Fundamental Study

Degrees of non-monotonicity for restarting automata

T. Jurdziński\textsuperscript{a}, F. Mráz\textsuperscript{b}, F. Otto\textsuperscript{c,\ast}, M. Plátek\textsuperscript{b}

\textsuperscript{a}Institute of Computer Science, University of Wrocław, 51-151 Wrocław, Poland
\textsuperscript{b}Charles University, Faculty of Mathematics and Physics, Department of Computer Science, Malostranské nám. 25, 11800 Praha 1, Czech Republic
\textsuperscript{c}Fachbereich Mathematik/Informatik, Universität Kassel, 34109 Kassel, Germany

Received 11 January 2005; received in revised form 26 July 2006; accepted 25 August 2006

Communicated by Z. Esik

Abstract

In the literature various notions of monotonicity for restarting automata have been studied. Here we introduce two new variants of monotonicity for restarting automata and for two-way restarting automata: left-monotonicity and right-left-monotonicity. It is shown that for the various types of deterministic and nondeterministic (two-way) restarting automata without auxiliary symbols, these notions yield infinite hierarchies, and we compare these hierarchies to each other. Further, as a tool used to simplify some of the proofs, the shrinking restarting automaton is introduced, which is a generalization of the standard (length-reducing) restarting automaton to the weight-reducing case. Some of the consequences of this generalization are also discussed.

© 2006 Elsevier B.V. All rights reserved.

MSC: 68Q45; 68Q10

Keywords: Restarting automata; Monotonicity; Degree of monotonicity

1. Introduction

In his work Noam Chomsky proposed \textit{generative grammars} as a device for generating languages, \textit{analytical grammars} as a device for analyzing languages, and \textit{automata} as a device for recognizing languages \cite{3}. \textit{Categorial grammars}, the oldest type of formal grammars, are analytical grammars (see, e.g., \cite{15}). \textit{Marcus contextual grammars}, on the other hand, are generative devices that describe the process of generating a language completely without using non-terminals \cite{14}. Through various means the constraints can be realized that regulate this process (see, e.g., \cite{26}). An analytical device that in some sense is complementary to Marcus contextual grammars are the \textit{restarting automata}, introduced in \cite{7}. They are analyzers that implement basic as well as enhanced features of analytical grammars. To each sentence

\textsuperscript{*}Some of the results of this paper were presented at DCFS’03 in Budapest, July 2003, and at DLT’04 in Auckland, December 2004. This work was mainly carried out while T. Jurdziński was visiting the University of Kassel, supported by a grant from the Deutsche Forschungsgemeinschaft (DFG). F. Mráz and M. Plátek were partially supported by the Grant Agency of the Czech Republic under Grant no. 201/04/2102 and by the program “Information Society” under project 1ET100300517. F. Mráz was also supported by the Grant Agency of Charles University in Prague under Grant no. 358/2006/A-INF/MFF.

\ast Corresponding author. Tel.: +49 561 804-4573; fax: +49 561 804-4008.

\textit{E-mail addresses:} tju@ii.uni.wroc.pl (T. Jurdziński), mraz@ksvi.ms.mff.cuni.cz (F. Mráz), otto@theory.informatik.uni-kassel.de (F. Otto), Martin.Platek@mff.cuni.cz (M. Plátek).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.08.029
of the language recognized, a restarting automaton associates all the corresponding derivations (through sequences of reduction steps). These reduction steps preserve the so-called “error preserving property” for the sentential forms derived. This is an important property that imitates a similar property of analytical grammars. It distinguishes restarting automata from all other types of automata. The relationships between restarting automata and Marcus contextual grammars have been investigated in several papers (see, e.g., [13]).

Restarting automata offer numerous options for implementing regulations (constraints) for their reduction steps (see, e.g., [4]). Here we study the influence of constraints that enforce some monotonicity conditions on the sequences of reduction steps.

The original motivation for introducing the restarting automaton in [7] was the desire to model the so-called analysis by reduction of natural languages. In fact, many aspects of the work on restarting automata are motivated by the basic tasks of computational linguistics (e.g., devising multi-level language descriptions) as well as by applied tasks (e.g., constructing grammar checkers for free word-order languages). The aim of this paper is to enrich the taxonomy of word-order constraints given in [27]. The word-order constraints play an important role in modern computational linguistics (see, e.g., [5]).

A (two-way) restarting automaton, RLWW-automaton for short, is a device $M$ with a finite-state control and a read/write window of a fixed size. This window is moved along a flexible tape containing a word delimited by sentinels by performing move-right and move-left operations until the control decides (nondeterministically) that the content of the window should be rewritten by some shorter string. After a rewrite, $M$ continues to move its window until it either halts and accepts, halts and rejects, or restarts, which means that 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, where a cycle is the part of a computation of $M$ from one restart step (or from an initial configuration) to the next restart step. Here it is important to notice that between any two rewrite steps, a restart step must be executed, and so the restart operation is of central importance even for the two-way restarting automaton. In fact, $M$ cannot only be considered as a device for accepting a language, but it can also be interpreted as a “rewriting system,” as each cycle replaces a factor of the tape content by a shorter factor, in this way performing a “rewrite” of the tape content. Actually in [23] various types of restarting automata have been described by certain types of infinite prefix-rewriting systems.

As each rewrite step shortens the length of the tape content, it is obvious that $M$ can be simulated in polynomial time by a nondeterministic Turing machine. Thus, the class of languages that are accepted by RLWW-automata is contained in the complexity class NP as well as in the language class CSL of context-sensitive languages.

Also various restricted versions of the restarting automaton have been considered. Here the RRWW-automaton, which can only move its window from left to right along the tape, and the RWW-automaton, which is in addition required to perform a restart step immediately after executing a rewrite operation are of particular interest. In general, a restarting automaton can use so-called auxiliary symbols in addition to the input symbols in its rewrite operations, but also various types of restarting automata without auxiliary symbols have been studied. It has been shown that deterministic RWW- and RRWW-automata characterize the class CRL of Church–Rosser languages [22,23], while the class of languages accepted by (nondeterministic) RWW-automata properly includes the class GCSL of growing context-sensitive languages [10]. In fact, RWW-automata already accept NP-complete languages [10].

In order to limit the expressive power of restarting automata, a monotonicity property was introduced for RLWW-automata which is based on the idea that from one cycle to the next in a computation, the actual place where a rewrite is performed must not increase its distance from the right end of the tape. Monotone restarting automata essentially model bottom-up one-pass parsers. Accordingly, it turned out that monotone RRWW- and RWW-automata characterize the class CFL of context-free languages, while monotone deterministic RRWW- and RWW-automata as well as several restricted versions thereof all characterize the class DCFL of deterministic context-free languages [8].

Also a generalization of the notion of monotonicity was introduced, which models the generalization from bottom-up one-pass parsers to bottom-up multi-pass parsers [27,28]. For an integer $j \geq 1$, a restarting automaton is called $j$-monotone if, for each of its computations, the corresponding sequence of cycles can be partitioned into at most $j$ subsequences such that each of these subsequences is monotone. It is shown in [28] that the expressive power of $j$-monotone (nondeterministic) restarting automata without auxiliary symbols increases with the size of the parameter $j$.

Here we extend the results of [27,28] in various ways. The aforementioned notion of monotonicity can be interpreted as that of right-monotonicity, and throughout this paper we will denote it as such in order to distinguish it clearly from
other notions of monotonicity. We introduce the symmetric notion of left-monotonicity for restarting automata, which is based on the idea that from one cycle to the next in a computation, the actual place where a rewrite takes place must not increase its distance from the left end of the tape. Also we consider the notion of right-left-monotonicity, which requires that a restarting automaton is simultaneously right- and left-monotone. Further, we generalize these notions to j-left- and j-right-left-monotonicity.

In the nondeterministic case each RLWW-automaton is equivalent to some RRWW-automaton, and this equivalence carries over to all the above notions of monotonicity. This, however, is not true in general in the deterministic case. We will see that, for all values of j and for all types of deterministic RRWW-automata, the class of languages that are accepted by j-right-monotone restarting automata of that type coincides with the class DCFL of deterministic context-free languages. On the other hand, the various types of j-right-monotone deterministic RLWW-automata are more expressive than the corresponding types of j-right-monotone deterministic RRWW-automata. In contrast to the situation for deterministic restarting automata that are right-monotone, we will see that for the left-monotone deterministic restarting automata, the RWW-model is as powerful as the RLWW-model. Further, at least for deterministic restarting automata without auxiliary symbols, the degree j of left-monotonicity yields an infinite hierarchy. Analogous hierarchies are obtained for the various types of nondeterministic restarting automata without auxiliary symbols.

The paper is structured as follows. After restating the main definitions and basic facts concerning restarting automata in the next section, we introduce a slight generalization of the basic model in Section 3. This model, called shrinking restarting automaton, is based on the notion of weight reduction, that is, it is required that each rewrite step reduces the overall weight of the actual tape content with respect to a fixed weight function. After discussing the expressive power of left-monotone (nondeterministic) restarting automata in Section 4, the shrinking model of the restarting automaton is then used in Section 5 to simplify the proof of our first main result stating that for left-monotone restarting automata, the deterministic RWW-model is as powerful as the deterministic RLWW-model (Theorem 5.6). In addition we associate a DCFL to every left-monotone deterministic RRWW-automaton M which, for each element w of the language L(M) accepted by M, contains an encoding of w together with some information on the computation of M on input w (Lemma 5.8). Then, in Section 6, we consider right-left-monotone restarting automata, and we will see that various types of right-left-monotone (deterministic) restarting automata characterize the class of (deterministic) linear languages.

In Section 7, we define the degree of (right-, left-, right-left-) monotonicity mentioned above, and we derive the aforementioned result that the various types of j-right-monotone deterministic restarting automata all characterize the class DCFL (Theorem 7.1). On the other hand we establish infinite hierarchies for all other types of restarting automata without auxiliary symbols and for all types of monotonicity. In the next section we compare the various classes at the same level of j-monotonicity to each other, and in Section 9 we separate the second from the first level of monotonicity for the various types of restarting automata with auxiliary symbols. Finally, in Section 10 we present an example language that is inherently non-monotone. The paper closes with a presentation of some open problems that are related to the notions and results presented here. In an appendix six diagrams can be found that summarize the hierarchy results obtained throughout the paper.

2. Definitions and notation

For an alphabet A, we denote by A+ the set of non-empty words over A, while A* denotes the set of all words over A including the empty word e. For i ∈ N, Ai := {x ∈ A* | |x| = i} and A≤i := {x ∈ A* | |x| ≤ i}, where |x| denotes the length of the word x. Further, for a word x ∈ A*, let x[i] denote the ith symbol of x, and let x[i, j] := x[i] · · · x[j] for i ≤ j. By xR we denote the reversal of x, that is, xR = x[|x|] · · · x[1]. For a language L, LR := {xR | x ∈ L}. Finally, N+ will denote the set of positive integers.

We start by restating in short the definitions of the various models of the restarting automaton that will be considered in this paper.

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, \eta, \$, q0, k, δ), where $Q$ is a finite set of states, $\Sigma$ is a finite input alphabet, $\Gamma$ is a finite tape alphabet containing Σ, the symbols $\$, $\$ $\notin \Gamma$ are used as markers for the left and right border of the work space, respectively, $q0 \in Q$ is the initial state, $k \geq 1$ is the size of the read/write window, and

$\delta : Q \times PC^{(k)} \rightarrow p((Q \times ((MVR, MVL) \cup PC^{\leq (k-1)})) \cup \{Restart, Accept\})$
is the transition relation. Here \( p(S) \) denotes the powerset of the set \( S \), \( \mathcal{P}C^{(k)} \) is the set of possible contents of the read/write window of \( M \), where

\[
\begin{align*}
\mathcal{P}C^{(0)} &= \{ \varepsilon \}, \\
\mathcal{P}C^{(1)} &= \Gamma \cup \{ \varepsilon, \$ \}, \\
\mathcal{P}C^{(i)} &= (\{ \varepsilon \} \cdot \Gamma^{i-1}) \cup \Gamma^i \cup (\Gamma^{i-1} \cdot \{ \varepsilon \}) \cup (\{ \varepsilon \} \cdot \Gamma^{i-2} \cdot \$) \quad (i \geq 2),
\end{align*}
\]

and \( \mathcal{P}C^{(k-1)} := \bigcup_{i=0}^{k-1} \mathcal{P}C^{(i)} \).

The transition relation describes five different types of transition steps:

1. **A move-right step** is of the form \((q', \text{MVR}) \in \delta(q, u)\), where \( q, q' \in Q \) and \( u \in \mathcal{P}C^{(k)}, u \neq \$ \). 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 shift to the right is possible.

2. **A move-left step** is of the form \((q', \text{MVL}) \in \delta(q, u)\), where \( q, q' \in Q \) and \( u \in \mathcal{P}C^{(k)} \) such that \( u \) does not begin with the symbol \( \varepsilon \). 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 \((q', \text{MLR}) \in \delta(q, u)\), where \( q, q' \in Q, u \in \mathcal{P}C^{(k)}, u \neq \$, \) and \( v \in \mathcal{P}C^{(k-1)} \) 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 by \(|u| - |v|\) cells, 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 \( \varepsilon \) 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.

4. **A restart step** is of the form \( \text{Restart} \in \delta(q, u) \), where \( q \in Q \) and \( u \in \mathcal{P}C^{(k)} \). 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 \( \varepsilon \), and to reenter the initial state \( q_0 \).

5. **An accept step** is of the form \( \text{Accept} \in \delta(q, u) \), where \( q \in Q \) \( u \in \mathcal{P}C^{(k)} \). It causes \( M \) to halt and accept.

If \( \delta(q, u) = \emptyset \) for some \( q \in Q \) \( u \in \mathcal{P}C^{(k)} \), then \( M \) necessarily halts, and we say that \( M \) rejects in this situation. Further, we require that within any computation, ignoring the MVR- and MVL-steps, Rewrite steps alternate with Restart steps, where a Rewrite step comes first. Finally, the letters in \( \Gamma \backslash \Sigma \) are called auxiliary symbols.

A configuration of \( M \) is a string \( x\alpha\beta \), where \( q \in Q \), and either \( x = \varepsilon \) and \( \alpha \in \{ \varepsilon \} \cdot \Gamma^* \cdot \{ \$ \} \) or \( x \in \{ \varepsilon \} \cdot \Gamma^* \) and \( \beta \in \Gamma^* \cdot \{ \$ \} \); here \( x \) represents the current state, \( \alpha \beta \) is the current content of the tape, and it is understood that the read/write window contains the first \( k \) symbols of \( \beta \) or all of \( \beta \) when \(|\beta| \leq k \). A restarting configuration is of the form \( q_0\varepsilon w\$, \) where \( w \in \Gamma^* \); if \( w \in \Sigma^* \), then \( q_0\varepsilon w\$ \) is an initial configuration. Thus, initial configurations are a special type of restarting configurations. By executing an Accept operation, \( M \) reaches an accepting configuration. These configurations will simply be denoted by the word Accept. By \( \vdash_M \) we denote the single-step computation relation that \( M \) induces on its set of configurations. The reflexive and transitive closure \( \vdash_M^* \) of \( \vdash_M \) is then the computation relation of \( M \).

