradicts the fact that we are dealing with an initial segment of an accepting computation of M.

Case 2. $n'_t = m'_t$ for each $t \leq i$, but $n'_l = m'_l \neq n$ for some $l < i$. Let l be the minimal index for which $n'_l \neq n$. As $n''_l = m''_l = n$, it follows that the rewrite operation that is executed in the cycle

$$a^{n''_1}b^{m''_1} \cdots a^{n''_j}b^{m''_j} \vdash_M^c a^{n'_1}b^{m'_1} \cdots a^{n'_j}b^{m'_j}$$

has the form $a^s b^s \to a^{s-u}b^{s-u}$ for some integers s and u satisfying $k > s \geq u > 0$, and that it is applied to the center of the l-th block. Thus, we have the following equalities and inequalities:

$$\begin{array}{ll} n'_l = m'_l = n''_l - u = n - u, & \\ n'_i = m'_i = n''_i = m''_i \geq 3r, & \text{and} \\ n'_t = m'_t = n \geq 3r & \text{for each } t \in [1, i-1] \setminus \{l\}. \end{array}$$

Now we define a word $\bar{v} := a^{\bar{n}_1} b^{\bar{m}_1} \cdots a^{\bar{n}_j} b^{\bar{m}_j}$ that differs from $a^{n''_1} a^{m''_1} \cdots a^{n''_j} a^{m''_j}$ only in the l-th and the i-th blocks as follows:

$$(\bar{n}_t, \bar{m}_t) := \left\{ \begin{array}{ll} (n''_t, m''_t) & \text{for } t > i \\ (n''_t, m''_t) & \text{for } i > t \neq l \\ (n''_t, m''_t + r) & \text{for } t = i \\ (n''_t + n, m''_t) & \text{for } t = l \end{array} \right\} = \left\{ \begin{array}{ll} (n''_t, m''_t) & \text{for } t > i, \\ (n, n) & \text{for } i > t \neq l, \\ (n''_t, n''_t + r) & \text{for } t = i, \\ (2n, n) & \text{for } t = l. \end{array} \right.$$

As $\bar{n}_i \neq \bar{m}_i$ and $\bar{n}_i \leq 2\bar{m}_i$, and as $\bar{n}_t \leq 2\bar{m}_t$ for each $t < i$, Lemma 3.1 implies that $\bar{v} \notin L_j$. On the other hand, from Claim 1 we see that $\bar{v} \vdash^c \hat{v}$, where $\hat{v} := a^{\hat{n}_1} b^{\hat{m}_1} \cdots a^{\hat{n}_j} b^{\hat{m}_j}$ is defined as follows:

$$(\hat{n}_t, \hat{m}_t) := \left\{ \begin{array}{ll} (n'_t, m'_t) & \text{for } t > i \\ (n'_t, m'_t) & \text{for } i > t \neq l \\ (n'_t, m'_t + r) & \text{for } t = i \\ (n'_t + n, m'_t) & \text{for } t = l \end{array} \right\} = \left\{ \begin{array}{ll} (n'_t, m'_t) & \text{for } t > i, \\ (n, n) & \text{for } i > t \neq l, \\ (n''_t, n''_t + r) & \text{for } t = i, \\ (2n - u, n - u) & \text{for } t = l. \end{array} \right.$$

Observe that $\hat{n}_i \neq \hat{m}_i$ and $\hat{n}_i \leq 2\hat{m}_i$. Hence, we obtain $\vartheta(\hat{n}_i, \hat{m}_i, \ldots, \hat{n}_j, \hat{m}_j) = 0$. Now the facts that $\hat{n}_l = 2n - u > 2(n-u) = 2\hat{m}_l$ and that $\hat{n}_t = \hat{m}_t$ for all $i > t \neq l$ imply that $\vartheta(\hat{n}_1, \hat{m}_1, \ldots, \hat{n}_j, \hat{m}_j) = 1$ (see Lem. 3.1). Thus, $\hat{v} \in L_j$, which in turn implies that M will also accept on input $\bar{v}$. This contradicts our assuption that $L(M) = L_j$, therewith completing the proof of Claim 2. □

Using these two claims we will now finish the proof of Proposition 3.5. Let $w = (a^n b^n)^j \in L_j$ be a basic input. We analyse the accepting computations on this input. From Claim 1 we see that w cannot be accepted by M as long as the first block is not short, since otherwise also the input $v \notin L_j$ would be accepted that is obtained from w by replacing the first block by $a^n b^{n+r}$.

On the other hand, Claim 2 implies that the first rewrite step that changes the i-th block is not applied before a configuration has been reached in which the $(i+1)$-st block is short. Thus, each accepting computation first reduces the j-th block into a short block, then the $(j-1)$-st block and so on. Further, the first rewrite operation that changes the i-th block is performed in the center of that block (see Claim 2). Hence, its right distance is larger than the right distance of the previous rewrite step that was applied within the blocks $i+1, \ldots, j$ by (at least) the number $n-k$. Thus, this rewrite step starts a new monotone subsequence. By applying this observation to all blocks, we obtain at least j monotone subsequences that follow sequentially one after the other. Thus, M is not sequentially $(j-1)$-monotone. □

Corollary 3.3 and Proposition 3.5 yield hierarchy results with respect to the degree of sequential monotonicity for RR(W)- and for RL(W)-automata. In order to also obtain corresponding hierarchies for R(W)-automata, we need a somewhat more involved variant of the language L_j.

Let $\Sigma := \{a, b, c, d\}$ be the new terminal alphabet. For a subset $\Delta \subseteq \Sigma$, we denote the projection from Σ^* onto Δ^* by π_Δ, that is, for $x \in \Sigma^*$, $\pi_\Delta(x)$ is

obtained from x by removing all occurrences of symbols from $\Sigma \setminus \Delta$. For $j \geq 1$, the language $L'_j \subseteq \Sigma^*$ is defined as follows:

$$\begin{aligned} L'_j := \{\, xy \mid\ & \exists\, i \in [0,j]\, \exists\, n_1, m_1, \ldots, n_j, m_j \in \mathbb{N}_+\, \exists\, y_{i+1}, \ldots, y_j \in \{c,d\}: \\ & x = a^{n_1} b^{m_1} \cdots a^{n_i} b^{m_i} \text{ and } y = a^{n_{i+1}} y_{i+1} b^{m_{i+1}} \cdots a^{n_j} y_j b^{m_j} \\ & \text{such that } \ (i)\ \ \pi_{\{a,b\}}(xy) \in L_j, \\ & \qquad\quad (ii)\ \ n_l < 3 \text{ or } m_l < 2 \text{ for each } l > i+1, \\ & \qquad\quad (iii)\ \ n_l \geq 3 \text{ and } m_l \geq 2 \text{ for each } l \leq i, \text{ and} \\ & \qquad\quad (iv)\ \ \text{for all } l \in [i+1,j],\ y_l = c \text{ if and only if} \\ & \qquad\qquad\ \ l = j \text{ or } \vartheta(n_{l+1}, m_{l+1}, \ldots, n_j, m_j) = 1 \,\}. \end{aligned}$$

The idea is that the symbol c is used to "mark" those blocks for which it should be checked whether $n_l = m_l$ holds, and the symbol d is used to "mark" those blocks for which it should be checked whether $n_l > 2m_l$ holds. Further, c and d only occur in a number of "suffix blocks", which will force the restarting automaton to process the blocks from right to left. For the language L'_j we have the following positive result.

Proposition 3.6. *For each $j \geq 1$, $L'_j \in \mathcal{L}(j\text{-s-mon-RW})$.*

Proof. We describe a sequentially j-monotone RW-automaton M that accepts the language L'_j. This automaton works as follows. Whenever the prefix of the tape content read by M during a cycle is not a prefix of the regular language $(a^{\geq 3} \cdot b^{\geq 2})^j \cup \bigcup_{i=0}^{j-1} R_i$, where, for all $i \in [0, j-1]$,

$$R_i := \left(a^{\geq 3} \cdot b^{\geq 2}\right)^i \cdot \left(a^+ \cdot \{c,d\} \cdot b^+\right) \cdot \left(a^{<3} \cdot \{c,d\} \cdot b^+ \cup a^+ \cdot \{c,d\} \cdot b^{<2}\right)^{j-i-1},$$

then M rejects immediately. Otherwise, in each cycle M chooses nondeterministically a block to which the next rewrite operation is to be applied, and which of the conditions $n_l = m_l$ or $n_l > 2m_l$ should be checked for this block (the l-th block). In the former case it applies the rewrite transition $aabb \to acb$ or $aacbb \to acb$ in the middle of this block, and in the latter case it applies the rewrite transition $a^3b^2 \to adb$ or $a^3db^2 \to adb$ at that position. If none of these rewrite steps is possible or a block preceding the chosen block does not belong to $a^{\geq 3} \cdot b^{\geq 2}$, then M rejects.