In general, the automaton \( M \) is nondeterministic, that is, there can be two or more instructions with the same left-hand side \( (q, u) \), and thus, there can be more than one computation for an input word. If this is not the case, the automaton is deterministic.

A word \( w \in \Gamma^* \) is accepted by \( M \), if there is a computation which, starting with the restarting configuration \( q_0\varepsilon w\$, \) finishes by executing an Accept instruction. By \( L^w(M) \) we denote the language consisting of all words accepted by \( M \); we say that \( L^w(M) \) is the language of sentential forms that are accepted (recognized) by \( M \). The language \( L(M) := L^w(M) \cap \Sigma^* \) of all input words accepted by \( M \) is the language accepted (recognized) by \( M \).

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, MVL, and Rewrite operations 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 stated above we require that \( M \) performs exactly one Rewrite operation during any cycle—thus each new phase starts on a shorter word than the previous one. During a tail at most one Rewrite operation may be executed.

An RLWW-automaton could have infinite tail computations, executing MVR- and MVL-steps indefinitely. However, as we are only interested in accepting computations, we can disregard these infinite tail computations and restrict our attention to finite computations.
We use the notation \( u \overset{c}{\Rightarrow}_{M} v \) to denote a cycle of \( M \) beginning with the restarting configuration \( q_0qu\$ \) and ending with the restarting configuration \( q_0qv\$ \); the relation \( \overset{c}{\Rightarrow}_{M} \) is the reflexive and transitive closure of \( \Rightarrow_{M} \). Thus, \( \overset{c}{\Rightarrow}_{M} \) can be seen as the *single-step rewrite relation* induced by \( M \), and \( \overset{c}{\Rightarrow}_{M} \) is the corresponding *rewrite relation*. We will often make use of the following facts that are of central importance for restarting automata (see [7]), usually without mentioning it.

**Fact 2.1 (Error Preserving Property).** Let \( M \) be a restarting automaton, and let \( u, v \) be words over its tape alphabet. If \( u \overset{c}{\Rightarrow}_{M} v \) and \( u \notin L_W(M) \), then \( v \notin L_W(M) \).

**Fact 2.2 (Correctness Preserving Property).** Let \( M \) be a deterministic restarting automaton, and let \( u, v \) be words over its tape alphabet. If \( u \overset{c}{\Rightarrow}_{M} v \) and \( u \in L_W(M) \), then \( v \in L_W(M) \).

As a technical tool in proofs, the following “pumping of cycles” will be used repeatedly (see [27]). Given an RLWW-automaton \( M \) and a cycle \( u \overset{c}{\Rightarrow}_{M} v \) of \( M \), we say that a nonempty word \( z \) is a *pumping subword with respect to this cycle*, if \( u = u_1zu_2 \), \( v = v_1zv_2 \), and \( u_1z^i u_2 \overset{c}{\Rightarrow}_{M} v_1 z^i v_2 \) for all \( i \geq 0 \).

**Fact 2.3 (Pumping Lemma).** For any RLWW-automaton \( M \), there exists a constant \( p \) such that, for any cycle \( uuvw \overset{c}{\Rightarrow}_{M} uvw \) of \( M \), each subword of length \( p \) of \( u \) and of \( w \) contains a pumping subword (with respect to that cycle). Such a pumping subword can also be found in any factor of length \( p \) of a word accepted in a tail computation.

Now we define those subclasses of RLWW-automata that are relevant for our investigation. These subclasses are obtained by combining two types of restrictions:

(a) Restrictions on the rewrite instructions, which are expressed by the first part of the class name:
   - **RL-**: no restriction, that is, MVR- and MVL-steps are available,
   - **RR-**: a one-way automaton that does not use MVL-operations,
   - **R-**: an RR-automaton which restarts immediately after rewriting, that is, for an automaton of this type each Rewrite transition is immediately followed by a Restart transition.

(b) Restrictions on the Rewrite instructions, which are expressed by the second part of the class name:
   - **WW**: no restriction, that is, auxiliary symbols can be used in Rewrite instructions,
   - **W**: no auxiliary symbols are available, that is, for an automaton of this type the tape alphabet coincides with the input alphabet,
   - **E**: each Rewrite instruction simply deletes some symbols, that is, if \( (q', v) \in \delta(q, u) \), then \( v \) is obtained by deleting some symbols from \( u \).

For example, RRW-automata do not use MVL-instructions, and they do not have auxiliary symbols.

Obviously, for an RLW-automaton \( M \), the language \( L_W(M) \) of sentential forms accepted by \( M \) coincides with the language \( L(M) \). From a linguistic point of view, RLW-automata are an interesting class of automata. They can be used to describe the sets of sentential forms of natural languages. Roughly speaking, sets of sentential forms (well-formed strings of word-forms, lexical and syntactical categories) together with the sequences of reductions obtained from an RLW-automaton are a more meaningful description of a natural language than the set of well-formed sentences itself.

**Notation.** For brevity, the prefix *det-* will be used to denote the property of being deterministic. For any class \( A \) of automata, \( \mathcal{L}(A) \) will denote the class of languages that can be accepted by automata from that class.

A pushdown automaton (PDA) is said to be *one-turn* if it executes no push operation after the first pop operation has occurred in a computation. It is well-known that a language is *linear* if and only if it is accepted by a one-turn PDA (see, e.g., [1]). By LIN we denote the class of linear languages, and by DLIN we denote the class of languages that are accepted by deterministic one-turn PDAs.

By \( \subseteq \) we denote the proper subset relation. Further, we will sometimes use regular expressions instead of the corresponding regular languages.

Finally we come to the various notions of *monotonicity*. Each cycle \( C \) contains a unique configuration \( zq\beta \) in which a Rewrite instruction is applied. Then \( |\beta| \) is the *right distance* of \( C \), denoted by \( D_r(C) \), and \( |z| \) is the *left distance* of \( C \), denoted by \( D_l(C) \).

We say that a sequence of cycles \( S = (C_1, C_2, \ldots, C_n) \) is *right-monotone* if \( D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n) \), and we say that this sequence is *left-monotone* if \( D_l(C_1) \geq D_l(C_2) \geq \cdots \geq D_l(C_n) \). Finally, we call it *right-left-monotone* if it is simultaneously right- and left-monotone.
For each prefix $Y \in \{\text{right, left, right-left}\}$, a computation is $Y$-monotone if the corresponding sequence of cycles is $Y$-monotone. Observe that the tail of the computation does not play any role here. An RLWW-automaton $M$ is called $Y$-monotone if all its computations that start from an initial configuration are $Y$-monotone. The prefix $Y$-mon- will be used to denote the corresponding classes of restarting automata. Observe that right-monotonicity is the concept called monotonicity in [8].

We conclude this section with a result that relates the nondeterministic variants of the RLWW-automaton to the corresponding variants of the nondeterministic RRWW-automaton. It is a slight extension of a result from [27].

**Proposition 2.4.** Let $M_L$ be an RLX-automaton for some $X \in \{WW, W, \varepsilon\}$. Then there is an RRX-automaton $M_R$ such that $L(M_L) = L(M_R)$, $u \mapsto_M v$ if and only if $u \mapsto_{M_R} v$, and the right and the left distance of the cycle of $M_L$ coincides with the right and the left distance of the cycle of $M_R$, respectively.

This result immediately yields the following equivalences.

**Corollary 2.5.** For each $Y \in \{\text{right, left, right-left}\}$, and $X \in \{WW, W, \varepsilon\}$,

$$L(Y\text{-mon-RRX}) = L(Y\text{-mon-RLX}).$$

Hence, as far as nondeterministic restarting automata are concerned, the one-way RR-variant is just as powerful as the two-way RL-variant.

### 3. Shrinking restarting automata

Each restarting automaton is length-reducing in the sense that each application of a Rewrite transition shortens the length of the actual tape content. This means, of course, that a computation that starts with a tape content of length $n$ can consist of at most $n$ cycles. Here we introduce a slight generalization of this kind of restarting automaton in that we consider restarting automata that are shrinking with respect to a weight function.

A shrinking RLWW-automaton $M = (Q, \Sigma, \Gamma, q_0, \delta, k)$ has the same components as an RLWW-automaton with the exception that it is not required that $|v| < |u|$ holds for each of its Rewrite instructions $(q', v) \in \delta(q, u)$. Instead there must exist a weight function $\phi: \Gamma \cup \{q, \varepsilon\} \to \mathbb{N}_+$ such that, for each Rewrite step $(q', v) \in \delta(q, u)$, $\phi(u) > \phi(v)$ holds. Here $\phi$ is extended to a morphism $\phi: (\Gamma \cup \{q, \varepsilon\})^* \to \mathbb{N}$ by taking $\phi(\varepsilon) := 0$ and $\phi(wa) := \phi(w) + \phi(a)$ for all $w \in (\Gamma \cup \{q, \varepsilon\})^*$ and $a \in \Gamma \cup \{q, \varepsilon\}$. Observe that due to the fact that occurrences of the delimiters $q$ and $\varepsilon$ are neither created nor deleted during rewrite steps, the values $\phi(q)$ and $\phi(\varepsilon)$ are immaterial. Therefore, in what follows we will simply define weight functions as mappings from $\Gamma$ to $\mathbb{N}_+$.

It is easily seen that, for a shrinking RLWW-automaton $M$, there exists a constant $c$ such that each computation of $M$ that starts with a tape content of length $n$ can consist of at most $c \cdot n$ cycles. In particular, we see that the class of languages that are accepted by shrinking RLWW-automata is still contained in the complexity class NP and in the language class CSL.

We use the prefix $s$ to denote classes of shrinking restarting automata. For example, $sRWW$ denotes the class of shrinking RWW-automata, and $det-sRLW$ denotes the class of deterministic shrinking RLW-automata. Obviously, the length function $w \mapsto |w|$ is a particular weight function, and hence, each (standard) restarting automaton is shrinking with respect to the length function. Thus, we have the following inclusions.

**Proposition 3.1.** For each $X \in \{R, RR, RL, RW, RRW, RLW, RWW, RRWW, RLWW\}$,

(a) $L(det-X) \subseteq L(det-sX)$.
(b) $L(X) \subseteq L(sX)$.

Actually for the deterministic classes we obtain the following characterization, where CRL denotes the class of Church–Rosser languages [16,18].

**Theorem 3.2.**

$$CRL = L(det-RRW) = L(det-sRRW) = L(det-RRWW) = L(det-sRRWW).$$
Proof. In [23] the equalities \( CRL = \mathcal{L}(\text{det-RWW}) = \mathcal{L}(\text{det-RRWW}) \) are established. Thus, it remains to argue that the shrinking variant of the deterministic RRWW-automaton only accepts Church–Rosser languages.

It has been shown in [20] (see also [19,24]) that a language is Church–Rosser if and only if it is accepted by a shrinking deterministic two-pushdown automaton (sdTPDA). This characterization is used in [23] to prove that deterministic RRWW-automata only accept Church–Rosser languages by presenting a simulation of deterministic RRWW-automata by sdTPDAs. Now it is straightforward to verify that this simulation also works for deterministic shrinking RRWW-automata. It follows that \( \mathcal{L}(\text{det-sRRWW}) \subseteq CRL \), thus completing the proof of the theorem. \( \square \)

This characterization does not extend to the deterministic RLWW-automaton. In fact, one can easily design a deterministic RLWW-automaton for the language \( L_{\text{pal}} := \{ww^R \mid w \in \{a,b\}^*\} \) of palindromes of even length. As this language is not a Church–Rosser language [9], we obtain the following proper inclusion.

Corollary 3.3. \( \mathcal{L}(\text{det-RRWW}) \subsetneq \mathcal{L}(\text{det-RLWW}) \).

Actually, also the Gladkij language
\[
L_{\text{Gld}} := \{w \# w^R \# w \mid w \in \{a, b\}^*\},
\]
which is not even growing context-sensitive (see, e.g., [2]), is accepted by a deterministic RLWW-automaton. Just observe that the (nondeterministic) RRWW-automaton for \( L_{\text{Gld}} \) presented in [23] can easily be converted into a deterministic RLWW-automaton. It is, however, still open whether the deterministic sRLWW-automaton is more powerful than the standard, that is, length-reducing, deterministic RLWW-automaton. Also it is open whether each context-free language is accepted by a deterministic (shrinking) RLWW-automaton, that is, whether the class CFL of context-free languages is a proper subclass of the class \( \mathcal{L}(\text{det-(s)RLWW}) \), or whether these classes are incomparable under set inclusion.

4. Left-monotone restarting automata

In this section we are concerned with the expressive power of the left-monotone nondeterministic restarting automaton.

Let \( C := u \xrightarrow{\gamma}_M v \) be a cycle of a computation of an RLWW-automaton \( M \), and let \( x \rightarrow y \) be the Rewrite operation applied during that cycle, that is, \( quS = u_1xu_2 \) and \( \varphi vS = u_1yu_2 \) for some strings \( u_1 \) and \( u_2 \). Then the left distance of \( C \) is \( D_l(C) = |u_1| \), and its right distance is \( D_r(C) = |yu_2| \). Thus, the right distance always includes the size of the part of the tape inscription that is inside the read/write window of \( M \), while the left distance is just the distance from the left end marker \( \gamma \) to the left end of the read/write window. Based on this observation it is not hard to see that the following proposition holds.

Proposition 4.1. Let \( M_L \) be a (deterministic) RLWW-automaton with read/write window of size \( k \). Then there is a (deterministic) RLWW-automaton \( M_R \) such that \( L(M_L)^R = L(M_R) \), and \( u \xrightarrow{\gamma}_{M_L} v \) if and only if \( u^R \xrightarrow{\gamma}_{M_R} v^R \). In addition, if the right distance of the cycle of \( M_L \) is \( r \) and its left distance is \( l \), then the left distance of the corresponding cycle of \( M_R \) is \( \max(r-k, 0) \) and its right distance is \( \min(l+k, l+r) \). Hence, \( M_L \) is right-monotone if and only if \( M_R \) is left-monotone, and \( M_L \) is left-monotone if and only if \( M_R \) is right-monotone.

In [8] it is shown that \( \mathcal{L}(\text{right-mon-RWW}) \) and \( \mathcal{L}(\text{right-mon-RRWW}) \), and therewith by Corollary 2.5 also \( \mathcal{L}(\text{right-mon-RLWW}) \), coincide with the class CFL of context-free languages. As CFL is closed under reversal, Proposition 4.1 implies that left-monotone RWW- and RRWW-automata can only accept context-free languages. However, also the converse holds.

Lemma 4.2. For each context-free language \( L \), there exists a left-monotone RWW-automaton \( M \) such that \( L(M) = L \).