Finally, M may decide nondeterministically that it is already executing the tail of a computation. In this case M does not execute any rewrite step, but it checks whether the current tape content belongs to the regular language

$$\{\, a^{n_1} y_1 b^{m_1} \cdots a^{n_j} y_j b^{m_j} \mid y_i \in \{c,d\},\ n_i < 3 \text{ or } m_i < 2 \text{ for each } i \in [1,j] \,\}.$$

In the negative, M rejects. In the affirmative M accepts if $a^{n_1} y_1 b^{m_1} \cdots a^{n_j} y_j b^{m_j}$ belongs to the language L'_j, which is the case if and only if the following conditions are satisfied simultaneously:

- for all $i \in [1,j]$, $y_i = c$ if and only if $i = j$ or $\vartheta(n_{i+1}, m_{i+1}, \ldots, n_j, m_j) = 1$,
- $n_1 = m_1$, if $y_1 = c$, or $n_1 > 2m_1$, if $y_1 = d$.

Because of the size restriction of all these blocks, M can verify these conditions while scanning the tape from left to right.

It is easily verified that in this way M accepts the language L'_j, and that M is sequentially j-monotone. Actually each computation consists of j monotone sequences during which rewrite operations are applied to the j-th block, the $(j-1)$-st block, and so on up to the first block, or it stops early in case an inappropriate block is chosen in any cycle or the tail of the computation is entered too early. □

Contrasting the above result we have the following negative result.

Proposition 3.7. *For each $j \geq 2$, $L'_j \notin \mathcal{L}((j-1)$-s-mon-RLW$)$.*

Proof. Let $j \geq 2$, and $M = (Q, \Sigma, \Sigma, ¢, \$, q_0, k, \delta)$ be an RLW-automaton that accepts the language L'_j. We analyse the accepting computations of M on those inputs from L'_j which initially do not contain any occurences of the symbols c or d.

Note that Claim 1 from the proof of Proposition 3.5 also holds for M. In fact, it even applies if some blocks of the restarting configuration considered belong to the regular language $a^+ \cdot c \cdot b^+$ or $a^+ \cdot d \cdot b^+$. Moreover, Claim 2 also holds for L'_j in those cases in which there are no occurrences of the symbols c or d in the first i blocks of the configuration considered. Hence, starting from a basic input $w = (a^n b^n)^j \in L'_j$, we can follow the proof of Proposition 3.5. The only difference is that now the automaton M may insert a c or a d in some block and after that we cannot apply Claim 2 to prefixes containing this block. However, the tape content of each configuration during an accepting computation must belong to L'_j. Thus, the symbols c and d can be inserted only from right to left, first in the center of the j-th block, then in the center of the $(j-1)$-st block, and so on. If c or d is inserted into the i-th block, while this block is not yet short, the rewrite step that inserts this symbol has a larger right distance than the last rewrite step in the $(i+1)$-st block preceding it (as the size of the window, k, is much smaller than r). On the other hand, as long as the first i blocks do not contain any occurrences of c or d, Claims 1 and 2 ensure that the rewrite steps up to the configuration in which the i-th block becomes short require at least $j-i+1$ sequentially monotone sequences. It follows that the RLW-automaton M is certainly not sequentially $(j-1)$-monotone. □

Propositions 3.6 and 3.7 establish a hierarchy result for RW-automata. In order to obtain the corresponding result also for R-automata, we derive the following technical result.

Let M be a sequentially j-monotone RW-automaton for the language L'_j, let k be the size of its read/write window, and let m be the size of its tape alphabet (that is, $m = 4$). Finally, let $L''_j := \varphi_{k,m}(L'_j)$, where $\varphi_{k,j}$ is the encoding defined in [8], Section 3.

Proposition 3.8. *For each $j \geq 2$, $L''_j \in \mathcal{L}(j$-s-mon-R$) \setminus \mathcal{L}((j-1)$-s-mon-RLW$)$.*

Proof. From [8] Theorem 2 it follows immediately that $L''_j \in \mathcal{L}(j$-s-mon-R$)$ holds.

In order to show that $L''_j \notin \mathcal{L}((j-1)$-s-mon-RLW$)$ by contradiction, we assume that there exists a sequentially $(j-1)$-monotone RLW-automaton M accepting

the language L''_j. From M we now construct a sequentially $(j-1)$-monotone RLW-automaton M' such that $L(M') = L'_j$, thus contradicting Proposition 3.7.

On a tape content $x \in \Sigma^*$, M' simulates a computation of M on the tape content $\varphi_{k,m}(x)$. Whenever M intends to perform a rewrite step that converts the tape content into a string that is not of the form $\varphi_{k,m}(y)$ for any $y \in \Sigma^*$, M' rejects, as by the correctness preserving property (see, *e.g.*, [13]) this rewrite step clearly indicates that the computation of M being simulated is not accepting. In all other cases M' is able to 'mimic' the corresponding steps of M. It is easily verified that in this way M' recognizes the language L'_j, and that M' is sequentially $(j-1)$-monotone if M is sequentially $(j-1)$-monotone. □

As a consequence of all the results above we obtain the following theorem.

Theorem 3.9. *For each $j \in \mathbb{N}_+$ and each $X \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RR}, \mathsf{RRW}, \mathsf{RL}, \mathsf{RLW}\}$,*

(a) $\mathcal{L}(j\text{-s-mon-X}) \subsetneq \mathcal{L}((j+1)\text{-s-mon-X})$;

(b) $\mathcal{L}((j+1)\text{-s-mon-R}) \setminus \mathcal{L}(j\text{-s-mon-RLW}) \neq \emptyset$.

Observe, however, that the language

$$L := \{f, ee\} \cdot \{\, a^n b^n \mid n \geq 0 \,\} \cup \{g, ee\} \cdot \{\, a^n b^m \mid m > 2n \geq 0 \,\}$$

is not accepted by any RR-automaton, although it is accepted by a monotone RW-automaton [6]. Thus, $\mathcal{L}(\text{mon-RW}) \not\subset \bigcup_{j\geq 1} \mathcal{L}(j\text{-mon-RR})$, which implies that $\mathcal{L}(j\text{-s-mon-RLW})$ is neither contained in $\mathcal{L}((j+1)\text{-s-mon-R})$ nor in $\mathcal{L}(j\text{-mon-R})$. In particular, the language classes $\mathcal{L}(j\text{-s-mon-RLW})$ and $\mathcal{L}((j+1)\text{-s-mon-R})$ are incomparable under inclusion.

In [9] it is established for each $j \geq 2$ that the language

$$\bar{L}_j := \{\, a^{n_1} b^{n_1} a^{n_2} b^{n_2} \cdots a^{n_j} b^{n_j} \mid n_1 \geq n_2 \geq \cdots n_j \geq 1 \,\}$$

belongs to the difference $\mathcal{L}(j\text{-mon-R}) \setminus \mathcal{L}((j-1)\text{-mon-RLW})$. In fact, in the proof of the noninclusion result it is shown that an RLW-automaton M for the language $\bar{L}_j$ must reduce all blocks from a^+b^+ in an almost synchronous manner. Accordingly, M cannot possibly be sequentially i-monotone for any $i \geq 1$, that is,

$$\bar{L}_j \in \mathcal{L}(j\text{-mon-R}) \setminus \bigcup_{i\geq 1} \mathcal{L}(i\text{-s-mon-RLW}).$$

The proofs of Propositions 3.5, 3.7, and 3.8 show that the language L''_j is not accepted by any RLW-automaton that is $(j-1)$-monotone. Thus,

$$L''_{j+1} \in \mathcal{L}((j+1)\text{-s-mon-R}) \setminus \mathcal{L}(j\text{-mon-RLW}).$$

Together with the above result on the language $\bar{L}_j$ this yields the following results.

Corollary 3.10. *For each $j \geq 1$ and each $\mathsf{X} \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RR}, \mathsf{RRW}, \mathsf{RL}, \mathsf{RLW}\}$, $\mathcal{L}((j+1)\text{-s-mon-X})$ and $\mathcal{L}(j\text{-mon-X})$ are incomparable under inclusion. In particular, $\mathcal{L}(j\text{-s-mon-X}) \subsetneq \mathcal{L}(j\text{-mon-X})$.*

4. Nondeterministic restarting automata with auxiliary symbols

It is known that the classes of languages $\mathcal{L}(\mathsf{mon\text{-}R(R)WW})$ and $\mathcal{L}(\mathsf{mon\text{-}RLWW})$ coincide with the class CFL of context-free languages [6]. Here we will show that this characterization extends to the classes of languages that are accepted by sequentially j-monotone $\mathsf{R(R)WW}$- and RLWW-automata for all $j \geq 2$. For establishing this result we will proceed as follows.