Proof. The idea of this proof is taken from [21]. Let \( L \subseteq \Sigma^* \) be a context-free language, and let \( G = (N, \Sigma, S, P) \) be a context-free grammar for \( L \). Without loss of generality we can assume that the productions of \( P \) have the following
Lemma 5.1. \[L \text{ left-monotone deterministic restarting automata by a pushdown automaton as described in [8], Theorem 3.2.}\]

for all \(\forall\) form, which is easily obtained from the Chomsky normal form:

\[S \rightarrow \varepsilon \quad \text{if } \varepsilon \in L,\]
\[S \rightarrow a \quad \text{for each } a \in \Sigma \cap L,\]
\[A \rightarrow BC \quad \text{where } A \in N \text{ and } B, C \in (N \setminus \{S\}) \cup \Sigma.\]

We define a restarting automaton \(M := (Q, \Sigma, \Gamma, \psi, q_0, \delta)\) as follows. Let \(Q := \{q_0, q_R\}\) and \(\Gamma := N \cup \Sigma \cup \{(A, B) \mid A, B \in N \cup \Sigma\}\). The transition relation \(\delta\) of \(M\) is now defined as follows, where \(A, B, C, D \in N \cup \Sigma\) and \(X, Z \in N\), and where \(\pi : \Gamma^* \rightarrow (N \cup \Sigma)^*\) denotes the morphism that is induced by \(\pi(A) := A\) and \(\pi((A, B)) := AB\) for all \(A, B \in N \cup \Sigma\):

\[
\delta(q_0, \psi S) := \text{Accept} \quad \text{for all } \gamma \in \Gamma \cup \{\varepsilon\} \text{ satisfying } S \Rightarrow_G^* \pi(\gamma),
\]
\[
\delta(q_0, AB) \ni (q_0, \text{MVR}) \quad \text{for all } A, B \in N \cup \Sigma,
\]
\[
\delta(q_0, ABC) \ni (q_0, \text{MVR}) \quad \text{for all } A, B, C \in N \cup \Sigma,
\]
\[
\delta(q_0, AB) \ni (q_R, (A, B)S) \quad \text{for all } A, B \in N \cup \Sigma,
\]
\[
\delta(q_0, AB) \ni (q_R, CS) \quad \text{if } (C \rightarrow AB) \in P,
\]
\[
\delta(q_0, AB(C, D)) \ni (q_R, (A, B)C, D) \quad \text{for all } A, B, C, D \in N \cup \Sigma,
\]
\[
\delta(q_0, AB(C, D)) \ni (q_R, X(C, D)) \quad \text{if } (X \rightarrow AB) \in P,
\]
\[
\delta(q_0, AB(C, D)) \ni (q_R, A(X, D)) \quad \text{if } (X \rightarrow BC) \in P,
\]
\[
\delta(q_0, AB(C, D)) \ni (q_R, AX) \quad \text{if } (X \rightarrow ZD), (Z \rightarrow BC) \in P \text{ for some } Z \in N,
\]
\[
\delta(q_0, \psi(A, B, C)) \ni (q_R, \psi(X, C)) \quad \text{if } (X \rightarrow AB) \in P,
\]
\[
\delta(q_0, \psi(A, B, C)) \ni (q_R, \psi X) \quad \text{if } (X \rightarrow ZC), (Z \rightarrow AB) \in P \text{ for some } Z \in N,
\]
\[
\delta(q_R, u) := \text{Restart} \quad \text{for each } u \in \mathcal{P}(3).
\]

Obviously \(M\) is an RWW-automaton. The encoding of two letters \(A, B \in N \cup \Sigma\) as a single letter \((A, B)\) is used to mark the position in the actual sentential form that has to be examined in the next step. Thus, the tape content always consists of a sequence of letters from \(N \cup \Sigma\) followed by a sequence of pairs, where any of these two subsequences can be empty. Since each rewrite step is performed at the border between these two subsequences, it is easily seen that \(M\) is left-monotone.

It remains to verify that \(M\) accepts the context-free language \(L\). It is easily seen that an accepting computation of \(M\) constructs a \(G\)-derivation for the given input in reverse order. Hence, we have \(L(M) \subseteq L\).

On the other hand, let \(w \in L\). Then there exists a leftmost \(G\)-derivation for \(w\). By induction on the number of derivation steps it can be shown that, for each \(x \in (N \cup \Sigma)^*\), if there is a leftmost derivation \(S \Rightarrow_G^* x\), then there exists an accepting computation of \(M\) that starts from the initial configuration \(q_0 \psi x S\). Actually, \(M\) reconstructs this leftmost derivation in reverse order. Thus, \(w \in L(M)\), that is, \(L = L(M)\). \(\square\)

As a consequence we obtain the following characterization.

**Theorem 4.3.**
\[
\text{CFL} = \mathcal{L}(\text{right-mon-R(R)WW}) = \mathcal{L}(\text{right-mon-RLWW}) = \mathcal{L}(\text{left-mon-R(R)WW}) = \mathcal{L}(\text{left-mon-RLWW}).
\]

Actually, it can be shown that this result extends to the shrinking restarting automaton [12]. In fact, the inclusion \(\mathcal{L}(\text{right-mon-sRRWW}) \subseteq \text{CFL}\) is obtained by a simple modification of the simulation of a right-monotone RRWW-automaton by a pushdown automaton as described in [8], Theorem 3.2.

5. Left-monotone deterministic restarting automata

Next we study the expressive power of the left-monotone deterministic restarting automaton. Our first result states that for deterministic shrinking restarting automata that are left-monotone, the RWW-model is as powerful as the RLWW-model.

**Lemma 5.1.** \(\mathcal{L}(\text{det-left-mon-sRLWW}) = \mathcal{L}(\text{det-left-mon-sRRWW})\).
Proof. Let $M = (Q, \Sigma, \Gamma, \varphi, s, q_0, k, \delta)$ be a deterministic sRLWW-automaton that is left-monotone and shrinking with respect to the weight function $\varphi$. We will construct a left-monotone deterministic sRWW-automaton $M'$ such that $M'$ accepts the same language as $M$. In fact, given an input $w \in \Sigma^*$, $M'$ will simulate the computation of $M$ on input $w$. For defining $M'$ we need to analyze the behaviour of $M$ in detail.

Each cycle of a computation of $M$ consists of three phases:

1. $M$ scans its tape by repeatedly performing MVR-steps and MVL-steps.
2. $M$ executes a Rewrite step, replacing a factor $u$ of the current tape content by a string $v$ satisfying $\varphi(v) < \varphi(u)$.
3. $M$ rescans its tape by repeatedly performing MVR-steps and MVL-steps until it eventually accepts, rejects, or restarts.

To simplify the following discussion we may assume without loss of generality that in phase (1) $M$ first scans its tape completely from left to right by performing a sequence of MVR-steps.

In contrast to the behaviour of $M$ described above, $M'$ simply scans its tape from left to right, performing a number of MVR-steps, until it decides to execute a Rewrite transition, thus ending the current cycle. Hence, if $M'$ is to simulate a cycle of $M$, then it needs to determine the information that $M$ collects during phases (1) and (3) before it can execute the simulation of the actual Rewrite step. This, however, $M'$ cannot do in general within a single cycle, as it does not see the content of the tape that is to the right of the actual position where the Rewrite step is to be performed. Thus, $M'$ will have to perform some preparatory cycles before it can actually execute the simulation of the Rewrite step of the current cycle of $M$.

Assume that the actual configuration of $M$ at the start of the current cycle is $q_0xwxyS$, where $x, u, y \in \Gamma^*$, $|u| = k$, and $u$ is the factor that $M$ is about to replace by the word $v$ in this cycle. In order to simulate this cycle of $M$, $M'$ will first encode information on the behaviour of $M$ on the suffix $y$ by performing a number of preparatory cycles that replace this suffix letter by letter from right to left by an encoding of $y$. This encoding replaces each letter $a$ of $y$ by a symbol that together with the letter $a$ encodes a crossing table describing the possible behaviour of $M$ at the tape square containing the letter $a$.

This encoding is defined as follows. Let the set of states of $M$ be $Q = \{q_0, q_1, \ldots, q_m\}$, and let

$$\Gamma_s := \{(a, (p_0, \ldots, p_m)) \mid a \in \Gamma, p_i \in Q \cup \{+, -, R, \uparrow\}, 0 \leq i \leq m\}.$$  

Then each letter of $y$ is replaced by a letter from $\Gamma_s$. The replacement of a letter $a$ is based on its position within $y$. Let $y = y_1y_2$. Then this particular occurrence of the letter $a$ is replaced by the symbol

$$\psi_{y_2}(a) := (a, (p_0, \ldots, p_m)) \in \Gamma_s,$$

where, for each $i = 0, 1, \ldots, m$, $p_i$ is chosen according to the following definition:

- $p_i := q_j$, if, starting from the configuration $q_iay_2S$, $M$ first makes a MVR-step, followed by a (possibly empty) sequence consisting entirely of MVR- and MVL-steps staying with its read/write window on the suffix $y_2S$, and eventually it performs a MVL-step taking $M$ to the configuration $q_jay_2S$;
- $p_i := +$, if, starting from the configuration $q_iay_2S$, $M$ either accepts immediately, or it makes a MVR-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the suffix $y_2S$, and accepts then;
- $p_i := -$, if, starting from the configuration $q_iay_2S$, $M$ either rejects immediately, or it makes a MVR-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the suffix $y_2S$, and rejects then;
- $p_i := R$, if, starting from the configuration $q_iay_2S$, $M$ either restarts immediately, or it makes a MVR-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the suffix $y_2S$, and restarts then;
- $p_i := \uparrow$, if, starting from the configuration $q_iay_2S$, $M$ either immediately performs a Rewrite step, or it makes a MVR-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the suffix $y_2S$, and makes a Rewrite step then;
- $p_i := q_j$, if, starting from the configuration $q_iay_2S$, $M$ immediately performs a MVL-step.

It is obvious that this encoding of the suffix $y$ can be computed by the deterministic sRWW-automaton $M'$ by simply replacing the letters of $y$ from right to left. In addition, it is clear that this computation is left-monotone, and that it is shrinking with respect to an appropriate weight function.
In the following we will use the notation $\psi_{gh}(g)$, where $g$, $h$ are strings, to denote the encoding of the string $g$ that we obtain according to the above definition from the suffix $gh$ of a tape content of $M$. Observe that, for all strings $f$, $g$, $h$, $\psi_{fgh}(fg) = \psi_{fgh}(f)\psi_{gh}(g)$ holds.

We define $M'$ as $M' := (Q', \Sigma, \Gamma', \varphi, s, q_0, 2k, \delta')$, where $\Gamma' := \Gamma \cup \Gamma_0$, and

$$Q' := \{(q, (s_0, \ldots, s_m)) \mid q \in Q, s_i \in Q \cup \{+, -, R, \uparrow\}, 0 \leq i \leq m\},$$

and we define the transition function $\delta'$ in such a way that a Rewrite step

$$\psi_{x1x2q1ubv} \vdash_M \psi_{x1x2qjvby} \quad (|x_2| = k - 1, |u| = k, b \in \Gamma)$$

of $M$ is simulated by $M'$ through a Rewrite step of the form

$$\psi_{x1q'x2\psi_{ubv}(u)\psi_{by}(v)\psi_{y}(y)} \vdash_{M'} \psi_{q_0x1x2\psi_{ubv}(v)\psi_{by}(b)\psi_{y}(y)}.$$

Observe that the actual simulation of the Rewrite step of $M$, which replaces the syllable $u$ by the string $v$, is performed by $M'$ on the border between the prefix of the tape content that is still unencoded, and the suffix of the tape content that has already been encoded. Further, we see that the read/write window of $M'$ contains one encoded symbol (the symbol $\psi_{by}(b)$ in (2)) to the right of the syllable $\psi_{ubv}(u)$ that is to be rewritten, and that it contains $k - 1$ still unencoded symbols to the left of that syllable (the string $x_2$ in (2)).

It is clear that from the transition function $\delta$ of $M$ and the information stored in $\psi_{by}(b)$, $M'$ can immediately determine the encoding $\psi_{ubv}(v)$. In addition, as $M$ is shrinking, so is $M'$, if the weight function is chosen appropriately.

As $M$ is left-monotone, the next cycle of $M$ has left distance at most $1 + |x_1x_2|$, and so the next Rewrite transition of $M'$ that simulates a Rewrite step of $M$ is to be performed on a prefix of $x_3x_2\psi_{ubv}(ub)v\psi_{y}(y)$, where $x_3$ is a suffix of $x_1$. Hence, $M'$ may first have to execute some cycles computing the encoding $\psi_{ubv}(ub)v\psi_{y}(y)$ of a suffix of $x_3$ of $x_1x_2$ before it can simulate the next Rewrite step of $M$. Thus, we see that with $M'$, also $M'$ is left-monotone.

It remains to define the MVR-steps of $M'$ and to describe in detail which state $q_i'$ will enable $M'$ to perform the above Rewrite step. Recall that we assume that in each cycle $M$ first scans its tape completely from left to right by performing a sequence of MVR-steps. The MVR-steps of $M'$ are now defined in such a way that they simulate the behaviour of $M$ on the prefix of the current tape content for the complete cycle of $M$. For that $M'$ stores a crossing table of $M$ in its state that corresponds to the possible behaviour of $M$ on the prefix to the left of the current position of the read/write window.

The initial state of $M'$ is chosen as $q_0' := (q_0, (q_0, \ldots, q_m))$. Next consider a configuration

$$\psi_{xqaybz1z2}$$

of $M$, where $q \in Q, a, b \in \Gamma, |y| = k - 2$, and $|z_1| \geq 1$, that is reached from the restarting configuration $q_0\psi_{xaybz1z2}$ by a sequence of MVR-steps. Assume that the restarting configuration of $M'$ that corresponds to the simulation of the actual cycle of $M$ is of the form $q_0'\psi_{xaybz1\psi_{z2}(z2)}$. Then starting from this configuration, $M'$ will reach the configuration

$$\psi_{xq'aybz1\psi_{z2}(z2)}$$

by performing MVR-steps. Here the state $q' \in Q'$ is defined as

$$q' := (q, (p_0, \ldots, p_m)),$$

where, for each $i = 0, 1, \ldots, m$, $p_i$ is determined according to the following definition:

$p_i := q_j$, if, starting from the configuration $\psi_{xq_iayb}$, $M$ first makes an MVL-step, followed by a (possibly empty) sequence consisting entirely of MVR- and MVL-steps staying with its read/write window on the prefix $\psi_{xay}$, and eventually it performs an MVR-step taking $M$ to the configuration $\psi_{xq_jayb}$;

$p_i := +$, if, starting from the configuration $\psi_{xq_iayb}$, $M$ either accepts immediately, or it makes an MVL-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the prefix $\psi_{xay}$, and accepts then;
\( p_i := - \), if, starting from the configuration \( qxq_i ayb \), \( M \) either rejects immediately, or it makes a MVL-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the prefix \( qxy \), and rejects then;

\( p_i := R \), if, starting from the configuration \( qxq_i ayb \), \( M \) either restarts immediately, or it makes a MVL-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the prefix \( qxy \), and restarts then;

\( p_i := \dagger \), if, starting from the configuration \( qxq_i ayb \), \( M \) immediately performs a Rewrite step, or it makes a MVL-step, followed by a (possibly empty) sequence of MVR- and MVL-steps where it stays with its read/write window on the prefix \( qxy \), and makes a Rewrite step then;

\( p_i := q_i \), if, starting from the configuration \( qxq_i ayb \), \( M \) immediately performs a MVR-step.