With an RRWW-automaton M with tape alphabet Γ we will associate the set $L_{\mathrm{mon}}(M)$ of words over Γ that M accepts by monotone computations, and the mapping f_M that maps each word $w \in \Gamma^*$ to the set of words that M can reach from w by performing a monotone computation. The set $L_{\mathrm{mon}}(M)$ is context-free, and we can obtain from M a pushdown automaton P_M and an alphabetic morphism h such that, for each $w \in \Gamma^*$, the set $f_M(w)$ coincides with the image under h of the set of final stack contents that P_M can generate on input w (Lem. 4.2). Based on this characterization we will then derive the fact that, for each $j \geq 2$, the language $L^{(j)}_{\mathrm{mon}}(M)$ of words $w \in \Gamma^*$ that M accepts by sequentially j-monotone computations is also context-free (Th. 4.3). If M is sequentially j-monotone, then $L(M) = L^{(j)}_{\mathrm{mon}}(M) \cap \Sigma^*$, where Σ is the input alphabet of M, which then yields the announced result. For this approach to work we need the following technical result on pushdown automata.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ be a pushdown automaton (PDA) with input alphabet Σ and stack alphabet $\Gamma \cup \{\#\}$, where $\#$ denotes the bottom marker of the pushdown store, Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is the transition relation. As usual we describe configurations of P by triples of the form (q, u, α), where $q \in Q$ denotes the current state, $u \in \Sigma^*$ is the still unread part of the input, and $\alpha \in \# \cdot \Gamma^*$ is the content of the pushdown store, where the first symbol of α is on the bottom of the pushdown store, and the last letter of α is the symbol on top of the pushdown store.

For a word $u \in \Sigma^*$,

$$\mathsf{SC}_P(u) := \{\, w \in \Gamma^* \mid \exists\, q \in F : (q_0, u, \#) \vdash_P^* (q, \varepsilon, \#w) \,\}$$

is the *language of final stack contents* that P can generate on input u. For a language $L \subseteq \Sigma^*$, $\mathsf{SC}_P(L) := \bigcup_{u \in L} \mathsf{SC}_P(u)$. It is well-known that the language $\mathsf{SC}_P(L)$ is regular for each regular language L [4].

Lemma 4.1. *Let P be a pushdown automaton with input alphabet Σ and pushdown alphabet $\Gamma \cup \{\#\}$, and let S be a subset of $\mathsf{SC}_P(\Sigma^*)$. If S is context-free, then so is the language $\mathsf{SC}_P^{-1}(S) := \{\, u \in \Sigma^* \mid \mathsf{SC}_P(u) \cap S \neq \emptyset \,\}$.*

Proof. Let $\bar{\Gamma}$ be a new alphabet in one-to-one correspondence to Γ such that Γ and $\bar{\Gamma}$ are disjoint. Further, let $\bar{\#}$ be another new symbol, and let R be the finite string-rewriting system

$$R := \{\, b\bar{b} \to \varepsilon \mid b \in \Gamma \cup \{\#\} \,\}$$

on $\Omega := \Gamma \cup \bar{\Gamma} \cup \{\#, \bar{\#}\}$. Then R is a special system that is confluent (see, *e.g.*, [2]). For $w \in \Omega^*$, $\Delta_R^*(w)$ is the set of all descendants of w with respect to the reduction relation induced by R, that is,

$$\Delta_R^*(w) := \{\, z \in \Omega^* \mid w \to_R^* z \,\},$$

and $\Delta_R^*(T) := \bigcup_{w \in T} \Delta_R^*(w)$ for each subset $T \subseteq \Omega^*$. It is well-known that the set $\Delta_R^*(T)$ is regular, whenever T is a regular language.

From the PDA P one easily obtains a finite-state transducer (FST) B with ε-transitions, input alphabet Σ, and output alphabet Ω that imitates the behaviour of P step by step as follows. Whenever P performs a transition $(q, a, b) \to (q', w)$, where q, q' are internal states of P, $a \in \Sigma \cup \{\varepsilon\}$ is the input symbol read, $b \in \Gamma$ is the topmost symbol on P's pushdown store, and $w \in \Gamma^*$ is the pushdown word by which the symbol b is replaced, then B executes the transition $(q, a) \to (q', \bar{b}w)$, that is, B performs the same input and change-of-state step as P, but it simulates the pushdown operation $b \mapsto w$ by producing the output $\bar{b}w$, which means that B just guesses the topmost symbol on the pushdown store of P. The string-rewriting system R is then used to verify that this guess is correct by matching the output symbol $\bar{b}$ to the last output symbol produced previously. It can be shown that the following equality holds for each word $u \in \Sigma^*$ (see [3] for the details):

$$\Delta_R^*(B(u)) \cap \bar{\#}\# \cdot \Gamma^* = \bar{\#}\# \cdot \mathsf{SC}_P(u).$$

Now let S be a subset of $\mathsf{SC}_P(\Sigma^*)$. Then a word $u \in \Sigma^*$ belongs to the set $\mathsf{SC}_P^{-1}(S)$ if and only if $\Delta_R^*(B(u)) \cap \bar{\#}\# \cdot S \neq \emptyset$ holds.

Let $\nabla_R^*(w)$ denote the set of ancestors of w with respect to the reduction relation induced by R, that is,

$$\nabla_R^*(w) := \{\, x \in \Omega^* \mid x \to_R^* w \,\},$$

and for $T \subseteq \Omega^*$, $\nabla_R^*(T) := \bigcup_{w \in T} \nabla_R^*(w)$. Then we obtain the following equality:

$$\mathsf{SC}_P^{-1}(S) = B^{-1}(\nabla_R^*(\bar{\#}\# \cdot S)).$$

Now if S is a context-free language, then so is the language $\bar{\#}\# \cdot S$. From the form of the rules of R we see immediately that then $\nabla_R^*(\bar{\#}\# \cdot S)$ is context-free, too, which implies that $B^{-1}(\nabla_R^*(\bar{\#}\# \cdot S))$ is context-free, as the class of context-free languages is closed under (inverse) finite transductions [1]. Thus, we see that $\mathsf{SC}_P^{-1}(S)$ is context-free. □

Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an RRWW-automaton. With M we associate the language

$$L_{\mathrm{mon}}(M) := \{\, w \in \Gamma^* \mid M \text{ accepts } w \text{ by a monotone computation} \,\},$$

and the mapping $f_M : \Gamma^* \to \mathfrak{P}(\Gamma^*)$ that is defined as follows:

$$f_M(w) := \{\, y \in \Gamma^* \mid \text{There exists a monotone sequence of cycles of } M \text{ that starts from the restarting configuration } q_0 ¢ w \$ \text{ and ends with the restarting configuration } q_0 ¢ y \$ \,\}.$$

Observe that the language $f_M(\Gamma^*)$ is simply the set Γ^*, as from each restarting configuration, a monotone computation of M originates, which, however, may consist of a single cycle only, or which may even consist of no cycle at all.

Lemma 4.2. *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an* RRWW*-automaton. Then the following statements hold:*

(a) *The language $L_{\mathrm{mon}}(M)$ is context-free.*

(b) *There exists a pushdown automaton P_M with input alphabet Σ and pushdown alphabet Θ, and an alphabetic morphism $h : \Theta^* \to \Gamma^*$ such that, for each word $w \in \Gamma^*$, $h(\mathsf{SC}_{P_M}(w)) = f_M(w)$ holds.*

Proof.
(a) In [6] it is shown that the language $L(M)$ that is accepted by a monotone RRWW-automaton M is necessarily context-free by presenting a simulation of M by a PDA P'_M. Even if the RRWW-automaton M is not monotone, then, given a word $w \in \Gamma^*$, this PDA simulates the monotone computations of M that originate from the restarting configuration $q_0 ¢ w \$$. Thus, this PDA accepts the language $L_{\mathrm{mon}}(M)$, which means that this language is indeed context-free.

(b) The PDA P'_M above simulates the monotone initial parts of all computations of M. It has input alphabet Σ and pushdown alphabet $\Theta' := \Gamma \times \mathfrak{P}(Q)$, as it uses its pushdown to store a prefix u of the tape content of M together with information on the subset of states of M that M can reach from its initial state q_0 by reading the prefix u. We modify P'_M in such a way that, each time it starts the simulation of a cycle of M, it may decide (nondeterministically) to abort the simulation process. In this case it pushes the remaining suffix of the input onto the pushdown store, and halts and accepts. Hence, the resulting PDA P_M has input alphabet Σ and pushdown alphabet $\Theta := \Theta' \cup \Gamma$. We define an alphabetic morphism $h : \Theta^* \to \Gamma^*$ through $a \mapsto a$ and $(a, Q') \mapsto a$ for each $a \in \Gamma$ and $Q' \subseteq Q$. Then $f_M(w) = h(\mathsf{SC}_{P_M}(w))$ holds for each word $w \in \Gamma^*$. □

Let M be an RRWW-automaton with tape alphabet Γ. For each integer $j \geq 1$, we denote by $L^{(j)}_{\mathrm{mon}}(M)$ the language

$$L^{(j)}_{\mathrm{mon}}(M) := \{\, y \in \Gamma^* \mid M \text{ accepts } y \text{ by a sequentially } j\text{-mon. computation} \,\}.$$

Then, for each word $w \in \Gamma^*$ and each number $j \geq 2$,

$$w \in L^{(j)}_{\mathrm{mon}}(M) \text{ if and only if } f_M(w) \cap L^{(j-1)}_{\mathrm{mon}}(M) \neq \emptyset.$$

The main result of this section is an immediate consequence of the following technical result.