While scanning the tape from left to right, \( M' \) can certainly determine the correct states from \( Q' \) according to the above definition.

Now if \( M \) is to perform a Rewrite step in the current cycle that replaces a syllable of \( ayb_{12} \) that overlaps with a nonempty suffix \( y' \) of the factor \( ayb \) (see (3)), then \( M' \) needs to encode the syllable \( y' z_1 \) into \( \psi_{y' z_1} \) by performing \( |y' z_1| \) many preparatory cycles before it can simulate this Rewrite step. If, however, in the current cycle \( M \) will perform a Rewrite step somewhere on the suffix \( z_1 z_2 \), then in configuration (4), \( M' \) simply makes another MVR-step based on the above definition. Observe that in this situation the factor \( ayb \) of the current tape content does not overlap with the factor that is to be replaced in the current cycle, and hence, the crossing table computed from \( q' \) and \( ayb \) according to the above definition is not influenced by the result of this Rewrite step.

Finally, the Rewrite transition

\[ (q'_1, x_2 \psi_{xby}(u) \psi_{by}(b)) \rightarrow (q'_0, x_2 \psi_{xby}(v) \psi_{by}(b)) \]

in (2) is enabled for \( M' \), if the information stored in \( q'_1 = (q_1, (p_0, \ldots, p_m)) \), the information stored in \( \psi_{by}(b) \) and the strings \( x_2, u \) and \( v \) fit, that is, the crossing table stored in \( q'_1 \), which describes the behaviour of \( M \) on the prefix already read, the crossing table stored in the letter \( \psi_{by}(b) \), which describes the behaviour of \( M \) on the suffix to the right of that letter, and the remaining infix \( x_2 \) of length \( k - 1 \) together with the replacement \( v \) for the factor \( u \) describe a valid cycle of \( M \).

It follows that \( M' \) accepts the same language as \( M \), which completes the proof of the lemma.

Our second technical result shows that for left-monotone deterministic RWW-automata the standard (length-reducing) variant is as powerful as the shrinking variant.

**Lemma 5.2.** \( L(\text{det-left-mon-RWW}) = L(\text{det-left-mon-RWW}) \).

**Proof.** We will essentially follow a simulation technique presented in [24]. This method was initially used for two-pushdown automata, but because of the correspondence between CRL and \( L(\text{det-RWW}) \) (Theorem 3.2), it can be generalized to restarting automata. However, if one adjusts this simulation directly to RWW-automata, then the resulting automaton is not left-monotone, even if the automaton being simulated is. Thus, we must follow through the steps of the simulation from [24] and discuss the changes that are required in order to guarantee that the property of being left-monotone is preserved by the simulation.

We begin by remarking that the essential property that is necessary for adjusting the simulation technique from [24] to restarting automata is the fact that two consecutive Rewrite steps must “overlap,” which is the case for all left-monotone deterministic sRWW-automata.

Our presentation of the simulation will proceed in two stages. We begin by giving a high level description of the simulation of a left-monotone deterministic sRWW-automaton \( M = (Q, \Sigma, \Gamma, q_0, k, \delta) \) (with weight function \( \varphi \)) by a left-monotone deterministic RWW-automaton. This simulation consists of three major steps, which are outlined below. Afterwards we will discuss these steps in some detail.

**Step 1.** For the deterministic sRWW-automaton \( M \), we first construct a deterministic sRWW-automaton \( M' \) such that \( L(M') = L(M) \) and each Rewrite transition of \( M' \) reduces the weight of the actual tape content exactly by one (using a method from [24, Lemma 3.6]). Here the main difference to that lemma is the fact that we do not assign weights to state symbols. Instead of using intermediate states to reduce the weight by one in each step, we simply use new tape
symbols in order to perform these “intermediate” Rewrite steps. As these additional Rewrite steps have the same left distance as the Rewrite step of $M$ being simulated, it follows that $M'$ is left-monotone, if $M$ is. Thus, to simplify the presentation we can assume in the following that each Rewrite transition of $M$ reduces the weight exactly by one.

**Step 2.** Let $\#$ be a new symbol, and let $h : \Gamma^* \rightarrow (\Gamma \cup \{\#\})^*$ be the morphism that is induced by the mapping $h(a) := a^{\rho(a)-1}$ ($a \in \Gamma$). Thus, for each string $w \in \Gamma^*$, the string $h(w) \in (\Gamma \cup \{\#\})^*$ satisfies the condition $|h(w)| = \rho(w)$. We construct a (length-reducing) deterministic RWW-automaton $M_1$ that simulates the computation of $M$ on the tape content $\varphi(w)\$ ($w \in \Gamma^*$) step by step on the tape content $\varphi(h(w))\$. Hence, $L(M_1) = h(L(M))$. Further, if $M$ is left-monotone, then so is $M_1$: in a left-monotone computation, if $p_1, p_2, p_3, \ldots$ are the prefixes to the left of the actual rewrite positions in a sequence of consecutive cycles, then $p_{i+1}$ is necessarily a prefix of $p_i$ (and all of them are prefixes of the input word). Hence, $|h(p_{i+1})| \leq |h(p_i)|$, and so $h(p_{i+1})$ is a prefix of $h(p_i)$. Moreover, $M_1$ is length-reducing, as each Rewrite step of $M$ reduces the weight by one, and so each Rewrite step of $M_1$ reduces the length of the actual tape content by one.

**Step 3.** To complete the construction we would now like to simulate the automaton $M_1$ by an RWW-automaton that, instead of processing an input of the form $h(x)$ ($x \in \Sigma^*$), works directly with the original input $x$. However, it might be impossible to simulate the computation of $M_1$ on $h(x)$ in a length-reducing manner on the input $x$ itself, as already the length of $h(x)$ will in general be larger than the length of $x$. In order to overcome this problem, we follow a strategy from [24] (Proof of Lemma 4.2):

(a) First an automaton $M_2$ is used to replace the input string $x$ by a “compressed” version of $h(x)$. As compression ratio we take the number $2\mu$, where $\mu := \max_{a \in \Gamma} \{\rho(a)\}$.

(b) Then an automaton $M_3$ is used, that in each cycle, simulates $2\mu$ cycles of $M_1$.

The compression of ratio $2\mu$ guarantees that the compressed version $x_c$ of $h(x)$ satisfies $|x_c| \leq |x|$, and by simulating $2\mu$ cycles of $M_1$ in a single cycle of $M_3$ we guarantee that the length of the actual tape content of $M_3$ is reduced by exactly one per cycle. Thus, we see that the composition of $M_2$ and $M_3$ is length-reducing. Further, $M_2$ can clearly be realized in a left-monotone manner, and also $M_3$ is left-monotone, as $M_1$ is.

However, the composition of $M_2$ and $M_3$ is clearly not left-monotone, as the “compression phase” realized by $M_2$, in which the tape content is completely rewritten, precedes the “real” simulation. In order to make this part of the simulation left-monotone, we skip the compression phase. Instead we adopt the strategy of “lazy” compression, that is, we start on the uncompressed input and use the compression in combination with the simulation of Rewrite steps. In fact, we will make sure that the tape content always consists of an uncompressed prefix followed by a compressed suffix, and that each Rewrite step just replaces a prefix of the compressed part.

Next we describe the above schema in more detail. We will concentrate on the last step, because the first two steps are similar to the corresponding steps in [24]. We have the following situation:

1. $M := (Q, \Sigma, \Gamma, \varphi, S, q_0, k, \delta)$ is a left-monotone deterministic sRWW-automaton that is shrinking with respect to the weight function $\varphi$. We assume that each Rewrite transition of $M$ reduces the weight exactly by one.

2. $M_1 := (Q_1, \Sigma_1, \Gamma_1, \varphi, S, q_0^{(1)}, k_1, \delta_1)$ is a left-monotone deterministic RWW-automaton such that $L(M_1) = h(L(M))$. Here $\Sigma_1 := \Sigma \cup \{\#\}$, $\Gamma_1 := \Gamma \cup \{\#\}$, and each Rewrite transition of $M_1$ reduces the length of the tape content exactly by one.

3. $M_3 := (Q_3, \Sigma_3, \Gamma_3, \varphi, S, q_0^{(3)}, k_3, \delta_3)$ is an RWW-automaton such that $L(M_3) = \text{compress}_\varphi(h(L(M)))$.

Here $\Sigma_3 := \{\sigma_x | x \in (\Sigma_1)^{\leq z} \setminus \{e\}\}$, $\Gamma_3 := \{\sigma_x | x \in (\Gamma_1)^{\leq z} \setminus \{e\}\}$, and $z := 2\mu$. Further, $\text{compress}_\varphi : \Gamma_1^* \rightarrow \Gamma_3^*$ is defined through

$$\text{compress}_\varphi(x) := \sigma_{x_1} \cdots \sigma_{x_t},$$

if $x = x_1 x_2 \cdots x_t \in \Gamma_1^*$, $|x_i| = z$ for $1 \leq i \leq t - 1$, and $0 < |x_t| < z$. Further, $M_3$ is

- deterministic, as $M$ and $M_1$ are deterministic;
- length-reducing, as $z$ Rewrite steps of $M$ reduce the weight of its tape content by $z$, which implies that $z$ Rewrite steps of $M_1$ shorten its tape content by $z$ cells, which in turn implies that each Rewrite step of $M_3$, simulating $z$ Rewrite steps of $M_1$, shortens its tape content exactly by one;
- left-monotone, since we can ensure that $M_3$ performs a Rewrite step only when the position of the middle of its read/write window corresponds to the position of the read/write window of $M_1$ during the first Rewrite step of
the sequence of a Rewrite steps of $M_1$ that $M_3$ simulates in this step. Then the monotonicity of the sequence of left-distances of consecutive Rewrite steps of $M_3$ follows from the left-monotonicity of $M_1$.

Finally, we describe an RWW-automaton $M_4 := (Q_4, \Sigma_4, \Gamma_4, \varphi, S, q_0(4), k_4, \delta_4)$, which, given a word $x \in \Sigma^*$ as input, simulates the computation of $M_3$ on the input compress$_3(h(x))$. In this way $M_4$ will accept the language $L(M)$, that is, $L(M_4) = L(M)$.

Essentially $M_4$ will simulate each cycle of a computation of $M_3$ by a sequence of cycles. Assume that starting from the initial configuration $q_0(3) \varphi \sigma_{x_1} \sigma_{x_2} \ldots \sigma_{x_r} S$, where $\sigma_{x_1} \sigma_{x_2} \ldots \sigma_{x_r} = \text{compress}_3(h(w))$ for some input $w \in \Sigma^m$, $M_3$ will first make $j$ MVR-steps and then rewrite the factor $\sigma_{x_j} \sigma_{x_{j+1}} \ldots \sigma_{x_{j+k_3-1}}$ by $\sigma_{y_1} \sigma_{y_2} \ldots \sigma_{y_{k_3-1}}$, which leads to the restarting configuration

$$q_0(3) \varphi \sigma_{x_1} \ldots \sigma_{x_{j-1}} \sigma_{y_1} \ldots \sigma_{y_{k_3-1}} \sigma_{x_{j+k_3}} \ldots \sigma_{x_r} S.$$ 

Then, starting from the corresponding initial configuration $q_0(4) \varphi w S$, $M_4$ will first rewrite the appropriate suffix $w[r+1, m]$ of $w$ into $\sigma_{x_j} \sigma_{x_{j+1}} \ldots \sigma_{x_r}$, and then simulate the above Rewrite step of $M_3$, which yields the restarting configuration

$$q_0(4) \varphi w[1, r] \sigma_{y_1} \ldots \sigma_{y_{k_3-1}} \sigma_{x_{j+k_3}} \ldots \sigma_{x_r} S.$$ 

The next Rewrite step of $M_3$ will be performed at the same position or on a position further to the left, as $M_3$ is left-monotone. Thus, for simulating the next cycle of $M_3$, the automaton $M_4$ may first have to perform some more compression steps, before it can actually simulate the Rewrite step of $M_3$. This is similar to the situation in the proof of Lemma 5.1. The main difference is that here, due to choice of the compression ratio $\alpha = 2\mu$, the Rewrite transitions that perform compressions are necessarily length-reducing. Hence, it suffices to concentrate on those Rewrite transitions of $M_4$ that actually simulate Rewrite transitions of $M_3$. Observe that the tape content of $M_4$ will always be a string of the form $\varphi uyS$, where $u \in \Sigma^*$ and $y \in \Gamma_4^*$, and that the simulation of a Rewrite transition of $M_3$ is always performed on a prefix of the syllable $y$s. Thus, in the following discussion we will ignore those parts of the transition function of $M_4$ that deal with the compression steps.

Let $Q_4 := Q_3 \times \{0, 1, \ldots, x-1\} \times \{\text{ON}, \text{OFF}\}$, $q_0(4) := (q_0(3), 0, \text{ON})$, $\Sigma_4 := \Sigma$, $\Gamma_4 := \Sigma \cup \Gamma_3$, and $k_4 := x \cdot k_3$. The components of a state $(q, \gamma, s)$ of $M_4$ are interpreted as follows: $q$ is the state in which $M_3$ would be in the “corresponding configuration,” $\gamma$ is the “offset” of the position of the read/write window of $M_4$ in comparison to the position of the read/write window of $M_3$ (which is required to make up for the fact that $M_4$ works on a tape content that is only partially compressed), and $s$ indicates whether the current position of the read/write window of $M_4$ already corresponds to the next position of the read/write window of $M_3$ ($s = \text{ON}$) or whether $M_4$ is currently in the process of moving its read/write window in that position ($s = \text{OFF}$).

For describing the transition function of $M_4$ in some detail, we need some additional notions:

(a) Let $h' : (\Sigma \cup \Gamma_3)^* \rightarrow \Gamma_4^*$ be the morphism that is defined by $h'(a) := h(a)$ for $a \in \Sigma$, and $h'(\sigma_x) = x$ for $\sigma_x \in \Gamma_3$.

(b) Let $x = a_1 \#_1 a_2 \#_2 \ldots \#_{l-1} a_l \#_l$, where $a_i \in \Gamma$ and $t_i \geq 0$, $1 \leq i \leq l$. Then clean$(x) := h(a_1)h(a_2) \ldots h(a_l)$, that is,

$$\text{clean}(a_1 \#_1 a_2 \#_2 \ldots a_l \#_l) = a_1 \#_1^{\varphi(a_1)-1}a_2 \#_2^{\varphi(a_2)-1} \ldots a_l \#_l^{\varphi(a_l)-1}.$$

(c) For a string $x \in (\Sigma \cup \Gamma_3)^*$ and a non-negative integer $\gamma < x$, we take

$$h_4(x, \gamma) := \text{compress}_x(\#_1, \gamma \text{ clean}(h'(x))),$$

where $\beta := \gamma$, if $x[1] \in \Sigma$, and $\beta := 0$, otherwise.