Theorem 4.3. *$L^{(j)}_{\mathrm{mon}}(M)$ is a context-free language for each* RRWW*-automaton M and each integer $j \geq 1$.*

Proof. We proceed by induction on j. For $j = 1$ this is just Lemma 4.2 (a), as sequential 1-monotonicity coincides with monotonicity. So assume that $j > 1$. From the induction hypothesis we get that $L^{(j-1)}_{\mathrm{mon}}(M)$ is a context-free language. Thus, $L^{(j-1)}_{\mathrm{mon}}(M)$ is a context-free subset of the language $f_M(\Gamma^*)$, which coincides with the set $h(\mathsf{SC}_{P_M}(\Gamma^*))$ for a PDA P_M and an alphabetic morphism that are obtained from M according to Lemma 4.2 (b).

As observed above, $w \in L^{(j)}_{\mathrm{mon}}(M)$ if and only if $f_M(w) \cap L^{(j-1)}_{\mathrm{mon}}(M) \neq \emptyset$. Thus,

$$\begin{aligned} L^{(j)}_{\mathrm{mon}}(M) &= \{\, w \in \Gamma^* \mid f_M(w) \cap L^{(j-1)}_{\mathrm{mon}}(M) \neq \emptyset \,\} \\ &= \{\, w \in \Gamma^* \mid h(\mathsf{SC}_{P_M}(w)) \cap L^{(j-1)}_{\mathrm{mon}}(M) \neq \emptyset \,\} \\ &= \mathsf{SC}^{-1}_{P_M}(h^{-1}(L^{(j-1)}_{\mathrm{mon}}(M))), \end{aligned}$$

implying that the language $L^{(j)}_{\mathrm{mon}}(M)$ is context-free by Lemma 4.1. □

If the RRWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is sequentially j-monotone, then $L(M) = L^{(j)}_{\mathrm{mon}}(M) \cap \Sigma^*$. By the result above this yields that $L(M)$ is context-free. As each context-free language is accepted by a monotone RWW-automaton [6], we obtain the following characterization.

Corollary 4.4. $\mathcal{L}(j\text{-s-mon-RRWW}) = \mathcal{L}(j\text{-s-mon-RWW}) = \mathsf{CFL}$ *for all* $j \geq 1$.

For each RLWW-automaton, there exists an RRWW-automaton that accepts the same language and that executes exactly the same cycles ([9], Prop. 2.4). Hence, we also get the following result.

Corollary 4.5. $\mathcal{L}(j\text{-s-mon-RLWW}) = \mathsf{CFL}$ *for all* $j \geq 1$.

Thus, for all types of nondeterministic restarting automata with auxiliary symbols the degree of sequential monotonicity does not yield a hierarchy in contrast to the situation for restarting automata without auxiliary symbols.

Finally observe that the context-free language

$$L := \{\, a^n b^n \mid n \geq 0 \,\} \cup \{\, a^n b^m \mid m > 2n \geq 0 \,\}$$

is not accepted by any RLW-automaton [6], which implies by the results above that $\bigcup_{j \geq 1} \mathcal{L}(j\text{-s-mon-RLW})$ is a proper subclass of CFL.

5. Deterministic restarting automata

It is known that all levels of j-monotonicity for deterministic R(R)(W)(W)-automata collapse to DCFL [9]. As each sequentially j-monotone computation is also j-monotone, we obtain the following result.

Corollary 5.1. *For each* $j \in \mathbb{N}_+$ *and each* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$,

$$\mathcal{L}(\text{det-}j\text{-s-mon-X}) = \mathsf{DCFL}.$$

Monotone deterministic RL-automata recognize some languages which are not in DCFL [14]. Further, we see from Propositions 3.2 and 3.5 that, for each $j \geq 2$, $L_j \in \mathcal{L}(\text{det-}j\text{-s-mon-RL}) \setminus \mathcal{L}(\text{det-}(j-1)\text{-s-mon-RLW})$. This yields the following hierarchies.

Theorem 5.2. *For each* $j \in \mathbb{N}_+$ *and* $X \in \{\mathsf{RL}, \mathsf{RLW}\}$,

$$\mathcal{L}(\text{det-}j\text{-s-mon-X}) \subsetneq \mathcal{L}(\text{det-}(j+1)\text{-s-mon-X}).$$

6. Concluding remarks

We have seen that sequential j-monotonicity yields interesting extensions for those language classes that are defined by monotone restarting automata without auxiliary symbols without losing the efficiency of the solution to the membership problem. On the other hand, it turned out that all degrees of sequential monotonicity collapse to the first level for restarting automata with auxiliary symbols. This fact may be interesting in its own right as it gives a much less restricted machine model for CFL than the pushdown automaton. However, this result says that the expressive capability of sequentially j-monotone restarting automata is not sufficient to cover many important aspects of the analysis by reduction. Thus, other generalizations of monotonicity are needed.