Example 5.3 (Data representation). Let $\Sigma := \{a, b, c\}$, $\varphi(a) := 3$, $\varphi(b) := 2$, $\varphi(c) := 1$. Then $\mu = 3$, $x = 6$, $h(a) = a\#$, $h(b) = b\#$, and $h(c) = c$. Further, let $x := aabbcabca$. Then

$$h(x) = h'(x) = a\#b\#c\#b\#b\#a\#b\#c\#,$$

and

$$h_4(x, 0) = \text{compress}_x(\text{clean}(h'(x))) = \sigma_{x_1} \sigma_{x_2} \sigma_{x_3} \sigma_{x_4},$$
where $x_1 := \texttt{a##a##}$, $x_2 := \texttt{b##b##ca}$, $x_3 := \texttt{##b##ca}$, and $x_4 := \texttt{##}$. Note that here the value of the function clean is equal to the identity, because it is applied to the homomorphic image $h(x)$ of the string $x$ from $\Sigma^*$, not to a “mixed” string.

**Example 5.4 (Why the clean operation is needed).** We use the alphabet and the weights from the previous example. Let the tape of $M_4$ contain a “mixed” representation $x := b\texttt{bcac} \texttt{##b##ca} \texttt{##} \ldots$ of the content of the tape of $M_3$. Then $h'(x) = b\texttt{b##ca} \texttt{##b##ca} \texttt{##} \ldots$. Thus, four occurrences of the symbol # appear to the right of the first occurrence of $a$. This, however, is not a correct encoding of a string from $\Sigma^*$, as the weight of $a$ is 3, not 5. This situation occurs because in $x$, $a$ appears in uncompressed form, while the suffix of $x$ following $a$ is already in compressed form. This shows why (and when) the clean operation is needed.

The above function $h_4$ is used in the compression phases of $M_4$, which we will not discuss any further. We want to point out, however, that an important property of our encoding is the fact that a factor of the string stored on the tape of $M_4$ is never shorter than the corresponding representation of that factor on the tape of $M_3$, because of the choice of the compression ratio $2\mu$.

Finally we describe the transition function of $M_4$. Let $w \in \Gamma_4^\leq k_4$ be the current content of the read/write window of $M_4$, and let $(q, \gamma, s)$ be the current state.

If $s = \text{ON}$, then $M_4$ interprets the content of its read/write window in order to deduce the content of the read/write window of $M_3$ in the corresponding configuration. Let $w = uy$, where $u$ is the shortest prefix of $w$ satisfying $|h_4(u, \gamma)| \geq k_3$, that is, $u$ is the prefix of $w$ that corresponds to the current content of the read/write window of $M_3$—note that the read/write window of $M_4$ is sufficiently large to tackle possibly “uncompressed” parts of the tape content. Now the transition $\delta_4((q, \gamma, \text{ON}), w)$ is determined on the basis of $\delta_3(q, x)$, where $x$ is the prefix of length $k_3$ of $h_4(u, \gamma)$:

1. If $\delta_3(q, x) = \text{Accept}$, then $\delta_4((q, \gamma, \text{ON}), w) := \text{Accept}$.
2. If $\delta_3(q, x) = (q', \text{MVR})$, then $\delta_4((q, \gamma, \text{ON}), w) := ((q', \gamma', s'), \text{MVR})$, where

   $$\gamma' := (\gamma + \varphi(w[1])) \mod (\alpha)\quad \text{and} \quad s' := \text{ON iff } \gamma + \varphi(w[1]) \geq \alpha.$$

Our simulation ensures that in this case $w[1] \in \Sigma$ if $M_3$ is left-monotone, as $M_4$ rewrites only that part of the tape content that would be “rewritten” by $M_3$; and $M_3$ does not move its read/write window further to the right as in previous cycles, if it is left-monotone.

3. If $\delta_3(q, x) = (q', v)$, that is, $M_3$ makes a Rewrite, which is then immediately followed by a Restart, and $w \not\in \Gamma_3^\leq$, that is, $w$ is not yet completely compressed, then $M_4$ performs further compression steps. Otherwise, if $w$ is already in compressed form, then $w = xy$, and accordingly we take $\delta_4((q, \gamma, \text{ON}), w) := (q'', wy)$, which leads to a Restart in the next step.

If $s = \text{OFF}$, then the current position of the read/write window of $M_4$ does not “fit” any position of the read/write window of $M_3$, and so $M_4$ needs to perform a MVR-transition without changing the state of $M_3$ that is stored in its finite control. Actually, the state $q$ of $M_3$ stored in $M_4$’s current state is the state that $M_3$ entered after processing a factor that starts at some symbol to the left of the actual position of $M_4$’s read/write window. Thus,

$$\delta_4((q, \gamma, \text{OFF}), w) := ((q, \gamma', s'), \text{MVR}),$$

where $\gamma' := (\gamma + \varphi(w[1])) \mod (\alpha)$, and $s' := \text{ON iff } \gamma + \varphi(w[1]) \geq \alpha$.

The example below illustrates the behaviour of $M_4$.

**Example 5.5 (State transitions of $M_4$).** We use the alphabet and the weights from the previous examples, and assume that the current tape content of $M_4$ is $q_0xS$, where $x := \texttt{aabbcabca}$. Then the corresponding tape content of $M_3$ is $q_0x_1x_2x_3x_4x_5S$ (see Example 5.3). We assume that $k_3 = 2$, and that the transition function $\delta_3$ of $M_3$ contains the following steps among others:

$$\delta_3(q_0^{(3)}, q_0x_1) = (q_0^{(3)}, \text{MVR}),$$
$$\delta_3(q_0^{(3)}, x_1x_2) = (q', \text{MVR}),$$
$$\delta_3(q', x_2x_3) = (q'', \text{MVR}).$$
Then $k_4 = 12$, and the corresponding computation of $M_4$ on $x$ proceeds as follows, where in each configuration we underline the first letter that $M_4$ sees in its read/write window:

$$(q_0^{(3)}, 0, \text{ON}), \underline{\varnothing}aabb\underline{c}abca$$_\underline{S} \vdash_{M_4} ((q_0^{(3)}, 0, \text{ON}), \underline{\varnothing}aabb\underline{c}abcaS) \vdash_{M_4}

$$(q', 3, \text{OFF}), \underline{\varnothing}aabb\underline{c}abca$$_\underline{S} \vdash_{M_4} ((q', 0, \text{ON}), \underline{\varnothing}aabb\underline{c}abcaS) \vdash_{M_4}

$$(q'', 2, \text{OFF}), \underline{\varnothing}aabb\underline{c}abca$$_\underline{S} \vdash_{M_4} ((q'', 4, \text{OFF}), \underline{\varnothing}aabb\underline{c}abcaS) \vdash_{M_4}

$$(q'', 5, \text{OFF}), \underline{\varnothing}aabb\underline{c}abca$$_\underline{S} \vdash_{M_4} ((q'', 2, \text{ON}), \underline{\varnothing}aabb\underline{c}abcaS) \vdash_{M_4} \ldots$$

Finally we argue that the automaton $M_4$ is indeed length-reducing and left-monotone. First, recall that $M_3$, which accepts the compressed version of $h(L(M))$, is length-reducing and left-monotone. The deterministic RWW-automaton $M_4$ simulates the automaton $M_3$, however, it works on a “mixture” of the compressed version of the homomorphic image $h(w)$ and the original input $w$. But despite this fact, $M_4$ is length-reducing and left-monotone:

(a) Note first that each compression step is length-reducing, as each symbol from $I_3$ created by the process of compressing the input of $M_4$ contains the encoding of the homomorphic images of at least two symbols from $\Sigma$. Further, each Rewrite transition $xy \rightarrow vy$ of $M_4$ corresponds to a Rewrite transition $x \rightarrow v$ of $M_3$. Hence, with $M_3$ also $M_4$ is length-reducing.

(b) $M_4$ performs compression steps from right to left until this process reaches the position where a Rewrite step of $M_3$ is to be simulated. Hence, left-monotonicity of $M_3$ follows from the fact that $M_3$ is left-monotone. □

From Lemmas 5.1 and 5.2 we immediately obtain the following result.

**Theorem 5.6.**

$$\mathcal{L}(\text{det-left-mon-sRLWW}) = \mathcal{L}(\text{det-left-mon-sRRWW})$$

$$= \mathcal{L}(\text{det-left-mon-sRWW}) = \mathcal{L}(\text{det-left-mon-RLWW})$$

$$= \mathcal{L}(\text{det-left-mon-RRWW}) = \mathcal{L}(\text{det-left-mon-RWW}).$$

It remains to derive a characterization of the class $\mathcal{L}(\text{det-left-mon-RWW})$ in terms of other language classes. As $\mathcal{L}(\text{det-right-mon-RLWW})$ properly contains the class of deterministic context-free languages [27], it follows that the class $\mathcal{L}(\text{det-left-mon-RWW})$ properly contains the reversals of all deterministic context-free languages, but it still remains to classify the additional expressive power of left-monotone deterministic RWW-automata.

For future reference we present the following technical result relating left-monotone deterministic RWW-automata to certain right-monotone deterministic RWW-automata. Recall that for an RWW-automaton we can assume without loss of generality that in each cycle it moves its read/write window all the way to the right end of the tape before it restarts, accepts or rejects, respectively.

**Definition 5.7.** Let $M = (Q, \Sigma, I, a, S, q_0, k, \delta)$ be a deterministic RWW-automaton. For $w \in \Sigma^+$, let $w = w_1u_w_2 \vdash_{M} w_1uvw_2$ be the first cycle of $M$ on input $w$, where $u \rightarrow v$ is the Rewrite step applied during this cycle, and $|u| = k$. Then, for $i = 1, \ldots, |w|$, let $q_i \in Q$ denote the state of $M$ during the above cycle when the leftmost position of the read/write window contains the symbol

$$-w[i], \text{ if } 1 \leq i \leq |w_1| \text{ or } |w_1| + k + 1 \leq i \leq |w|, \text{ or}$$

$$-u[1], \text{ if } |w_1| + 1 \leq i \leq |w_1| + k.$$ 

Here the second alternative corresponds to the fact that after performing the Rewrite transition $u \rightarrow v$, the read/write window of $M$ jumps from position $|w_1| + 1$ to position $|w_1| + |v| + 1$. The string $q_1q_2 \cdots q_{|w|}$ will be called the trace of $M$ on input $w$.

Further, the string $x \in (\Sigma \times Q)^{|w|}$, defined by $x[i] := \langle w[i], q_i \rangle$ is denoted as $\text{reach}_M(w)$ of $w$. For a string $y \in (\Sigma \times Q)^*$, $\text{src}(y)$ (source) denotes the projection onto the $\Sigma$-component, and $\text{trc}(y)$ (trace) denotes the projection onto the $Q$-component. Finally, by $\overline{L}_M$ we denote the language

$$\overline{L}_M := \{y \in (\Sigma \times Q)^* \mid y = \text{reach}_M(w) \text{ for some } w \in L(M)\}.$$
Thus, $\bar{L}_M$ contains the elements $w$ of $L(M)$ together with a description of the states encountered by $M$ during the first cycle of its computation on input $w$.

**Lemma 5.8.** For each left-monotone deterministic RRWW-automaton $M$, $\bar{L}_M^R \in DCFL$.

**Proof.** Let $M = (Q, \Sigma, \Gamma, \delta, s_0, MVR)$ be a deterministic RRWW-automaton that is left-monotone. As $L$ = $L$ (deterministic RRWW), it suffices to construct a right-monotone deterministic RRWW-automaton $M'$ with input alphabet $\Sigma \times Q$ for the language $\bar{L}_M^R$. This automaton will simulate $M$ cycle by cycle.

Recall that each cycle of a computation of $M$ consists of three phases:

1. a MVR-phase, in which $M$ scans a prefix of the current tape content from left to right using MVR-instructions only;
2. a rewrite-phase, in which $M$ applies a Rewrite step;
3. another MVR-phase, in which $M$ scans the suffix of the tape content to the right of the position where the Rewrite step took place. This phase ends with either a Restart step, or an Accept instruction, or rejection when the read/write window contains only the right delimiter $\$.

Let $C_1, C_2, \ldots, C_m$ be the sequence of cycles of a computation of $M$. As $M$ is left-monotone, we have $D_t(C_1) \geq D_t(C_2) \geq \cdots \geq D_t(C_m)$. On the other hand, as $M$ is deterministic, we have $D_t(C_{i+1}) > D_t(C_i) - k$, as $M$ cannot make a Rewrite step in cycle $C_{i+1}$ while its read/write window is still completely on the prefix of the current tape content that was scanned in phase (1) of the previous cycle, that is,

$$D_t(C_i) - k + 1 \leq D_t(C_{i+1}) \leq D_t(C_i) \quad (1 \leq i < m).$$

Now we describe the behaviour of the right-monotone deterministic RRWW-automaton $M'$. For an input $y \in (\Sigma \times Q)^*$, it simulates the computation of $M$ for the input $w := \text{src}(y)^R$. Each cycle of a computation of $M'$ consists of three phases corresponding to the phases of $M$'s cycle.

1. $M'$ first performs a number of MVR-steps, scanning a prefix of $y$. This corresponds to phase (3) of the corresponding cycle of $M$'s computation on input $w$, and it continues until the left-hand side of a Rewrite step of $M$ is found, and the state stored with the rightmost of these letters indicates that $M$ would now execute a Rewrite step. While making these MVR-steps $M'$ executes the following computations:
   
   (a) As long as the symbols read are input symbols, it checks that the states stored in the second components describe the reverse of a valid sequence of state transitions of $M$ that is compatible with the letters from $\Sigma$ stored in the first components and that correspond to a third phase of a cycle of $M$, that is, the states from $Q$ encountered during this part of the computation belong to the subset of states of $M$ that are used by $M$ only after having performed a Rewrite step. This checking is abandoned if and when $M'$ encounters a symbol that indicates by a special mark that a Rewrite has been performed before (see (2) below).
   
   (b) For the current position of its read/write window, $M'$ determines the following two sets of states of $M$, where $y_1$ is the prefix that has been scanned so far (including the leftmost symbol of the current content of the read/write window), and $w_1 := \text{src}(y_1)^R$ (see, e.g., [23]):

$$Q_+(w_1) := \{ q \in Q \mid qw_1s \vdash_M^{MVR} \text{Accept} \},$$

$$Q_-(w_1) := \{ q \in Q \mid qw_1s \vdash_M^{MVR} \text{Restart} \},$$

where $\vdash_M^{MVR}$ denotes a finite sequence of MVR-steps of $M$.

2. For the size of the read/write window of $M'$, we choose the number $k' := 2k + 1$. The automaton $M'$ uses the part from positions 2 to $k + 1$ of its read/write window to simulate the read/write window of $M$. The part from positions $k + 2$ to $2k + 1$ serves as a kind of look-ahead, in which $M'$ sees those symbols (in reverse) that $M$ saw in the previous $k$ positions (see the diagram in Fig. 1). When $M'$ contains (the reversal of) the leftmost side of the actual Rewrite step of $M$ in positions 2 to $k + 1$ of its read/write window, then it can determine from the state stored with the rightmost letter in its read/write window (that is, the state corresponding to the first letter of the syllable $w_4$ in the diagram above) the actual state of $M$, and it can verify that $M$ would now perform a Rewrite step. Observe that due to the fact that $M$ is left-monotone, the prefix $w_3w_4$ to the left of the current position of the read/write window
of \( M \) has not yet been rewritten in any way, that is, these letters are still from the original input. Now \( M' \) simulates the Rewrite step of \( M \), using a special auxiliary symbol for replacing the last letter of \( w_1^R \) to indicate for future cycles (see (1)) that a Rewrite has been performed. The first position of the read/write window of \( M' \) still contains the first letter of the suffix \( w_1 \) of the tape content of \( M \). Hence, from the state that \( M \) enters after performing the Rewrite step and the sets \( Q_+(w_1) \) and \( Q_{ns}(w_1) \) associated with this particular letter (see (1)), \( M' \) can determine the outcome of phase (3) of the current cycle of \( M \). Accordingly, \( M' \) now knows whether \( M \) will restart, accept or reject.

(3) Finally \( M' \) scans the remaining suffix of its tape content. As long as the symbols read are input symbols, it checks that the states stored in the second components describe the reverse of a valid sequence of state transitions of \( M \) that is compatible with the letters from \( \Sigma \) stored in the first components and that correspond to a first phase of a cycle of \( M \), that is, the states from \( Q \) encountered during this part of the computation belong to the subset of states of \( M \) that are used by \( M \) only before having performed a Rewrite step. In particular, it checks that the state stored with the last symbol is the state reached by \( \delta(q_0, \text{src}(y_2))^R \) and \( y_2 \) is the suffix of length \( k - 1 \) of \( w_3^R \). If all these tests are successful, then \( M' \) restarts or accepts, if \( M \) would restart or accept, respectively. Otherwise, \( M' \) rejects.

In each cycle \( M' \) can verify that it has found the correct position of the Rewrite step by extracting the information on the behaviour of \( M \) during its first phase of the corresponding cycle from the trc-part (that is, the second component) of the current suffix of the tape inscription. This information is available in each cycle, as \( M \) is left-monotone. It follows that \( M' \) does indeed accept the language \( \mathcal{L}_{M}^R \), and that \( M' \) is right-monotone. \( \square \)

6. Right-left-monotone restarting automata

Next we consider right-left-monotone restarting automata. For the nondeterministic variant we have the following characterization.

**Theorem 6.1.**

\[ \mathcal{L}(\text{right-left-mon-RWW}) = \mathcal{L}(\text{right-left-mon-RRWW}) = \mathcal{L}(\text{right-left-mon-RLWW}) \]

**Proof.** A language is linear if and only if it is accepted by a one-turn PDA that during the first phase of each of its computations simply shifts input symbols onto its pushdown store (see, e.g., [1]). Now it is easily seen that a one-turn PDA of this type can be simulated by an RWW-automaton that is right-left-monotone.

Conversely, assume that \( M \) is a right-left-monotone RLWW-automaton. By Proposition 2.4 there is an equivalent RWW-automaton \( M' \) that is also right-left-monotone. In [8] it is shown how to simulate the right-monotone RWW-automaton \( M' \) by a pushdown automaton \( P \). As \( M' \) is not only right-monotone, but also left-monotone, the pushdown automaton \( P \) can easily be modified to be one-turn. This completes the proof. \( \square \)
Next we turn to the deterministic case. It is shown in [6] that the class DLIN of deterministic linear languages coincides with the class of languages that are generated by linear grammars that satisfy an additional LR(1) condition. Based on this characterization we obtain the following result.

**Theorem 6.2.** For each $X \in \{R, RR, RW, RRW, RWW, RRWW\}$,

$$DLIN = \mathcal{L}(\text{det-right-left-mon-X})\text{.}$$

**Proof.** The relation $DLIN \supseteq \mathcal{L}(\text{det-right-left-mon-RRWW})$ follows from the construction used for proving that $\mathcal{L}(\text{det-right-mon-RRWW}) \subseteq \text{DCFL}$ in [8] (Theorem 3.2). In the paper it is shown how to construct a deterministic PDA accepting the same language as a given right-monotone deterministic RRWW-automaton. Inspecting the construction, we find that in the case of a det-right-mon-RRWW-automaton $M$ the constructed PDA is a deterministic one-turn PDA. Hence, the language $L(M)$ is in DLIN.

The inclusion $DLIN \subseteq \mathcal{L}(\text{det-right-left-mon-R})$ can be shown using the construction from the proof of Lemma 3.3 in [8]. There it is shown that for any language $L$ from DCFL, a det-right-mon-R-automaton $M$ accepting $L$ can be constructed. The proof is based on the characterization of the deterministic context-free languages by LR(1) grammars. Hence, it can be applied also to languages from DLIN. Looking at the construction it is evident that the constructed deterministic right-mon-R-automaton $M$ rewrites (deletes) on places which cannot increase their distance from the left end of the current word, if the grammar considered is linear (and LR(1)). Hence $M$ is also left-monotone. This completes the proof. $\square$

We mention in passing that it is shown in [6] that DLIN is complete for DSPACE($\log n$) with respect to deterministic logtime reductions.

### 7. Degrees of non-monotonicity for restarting automata

Let $j \geq 1$ be a natural number. We say that a sequence of cycles $S = (C_1, C_2, \ldots, C_n)$ is $j$-right-monotone if there is a partition of $S$ into $j$ interleaved subsequences

- $S_1 = (C_{1,1}, C_{1,2}, \ldots, C_{1,p_1})$,
- $S_2 = (C_{2,1}, C_{2,2}, \ldots, C_{2,p_2}),$
- $\ldots$
- $S_j = (C_{j,1}, C_{j,2}, \ldots, C_{j,p_j}),$

such that each $S_i, 1 \leq i \leq j$, is right-monotone. Analogously, the notions of $j$-left-monotonicity and of $j$-right-left-monotonicity are defined.

Obviously a sequence of cycles $(C_1, C_2, \ldots, C_n)$ is not $j$-right-monotone if and only if there exist indices $1 \leq i_1 < i_2 < \cdots < i_{j+1} \leq n$ such that

$$D_r(C_{i_1}) < D_r(C_{i_2}) < \cdots < D_r(C_{i_{j+1}}).$$

A corresponding observation holds for $j$-left- and $j$-right-left-monotonicity.

Let $j \geq 1$, and let $Y \in \{\text{right, left, right-left}\}$. A computation of a restarting automaton is called $j$-$Y$-monotone if the corresponding sequence of cycles is $j$-$Y$-monotone. Again the tail of the computation does not play any role here. An RLWW-automaton is called $j$-$Y$-monotone if all its computations that start with an initial configuration are $j$-$Y$-monotone. The prefixes $j$-$Y$-mon- are used to denote the corresponding classes of restarting automata. Observe that $1$-$Y$-monotonicity coincides with $Y$-monotonicity. Also notice that, based on Proposition 2.4, we can conclude that Corollary 2.5 extends to all levels $j$ of (right-, left-, and right-left-) monotonicity.

Our first result shows that the degree of right-monotonicity does not influence the expressive power of deterministic restarting automata, which is a slight generalization of a result of [27].

**Theorem 7.1.** For each $j \in \mathbb{N}_+$ and $X \in \{R, RR, RW, RRW, RWW, RRWW\}$,

$$\text{DCFL} = \mathcal{L}(\text{det-right-mon-X}) = \mathcal{L}(\text{det-j-right-mon-X}).$$
Proof. Let \( j > 1 \), and let \( M = (Q, \Sigma, \Gamma, \epsilon, q_0, k, \delta) \) be a deterministic \( j \)-right-monotone RRWW-automaton. In each cycle \( M \) scans at least one symbol which was written in the previous cycle, that is, per cycle the right distance can increase by at most \( k - 2 \). By condition (*) above this implies that, for each sequence of cycles \( C_1, C_2, \ldots, C_n \) that is part of a \( j \)-right-monotone computation of \( M \), and for each pair of indices \( i, l \) satisfying \( 1 \leq i < l \leq n \),

\[
D_r(C_i) \leq D_r(C_i) + (j - 1) \cdot (k - 2) < D_r(C_i) + (j - 1) \cdot k.
\]

Based on this observation we now construct a right-monotone RRWW-automaton \( M_j := (Q', \Sigma, \Gamma', \epsilon, q'_0, k', \delta') \) that accepts the same language as \( M \), where \( \Gamma' := \Gamma \cup \bar{T} \), and \( k' := (j + 1) \cdot k \). Here \( \bar{T} := \{ \bar{a} \mid a \in \Gamma \} \) is a new alphabet in one-to-one correspondence to \( \Gamma \). The symbols from \( \bar{T} \) will be used by \( M_j \) to mark those tape positions at which \( M \) has executed a Rewrite operation (see below).

The automaton \( M_j \) works as follows. The rightmost \( k + 1 \) positions of its read/write window are used as a kind of “look-ahead,” while the remaining part will be used to simulate the Rewrite steps of \( M \). \( M_j \) simulates \( M \) cycle by cycle. Performing MVR-steps, it scans its tape from left to right looking for the position at which \( M \) would now perform the next Rewrite operation, say \( u \to v \). Having found that position \( M_j \) keeps moving its read/write window to the right for \( k + 1 \) more steps.

(i) If the look-ahead does not contain a symbol from \( \bar{T} \) in one of its rightmost \( k \) positions, then \( M_j \) executes a Rewrite operation that consists of the following two parts:

(a) the actual Rewrite operation \( u \to v \) of \( M \) is being simulated;

(b) if the symbol \( a \) that is immediately to the right of the replaced factor \( u \) belongs to \( \Gamma \), then it is replaced by the corresponding symbol \( \bar{a} \in \bar{T} \), in this way marking the position of the current Rewrite for the next cycles.

Thereafter \( M_j \) scans the remaining part of the tape and restarts, accepts, or rejects just like \( M \).

(ii) If there is a symbol from \( \bar{T} \) in one of the rightmost \( k \) positions of the look-ahead, thus indicating that in a previous cycle a Rewrite operation was performed to the right of the position of the current Rewrite, then \( M_j \) moves its read/write window to the right until either the right end of the tape is reached or until the \( k \) rightmost positions of the look-ahead do not contain any symbols from \( \bar{T} \) anymore. As \( M \) is \( j \)-right-monotone, the place where the actual Rewrite operation is to be performed is less than \( (j - 1) \cdot k \) positions to the left of the rightmost position at which \( M \) has executed a Rewrite operation in a previous cycle (see (+)). Hence, the place where the actual Rewrite operation is to be performed is still contained in the read/write window of \( M_j \). From the considerations above it follows that the suffix of the tape content that has not yet been scanned in this cycle does not contain any symbols from \( \bar{T} \). Now \( M_j \) continues by executing the Rewrite step in the same way as in (i).

It follows that \( M_j \) accepts the language \( L(M) \). As observed above \( M \) scans in each cycle the rightmost “marked” symbol. Hence, \( M_j \) is necessarily right-monotone.

It follows that, for each \( j > 1 \),

\[
L(\text{det-}j\text{-right-mon-RRWW}) = L(\text{det-right-mon-RRWW}).
\]

As \( L(\text{det-right-mon-X}) = \text{DCFL} \) for all \( X \in \{ R, RR, RW, RRW, RWW, RRWW \} \) (see, e.g., [8]), the statement of the theorem follows. \( \square \)

Below we will show that, based on the degree of (right-, left-, right-left-) monotonicity, we obtain infinite hierarchies for restarting automata without auxiliary symbols in all other cases. First we will consider the degree of left-monotonicity and right-left-monotonicity for deterministic restarting automata.

Theorem 7.2. For each \( j \in \mathbb{N}_+ \) and for each \( X \in \{ R, RR, RW, RRW \} \),

(a) \( L(\text{det-j-left-mon-X}) \subset L(\text{det-(j + 1)-left-mon-X}) \).

(b) \( L(\text{det-j-right-left-mon-X}) \subset L(\text{det-(j + 1)-right-left-mon-X}) \).

Proof. For \( j \geq 2 \), let \( L^{(j)} \) denote the language

\[
L^{(j)} := \{ a^{m_1} b^{m_2} \cdots a^{m_l} b^{m_j} \mid m_1, \ldots, m_j > 0 \}.
\]

We claim that \( L^{(j)} \) is accepted by a \( j \)-right-left-monotone deterministic R-automaton.
Let $M^{(j)}$ be the R-automaton that is defined through the following sequence of meta-instructions (see, e.g., [25]):

$$(ab)^{i}a^{+}, abb \rightarrow b) \ (0 \leq i \leq j - 1),$$

$$(ab)^{j}$$(, Accept).

A meta-instruction of the form $(E, u \rightarrow v)$, where $E$ is a regular expression, and $u, v \in \Sigma^{*}$, $|u| > |v|$, describes possible cycles of computations of $M^{(j)}$, and a meta-instruction of the form $(E, \text{Accept})$, where $E$ is a regular expression, describes possible tails of accepting computations of $M^{(j)}$. In a restarting configuration $q_{0}wuS$, a meta-instruction is chosen nondeterministically. If the meta-instruction chosen has the form $(E, u \rightarrow v)$, then $M^{(j)}$ halts without accepting, if $w$ does not admit a factorization of the form $w = w_{1}uw_{2}$ satisfying $qw_{1} \in L(E)$. If, however, $w$ does admit such a factorization, then one such factorization is chosen nondeterministically, and the restarting configuration $q_{0}w_{1}vw_{2}S$ is reached. If the meta-instruction chosen has the form $(E, \text{Accept})$, then $M^{(j)}$ halts and accepts if $qwS \in L(E)$; otherwise it halts without accepting.

From the form of the above meta-instructions we see that they actually describe a deterministic computation, that is, $M^{(j)}$ is a deterministic R-automaton. It is easily verified that $M^{(j)}$ accepts the language $L^{(j)}$.

**Claim 1.** $M^{(j)}$ is $j$-right-left-monotone.

**Proof.** A computation of $M^{(j)}$ can be divided into $j$ left-monotone subsequences as follows: For each $t = 1, \ldots, j$, the $t$th subsequence consists of those cycles that delete factors of the form $ab$, where the corresponding prefix is of the form $q(ab)^{-1}a^{+}$. The $i$th cycle in the $t$th subsequence has left distance $1 + 2(t - 1) + m_{t} - i$, and hence, each of these subsequences is left-monotone.

As the union of these subsequences is the complete computation of $M^{(j)}$, we see that $M^{(j)}$ is indeed $j$-left-monotone. Further, the above computation is right-monotone, that is, $M^{(j)}$ is actually $j$-right-left-monotone. □

On the other hand we have the following negative result.