An interesting open problem is whether sequential j-monotonicity gives an infinite hierarchy for deterministic RLWW-automata. Our feeling is that it might be possible to establish such a hierarchy with some technical effort. It would give the first example of a strict hierarchy for restarting automata that use auxiliary symbols.

Further, by requiring that the left distance must not increase from one cycle to the next in a computation, the notion of *left-monotonicity* was introduced for restarting automata in [9]. There also the generalization to j-left-monotonicity was considered. Now it is straightforward to also introduce the sequential variant of j-left-monotonicity. It appears that the results obtained in this paper for nondeterministic restarting automata with and without auxiliary symbols should carry over to sequential left-monotonicity. However, for deterministic restarting automata without auxiliary symbols, j-left-monotonicity yields infinite hierarchies, and even in the presence of auxiliary symbols, it is known that 2-left-monotonicity is more expressive than left-monotonicity. Is it possible to establish corresponding hierarchy results also for sequentially left-monotone deterministic restarting automata?

References

[1] J. Berstel. *Transductions and Context-Free Languages*. Teubner, Stuttgart (1979).

[2] R.V. Book and F. Otto, *String-Rewriting Systems*. Springer, New York (1993).

[3] R. Cremanns and F. Otto, The language of final stack contents of a pushdown automaton is effectively regular. *Mathematische Schriften Kassel* 11/93, Universität Kassel (1993).

[4] S. Greibach, A note on pushdown store automata and regular systems. *Proc. Amer. Math. Soc.* **18** (1967) 263–268.

[5] P. Jančar, F. Mráz, M. Plátek and J. Vogel. Restarting automata, in *FCT'95, Proc.*, edited by H. Reichel, Springer, Berlin. *Lect. Notes Comput. Sci.* **965** (1995) 283–292.

[6] P. Jančar, F. Mráz, M. Plátek and J. Vogel, On monotonic automata with a restart operation. *J. Autom. Lang. Comb.* **4** (1999) 287–311.

[7] T. Jurdziński, K. Loryś, G. Niemann and F. Otto, Some results on RWW- and RRWW-automata and their relation to the class of growing context-sensitive languages. *J. Autom. Lang. Comb.* **9** (2004) 407–437.

[8] T. Jurdziński, F. Otto, F. Mráz and M. Plátek, On the complexity of 2-monotone restarting automata, in *DLT 2004, Proc.*, edited by C.S. Calude, E. Calude and M.J. Dinneen, Springer, Berlin. *Lect. Notes Comput. Sci.* **3340** (2004) 237–248.

[9] T. Jurdziński, F. Mráz, F. Otto and M. Plátek, Degrees of non-monotonicity for restarting automata. *Theoret. Comput. Sci.* **369** (2006) 1–34.

[10] G. Niemann and F. Otto, Further results on restarting automata, in *Words, Languages and Combinatorics III, Proc.*, edited by M. Ito and T. Imaoka. World Scientific, Singapore (2003) 352–369.

[11] K. Oliva, P. Květoň and R. Ondruška, The computational complexity of rule-based part-of-speech tagging, in *TSD 2003, Proc.*, edited by V. Matousek and P. Mautner, Springer, Berlin. *Lect. Notes Comput. Sci.* **2807** (2003) 82–89.

[12] F. Otto, Restarting automata and their relations to the Chomsky hierarchy, in *Developments in Language Theory, Proc. DLT'2003*, edited by Z. Esik and Z. Fülöp, Springer, Berlin. *Lect. Notes Comput. Sci.* **2710** (2003) 55–74.

[13] F. Otto, Restarting Automata, in *Recent Advances in Formal Languages and Applications*, Z. Ésik, C. Martin-Vide and V. Mitrana (Eds.), Springer, Berlin. *Stud. Comput. Intelligence* **25** (2006) 269–303.

[14] M. Plátek, Two-way restarting automata and j-monotonicity, in *Theory and Practice of Informatics, Proc. SOFSEM 2001*, edited by L. Pacholski and P. Ružička, Springer-Verlag, Berlin. *Lect. Notes Comput. Sci.* **2234** (2001) 316–325.

[15] M. Plátek and F. Mráz, Degrees of (non)monotonicity of RRW-automata, in *Preproceedings of the 3rd Workshop on Descriptional Complexity of Automata, Grammars and Related Structures*, Report No. 16, edited by J. Dassow and D. Wotschke, Fakultät für Informatik, Universität Magdeburg (2001) 159–165.

[16] M. Plátek, M. Lopatková and K. Oliva, Restarting automata: motivations and applications. In *Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten', Proc.*, edited by M. Holzer, Institut für Informatik, Technische Universität München (2003) 90–96.

Communicated by J. Berstel.
Received November 17, 2005. Accepted June 15,2006.