**Claim 2.** $L^{(j)} \notin L(\text{det-}(j - 1)-\text{left-mon-RRW})$.

**Proof.** Assume to the contrary that there is a deterministic RRW-automaton $M$ for $L^{(j)}$ that is $(j - 1)$-left-monotone. As $M$ has no auxiliary symbols, each Rewrite operation of $M$ that is applied during an accepting computation must transform a word from $L^{(j)}$ into another (shorter) word of $L^{(j)}$ by Fact 2.2. Let $r$ be a sufficiently large constant, larger than the size of $M$’s read/write window, and also larger than the constant $p$ from the Pumping Lemma (Fact 2.3), and let $w$ be the following element of $L^{(j)}$:

$$w := a^{i}b^{1}a^{2r}b^{2r} \cdots a^{jr}b^{jr}.$$  

No word from $L^{(j)}$ containing a subword $a^{p}$ can be accepted by $M$ without a Restart. The reason is that on the subword $a^{p}$ we can apply the Pumping Lemma to get an accepting tail for a word that does not belong to $L^{(j)}$.

There is only one type of Rewrite transition that $M$ can possibly apply to $w$: $M$ can replace a factor $a^{m}b^{m}$ by $a^{m-i}b^{m-i}$ for some $0 < i \leq m$.

If $M$ does not use such a transition on the first syllable of the form $a^{n}b^{n}$ to which it is applicable, then it will never be able to apply a transition of this form to that syllable, because $M$ is deterministic, and it cannot move left again on realizing that no transitions are applicable to any factor to the right of this particular syllable. Thus, $M$ must apply these transitions at the first possible position. This means that $M$ rewrites the syllables $a^{ir}b^{ir} (1 \leq i \leq j)$ strictly from left to right, that is, $M$ behaves essentially like the R-automaton $M^{(j)}$ above, that is, $M$ is in particular not $(j - 1)$-left-monotone. □

Hence, $L^{(j)} \in L(\text{det-}(j - 1)-\text{left-mon-R}) \setminus L(\text{det-}(j - 1)-\text{left-mon-RRW})$, which completes the proof of the theorem. □
Next we consider a sequence of languages that will be used as witness languages in further separation results. For \( j \in \mathbb{N}_+ \), let
\[
\tilde{L}_j := \{a^{n_1}b^{n_1}a^{n_2}b^{n_2} \ldots a^{n_t}b^{n_t} | n_1 \geq n_2 \geq \ldots \geq n_t \geq 1 \}.
\]

**Proposition 7.3.** For each \( j \geq 2 \),
\[
\tilde{L}_j \in \mathcal{L}(j\text{-right-left-mon-R}) \cap \mathcal{L}(\det-j\text{-right-left-mon-RL}).
\]

**Proof.** For a word of the form \( (a^+b^+)^j \) we denote the factors from \( a^+b^+ \) as “blocks.” We construct an R-automaton \( M \) for the language \( \tilde{L}_j \) as follows. In each cycle \( M \) guesses in which of the blocks a Rewrite is to be executed, and it removes a factor \( ab \) from that block, provided the block has length of at least four. However, \( M \) is allowed to make this operation in the \( r \)th block only if the parity of the number of \( a \)'s and \( b \)'s in all previous blocks is equal to the parity of the number of \( a \)'s in block \( t \). In this way we ensure that it is not possible to make two Rewrites in the \( r \)th block without any Rewrite in the \((r-1)\)st block. \( M \) accepts if and only if the tape contents is \( (ab)^j \). It is easily verified that in this way \( M \) recognizes the language \( \tilde{L}_j \) and that \( M \) is \( j \)-right-left-monotone, because its computations consist of \( j \) right-left-monotone sequences of cycles.

A deterministic RL-automaton can realize the same strategy. In each cycle it simply removes a factor \( ab \) from the last block for which the parity of the number of \( a \)'s and \( b \)'s is equal to the parity of the number of \( a \)'s in all previous blocks. Thus, it is \( j \)-right-left-monotone just as the R-automaton described above. \( \square \)

On the other hand there is the following negative result.

**Proposition 7.4.** For each \( j \geq 2 \),
\[
\tilde{L}_j \notin \mathcal{L}((j-1)\text{-right-mon-RLW}) \cup \mathcal{L}((j-1)\text{-left-mon-RLW}).
\]

**Proof.** Let \( M \) be an RLW-automaton for the language \( \tilde{L}_j \), and let \( p \) be the constant from the Pumping Lemma for \( M \). We will show that \( M \) is neither \((j-1)\)-right-monotone nor \((j-1)\)-left-monotone. For deriving this result, we consider a particular family of inputs
\[
F := \{a^{n+i_1}b^{n+i_1}a^{n+i_2}b^{n+i_2} \ldots a^{n+i_t}b^{n+i_t} | i_1, \ldots, i_t \geq 0, l_1, \ldots, l_t \geq 0 \},
\]
where \( \tau := p! \) and \( n \) is a sufficiently large integer. The basic input from the family \( F \) is the word of the form \( (a^nb^n)^j \), which belongs to \( \tilde{L}_j \). Hence, \( M \) has at least one accepting computation for this input.

Next we state some properties of accepting computations of \( M \) on the basic input and the relationship to possible computations of \( M \) on other inputs from \( F \).

**Claim 1.** Let \( \left. w_0 \right|_M \left. w_1 \right|_M \ldots \left. w_m \right|_M \) be an initial segment of an accepting computation of \( M \) on the basic input \( \left. w_0 := (a^nb^n)^j \right|_M \) such that each block \( a^ib^j \) of \( w_m \) still satisfies the condition \( i, l \geq p \). Then the following statements hold for each step of the above sequence:
(a) Each Rewrite transforms a factor \( a^t b^t \) into \( a^{s-t}b^{s-t} \), where \( 2s \) is not larger than the size of the window of \( M \) and \( 0 < t \leq s \).
(b) Each tape content \( w_i \) \((1 \leq i \leq m)\) belongs to the set \( \tilde{L}_j \).

**Proof.** The statements above follow from the fact that an accepting computation must not transform an input from the language \( \tilde{L}_j \) into a word which does not belong to \( \tilde{L}_j \), as such a word cannot possibly lead to acceptance due to the Error Preserving Property. \( \square \)

**Claim 2.** Assume that during an initial part of an accepting computation, \( M \) reduces the basic input into a word of the form \( a^{n-i_1}b^{n-r_1} \ldots a^{n-r_j}b^{n-r_j} \), where \( n - r_i > r \) for each \( 1 \leq i \leq j \). Then, for each input \( \left. a^{n_1}b^{m_1} \ldots a^{n_t}b^{m_t} \right|_M \) from the family \( F \), there is a computation of \( M \) which reduces this word into the word \( a^{n_1-r_1}b^{m_1-r_1} \ldots a^{n_t-r_t}b^{m_t-r_t} \).

**Proof.** It follows from Claim 1(a) that the considered initial part of the accepting computation on the basic input does not change the structure of the tape content. The Pumping Lemma implies that one can perform pumping on each of
the \(a\)-syllables and \(b\)-syllables of the basic input. Thus, taking a multiple of all the possible lengths of such “pumped” words \(z\) (see the Pumping Lemma) we may use pumping for the whole of the above computation (note that we pump one-letter blocks). By the Pumping Lemma no \(z\) is longer than the constant \(p\), so \(r = p!\) is a multiple of all the possible lengths of the factors that are pumped. \(\square\)

**Claim 3.** Assume that \(M\) reduces the basic input into \(a^{n_1}b^{n_1} \cdots a^{n_j}b^{n_j}\) during an initial part of an accepting computation, where \(n_i > r\) for each \(1 \leq i \leq j\). Then \(0 < n_i - n_{i+1} \leq r\) for each \(1 \leq i < j\), and hence, \(0 \leq n_i - n_i \leq (l - i)r\) for each \(1 \leq i \leq l \leq j\).

**Proof.** Assume to the contrary that there is an initial part of an accepting computation \((a^n b^n)^j \cdots a^{n_1}b^{n_1}\) such that \(n_i - n_{i+1} < 0\) or \(n_i - n_{i+1} > r\) for some \(1 \leq i < j\). We consider two cases.

Case 1. \(n_i - n_{i+1} < 0\) for some \(1 \leq i < j\). Then \(n_i < n_{i+1}\), which implies that \(a^{n_i}b^{n_i} \cdots a^{n_j}b^{n_j} \notin \tilde{L}_j\). This means that we do not consider an accepting computation (see Claim 1(b)), contradicting our hypothesis.

Case 2. \(n_i - n_{i+1} > r\) for some \(1 \leq i < j\). Then by Claim 2 we obtain that

\[
\tilde{L}_j \neq (a^n b^n)^j(a^n b^n)^{i-j}(a^n b^n)^{i-j}(a^n b^n)\cdots a^{n_1}b^{n_1}a^{n_1+r}b^{n_1+r}a^{n_1+r}b^{n_1+r}a^{n_1+r}b^{n_1+r},
\]

where \(n_1 \geq \cdots \geq n_j\) and \(n_i + r \geq \cdots \geq n_j + r\), as \(a^{n_1}b^{n_1} \cdots a^{n_j}b^{n_j} \notin \tilde{L}_j\). Further, \(n_i \geq n_{i+1} + r\), because of our hypothesis. Thus, the above computation contradicts the Error Preserving Property. \(\square\)

It follows from Claim 3 that each accepting computation of \(M\) for the basic input \((a^n b^n)^j\) must reduce all \(j\) blocks in an almost synchronous manner. Hence, \(M\) is neither \((j-1)\)-right-monotone nor \((j-1)\)-left-monotone. This completes the proof of Proposition 7.4. \(\square\)

From the above results on the languages \(\tilde{L}_j\) \((j \geq 2)\) we obtain the following proper inclusion results.

**Theorem 7.5.** For each \(j \in \mathbb{N}_+\) and each \(Y \in \{\text{right, left, right-left}\},\)

(a) for each \(X \in \{R, RW, RR, RRW, RL, RLW\}\), \(L((j+1)-Y\text{-mon}-X) \subseteq L((j)-Y\text{-mon}-X)\).

(b) for each \(X \in \{RL, RLW\}\), \(L(\text{det-}Y\text{-mon}-X) \subseteq L(\text{det-}(j+1)-Y\text{-mon}-X)\).

8. Comparing the classes at the same level of monotonicity

In the proof of Theorem 7.2 we have seen that the language \(L^{(j)}\) cannot be accepted by any deterministic \((j-1)\)-left-monotone RRW-automaton. Contrasting this result, we see below that auxiliary symbols do help in accepting this language.

**Proposition 8.1.** For each \(j \geq 2\), \(L^{(j)} \in L(\text{det-left-mon-RWW})\).

**Proof.** Let \(j \geq 2\). We describe an RWW-automaton \(M\) for the language \(L^{(j)}\).

1. First \(M\) moves its read/write window all the way to the right, verifying that the given input is of the form \(w := a^{m_1}b^{n_1}a^{m_2}b^{n_2} \cdots a^{m_j}b^{n_j}\) for some positive integers \(m_1, n_1, m_2, n_2, \ldots, m_j, n_j\). In the negative it rejects immediately, in the affirmative it goes to (2).

2. Using the auxiliary symbol \(B_j\), \(M\) rewrites the suffix \(b^{n_j}\) into \(B_j^{n_j/2}\) within \(n_j/2\) cycles. If \(n_j\) is not an even number, then the factor \(ab\) of \(a^{m_j}b^{n_j}\) is deleted in this process.

3. Then within the next \(n_j/2\) cycles it is checked whether \(m_j = n_j\) holds by deleting factors of the form \(a^2B_j\).

This phase ends by generating an occurrence of the auxiliary symbol \(A_j\) in the affirmative.

4. Now steps (2) and (3) are repeated for the factors \(a^{m_{j-1}}b^{n_{j-1}}, a^{m_{j-2}}b^{n_{j-2}}\) down to \(a^{m_1}b^{n_1}\).

5. \(M\) accepts if the tape content is of the form \(A_1A_2 \cdots A_j\).

Obviously \(M\) is a deterministic RWW-automaton, and it is easily seen that \(M\) accepts the language \(L^{(j)}\) and that it is left-monotone. Hence, we see that \(L^{(j)} \in L(\text{det-left-mon-RWW})\). \(\square\)

Together with the fact that \(L^{(j)} \notin L(\text{det-}(j-1)\text{-left-mon-RRW})\), this yields the following separation results.
Corollary 8.2. For each \( j \in \mathbb{N}_+ \),
(a) \( \mathcal{L}(\text{det-} j\text{-left-mon-RW}) \subseteq \mathcal{L}(\text{det-} j\text{-left-mon-RWW}) \).
(b) \( \mathcal{L}(\text{det-} j\text{-left-mon-RRW}) \subseteq \mathcal{L}(\text{det-} j\text{-left-mon-RRWW}) \).

Actually the above example languages will give us still further separation results. Let \( j > 1 \), and let \( M \) be a det-left-mon-RWW-automaton for the language \( L(j) \). By including all the auxiliary symbols of \( M \) in its input alphabet, we obtain a det-left-mon-RW-automaton \( M' \) for some language \( \hat{L}(j) \). Observe that \( \hat{L}(j) \cap \{a, b\}^* = L(j) \) holds.

Lemma 8.3. For each \( j > 1 \), \( \hat{L}(j) \notin \mathcal{L}(\text{det-(} j-1\text{-left-mon-RR}) \).

Proof. If we had a deterministic \(( j-1\text{-left-monotone RR-automaton} \) \( M'' \) for the language \( \hat{L}(j) \), we could simply turn it into a deterministic \(( j-1\text{-left-monotone RR-automaton} \) for the language \( L(j) \) by requiring that it checks that its tape content is a string over \( \{a, b\} \), that is, as soon as a symbol different from \( a \) or \( b \) is detected, the automaton halts and rejects. This, however, contradicts the fact that \( L(j) \notin \mathcal{L}(\text{det-(} j-1\text{-left-mon-RR}) \). \( \square \)

Thus, we obtain the following separation results.

Corollary 8.4. For each \( j \in \mathbb{N}_+ \),
(a) \( \mathcal{L}(\text{det-} j\text{-left-mon-RR}) \subseteq \mathcal{L}(\text{det-} j\text{-left-mon-RW}) \).
(b) \( \mathcal{L}(\text{det-} j\text{-left-mon-RRW}) \subseteq \mathcal{L}(\text{det-} j\text{-left-mon-RRWW}) \).

For deriving a corresponding result separating the RR(W)-classes from the R(W)-classes, we consider the following example language
\[
L_1 := L_{11} \cup L_{12} \cup L_{13} \cup L_{14},
\]
where
\[
L_{11} := \{a^m a^n (bc)^n f a^m \mid m, n > 0\},
L_{12} := \{a^m a^n (bc)^i b l a^m \mid m, n, j > 0, i > 0, n = i + j\},
L_{13} := \{a^n (bc)^i c^j n, j > 0, i > 0, n = 2(i + j)\},
L_{14} := \{a^m (bc)^i f a^k f \mid m, n, k > 0\}.
\]

Lemma 8.5. \( L_1 \in \mathcal{L}(\text{det-left-mon-RR}) \setminus \mathcal{L}(\text{det-RR}) \).

Proof. First, we will show that \( L_1 \) is accepted by a det-left-mon-RR-automaton \( M \). This automaton works as follows:

1. On a word with prefix \( a^+ (bc)^+ f \), the automaton \( M \) deletes the factor \( cf \) from the factor \( bcf \) and scans the rest of the current tape content. If the suffix is of the form \( a^+ f \), then \( M \) accepts, as the tape content belongs to \( L_{14} \), if the suffix is of the form \( a^+ \), then \( M \) restarts, as the tape content is possibly a word from \( L_{11} \), otherwise \( M \) rejects.

2. On a word with prefix \( a^+ (bc)^+ ba \) or \( a^+ (bc)^+ bb \), \( M \) deletes the letter \( c \) from the factor \( bcb \) or \( bcbb \), respectively, and restarts, as the tape content is a possibly a word from \( L_{12} \).

3. On a word with prefix \( a^+ (bc)^+ c \), \( M \) deletes the letter \( b \) from the factor \( bcc \) and restarts, as the tape content is a possibly a word from \( L_{13} \).

4. On a word with prefix \( a^+ bb \), \( M \) deletes the factor \( ab \) from the factor \( abb \) and restarts, as the tape content is a possibly a word from \( L_{12} \).

5. On a word with prefix \( a^i cc \), where \( i > 1 \), \( M \) deletes the factor \( aac \) and restarts, as the tape content is a possibly a word from \( L_{13} \).

6. On a word with prefix \( a^i ba \), where \( i \geq 3 \), \( M \) deletes the first and the last occurrence of the letter \( a \) from the factor \( aaba \) and restarts, as the tape content is a possibly a word from \( L_{12} \).

7. \( M \) immediately accepts the words \( aaba \) and \( aac \).

8. If the tape content does not meet any of the above cases, \( M \) rejects.

It is easily seen that \( M \) accepts the language \( L_1 \), and that \( M \) is deterministic and left-monotone.
Next we show that $L_t$ cannot be accepted by any deterministic RW-automaton. Assume to the contrary that there exists a deterministic RW-automaton $M'$ that accepts $L_t$. Let us consider the accepting computation of $M'$ on the word $w := a^m a^m (bc)^m f a^m$, where $m$ is larger than the constant $p$ for $M'$ from the Pumping Lemma (Fact 2.3). What is the Rewrite step that $M'$ can execute in the first cycle $w \vdash_c^t M'$? v of this computation? Because of the Correctness Preserving Property (Fact 2.2), $v$ belongs to $L_t$. There are four possibilities:

- Either $v \in L_{t1}$, that is, $v = a^n a^n (bc)^n f a^m$ for some $n < m$. In this case, the Rewrite step occurred at the border between $a^m$ and $(bc)^m$. Hence, $M'$ would perform the same Rewrite step on the word $a^m a^m (bc)^m f a^m$ from $L_{t1}$. This would yield the word $a^m a^m (bc)^m f a^m$, which does not belong to the language $L_t$, as $m + n > 2n$, thus contradicting the Correctness Preserving Property.

- The second possibility is that $v \in L_{t2}$, that is, $v = a^m a^m (bc)^m f a^m$ for some $n < m$. In this case, the Rewrite step occurred at the border between $(bc)^m$ and $f a^m$. Hence, $M'$ would perform the same Rewrite step on the word $a^m a^m (bc)^m f a^m$ from $L_{t2}$, obtaining the word $a^m a^m (bc)^m f a^m$, which does not belong to the language $L_t$. This contradicts the Correctness Preserving Property.

- The third possibility is that $v \in L_{t3}$, that is, $v = a^n (bc)^n b^n a^m$ for some integer $n$ satisfying $0 < n < m$. This, however, is impossible, as in a single Rewrite step, $M'$ cannot replace the suffix $(bc)^n f a^m$ of $w$ of length $2n + 1 + m$ by the word $c^n$.

- Finally it is possible that $v \in L_{t4}$, that is, $v = a^m (bc)^m f a^m$ for some $n < m$. In this case, the Rewrite step was applied to a suffix of $w$. By applying the Pumping Lemma (Fact 2.3) to the suffix $a^m$ of $w$, we see that, for some integer $0 < i < p$, $M'$ would execute the following cycle:

$$w' := a^m (bc)^m f a^{m+i} \vdash_c^t M' a^m (bc)^m f a^{m+i} f =: v'.$$

However, $v' \in L_{t4}$, while $w' \notin L_t$, which contradicts the Error Preserving Property. As this covers all cases, we see that $L_t \notin L(\text{det-RW})$, thus completing the proof. $\square$

This technical result yields the following proper inclusion results.

**Corollary 8.6.** For each $j \geq 1$,

(a) $L(\text{det-} j\text{-left-mon-R}) \subset L(\text{det-} j\text{-left-mon-RR}).$

(b) $L(\text{det-} j\text{-left-mon-RW}) \subset L(\text{det-} j\text{-left-mon-RRW}).$

It is not hard to see that the language

$$L_a := \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^n d \mid n \geq 0\}$$

can be recognized by a det-right-left-mon-RL-automaton. On the other hand, it is well-known that $L_a \notin \text{DCFL}$, and it is easily seen that $L_a$ is not accepted by any deterministic RRW-automaton. Together with Theorem 7.1 this yields the following proper inclusions.

**Corollary 8.7.**

(a) For each $j \in \mathbb{N}_+$, $X \in \{WW, W, \varepsilon\}$, and $Y \in \{\text{right, right-left}\}$, $L(\text{det-} j\text{-Y-mon-RRX}) \subset L(\text{det-} j\text{-Y-mon-RLX}).$

(b) For each $j \in \mathbb{N}_+$ and $X \in \{W, \varepsilon\}$, $L(\text{det-} j\text{-left-mon-RRX}) \subset L(\text{det-} j\text{-left-mon-RLX}).$

From the fact that $L_a \in L(\text{det-right-left-mon-RL})$ and Corollary 2.5 it follows that $L_a \in L(\text{right-left-mon-RR})$. As it is further known ([8, Lemma 4.1]) that $L_a \notin L(\text{RW})$, this has the following consequences.

**Corollary 8.8.** For each $j \geq 1$ and each $Y \in \{\text{right, left, right-left}\}$,

(a) $L(j\text{-Y-mon-R}) \subset L(j\text{-Y-mon-RR}).$

(b) $L(j\text{-Y-mon-RW}) \subset L(j\text{-Y-mon-RRW}).$

The following result shows that for all types of monotone restarting automata without auxiliary symbols, the deleting model is strictly weaker than the rewriting model.
Theorem 8.9. For each $j \in \mathbb{N}_+$, $X \in \{R, RR, RL\}$, and $Y \in \{\text{right, left, right-left}\}$,

$$L(j-Y-mon-X) \subseteq L(j-Y-mon-XW).$$

Proof. The inclusions are obvious. To prove that they are proper, we consider the witness language

$$L_b := \{c^n f d^n, c^n eed^m | n \geq 0\} \cup \{c^n gd^m, c^n eed^m | m > 2n \geq 0\}.$$ 

This language can be recognized by a right-left-monotone RW-automaton $M$ that works as follows:

- $M$ immediately accepts the word $f$.
- If the word starts with $c$’s and contains $cfd$, then $M$ simply replaces $cfd$ by $f$ and restarts.
- If the word starts with $c$’s and contains $cgdd$, then $M$ simply replaces $cgdd$ by $g$ and restarts.
- If the word starts with $gd$, then $M$ scans the rest of the word. If it contains only $d$’s, then $M$ accepts, otherwise it rejects.
- If the word starts with $c$’s followed by $ee$, then $M$ nondeterministically renwrites $ee$ by $f$ or by $g$ and restarts.

As all rewrite steps take place “in the middle” of the word, it is easily seen that $M$ is right-left-monotone and that $L(M) = L_b$.

To complete the proof it suffices to show that $L_b$ cannot be recognized by any RL-automaton. Assume to the contrary that $L_b = L(M')$ for some RL-automaton $M'$ with read/write window of size $k$. Let $p$ be the constant for $M'$ that we obtain from the Pumping Lemma (Fact 2.3), and let $n > k$ be a sufficiently large number that is divisible by $p!$.

Consider an accepting computation of $M'$ on input $c^n eed^n$. Obviously this computation consists of at least two cycles. Let $C$ be the first of these cycles. During this cycle $M'$ can only shorten both segments of $c$’s and $d$’s in the same way, that is, $c^n eed^n \not\subseteq L_M$, $c' eed^l$ for some $l < n$. By Fact 2.3, a factor $d^s (s > 0)$ of $d^n$ is a pumping subword with respect to this cycle, implying that $c^n eed^{n-s+i-s} M, c' eed^{l-s+i-s}$ for all $i \geq 0$. Obviously, $s$ divides $n$ due to our choice of $n$. Hence, for $i = n/s + 1$, we get

$$n - s + \left(\frac{n}{s} + 1\right) s = 2n \quad \text{and} \quad l - s + \left(\frac{n}{s} + 1\right) s = l + n.$$ 

Now $c^n eed^{2n} \not\in L(M')$, but we have $c^n eed^{2n} \subseteq L_c \subseteq L(M) \supseteq c' eed^{n+l}$ by the natural extension of the cycle $C$. As $2l < n + l$, we have $c' eed^{n+l} \in L(M')$, which contradicts the Error Preserving Property. $\square$

In [8] it is shown that the language

$$L_c := \{c^n d^n | n \geq 0\} \cup \{c^n d^m | m > 2n > 0\}$$

is not accepted by any RRW-automaton, and hence, $L_c \not\in L(RLW)$ by Proposition 2.4. On the other hand, it is easy to construct a (nondeterministic) right-left-mon-RWW-automaton for $L_c$. Hence, we obtain the following result.

Corollary 8.10. For each $j \geq 1$ and each $Y \in \{\text{right, left, right-left}\}$,

(a) $L(j-Y-mon-RLW) \subseteq L(j-Y-mon-RLWW)$,
(b) $L(j-Y-mon-RRW) \subseteq L(j-Y-mon-RRWW)$,
(c) $L(j-Y-mon-RW) \subseteq L(j-Y-mon-RWW)$.

Let $L_d := \{cwcw | w \in \{a, b\}^* \text{ and } |w| = 2n \text{ for some } n \geq 0\}$. Concerning this language we have the following result.

Proposition 8.11. $L_d \in L(\text{det-2-right-left-mon-RLWW}) \setminus L(RLW)$. 
Proof. It is straightforward to show that $L_d$ is not accepted by any RLW-automaton. Essentially this follows from the fact that without the use of auxiliary symbols a single Rewrite operation is in general not sufficient to transform a word from $L_d$ into another word from $L_d$.

It remains to prove that $L_d$ belongs to the class $\mathcal{L}($det-2-right-left-mon-RLWW$)$. Below we sketch a det-2-right-left-mon-RLWW-automaton $M_d$ for $L_d$. This automaton has one auxiliary symbol $D$ and it works as follows:

- $M_d$ accepts $cc$ and $DD$ without a Restart.
- On words of length at least 6, $M_d$ works according to the following table, where $x, y \in \{a, b\}$, and $w, w' \in \{aa, bb, ab, ba\}^*$:

<table>
<thead>
<tr>
<th>Word of the form</th>
<th>Rewrites into</th>
</tr>
</thead>
<tbody>
<tr>
<td>$cxywcxyw'$</td>
<td>$Dwxcxyw'$</td>
</tr>
<tr>
<td>$Dwxcyw'$</td>
<td>$DwDw'$</td>
</tr>
<tr>
<td>$DxywDxyw'$</td>
<td>$cwDxyw'$</td>
</tr>
<tr>
<td>$cwDxyw'$</td>
<td>$cwcw'$</td>
</tr>
</tbody>
</table>

- If the content of its tape has any other form, then $M_d$ rejects.

Clearly, as an RLW-automaton, $M_d$ can first read its tape completely and can then perform the corresponding Rewrite transition deterministically. All Rewrite steps are done either at the beginning or in the middle of the current tape content, and therefore $M_d$ is a det-2-right-left-mon-RLWW-automaton.

By induction on the length of $w$, it can be shown that the computation of $M_d$ is accepting on all words of the form $cwcw$ or $DwDw$, where $w \in \{a, b\}^*$, $|w| = 2n$ for some $n \geq 0$. For $n = 0$, this is trivial. The induction step follows easily: For each $w \in \{a, b\}^*$, $|w| = 2n$, $x, y \in \{a, b\}$, we have the following computations:

$$cxywcxyw' \xrightarrow{c} M_d Dwxcyw' \xrightarrow{c} M_d DwDw,$$

and

$$DxywDxyw' \xrightarrow{c} M_d cwDxyw' \xrightarrow{c} M_d cwcw.'$$

On the other hand, any word from $\{a, b, c\}^* \setminus L_d$ is rejected by $M_d$, because:

- $M_d$ rejects all words from $\{a, b, c\}^*$ which are not of the form $cwcw'$, where $w, w' \in \{a, b\}^*$ and both $w, w'$ are of even length.
- On a word $cucvcuv'$, where $u, v, v' \in \{a, b\}^*$, $|u| = 2n$ for some $n \geq 0$, and $v$ differs from $v'$ in the first or the second letter,

$$cucvcuv' \xrightarrow{c} M_d DuDv' or cucvcuv' \xrightarrow{c} M_d cwcw'$$

and $DuDv'$ as well as $cwcw'$ are rejected.

Thus, $L(M_d) = L_d$, that is, $L_d \in \mathcal{L}($det-2-right-left-mon-RLWW$)$. □

Further, let $M_d'$ be the RLW-automaton that is obtained from $M_d$ by including the symbol $D$ into its input alphabet, and let $L_d' := \mathcal{L}(M_d')$. By using similar arguments as in the proof of Proposition 8.11, the following result can be shown.

Proposition 8.12. $L_d' \in \mathcal{L}($det-2-right-left-mon-RLW$) \setminus \mathcal{L}($RL$)$.

Together, these technical results yield the following consequences.

Theorem 8.13. For each $j \geq 2$ and $Y \in \{right, left, right-left\}$,

(a) $\mathcal{L}($det-$j$-$Y$-mon-RLW$) \subset \mathcal{L}($det-$j$-$Y$-mon-RLWW$),$

(b) $\mathcal{L}($det-$j$-$Y$-mon-RL$) \subset \mathcal{L}($det-$j$-$Y$-mon-RLW$).$
9. Separating the second from the first level of monotonicity for restarting automata with auxiliary symbols

From the Pumping Lemma for linear languages (see, e.g., [1]) it is easily seen that the language $L^{(2)}$ considered in the proof of Theorem 7.2 is not linear. As this language is accepted by a det-2-right-left-mon-R-automaton, Theorems 6.1 and 6.2 yield the following proper inclusions.

**Corollary 9.1.** For each $X \in \{ \text{RWW}, \text{RRWW} \}$,

(a) $L(\text{det-right-left-mon-X}) \subset L(\text{det-2-right-left-mon-X})$.
(b) $L(\text{right-left-mon-X}) \subset L(\text{2-right-left-mon-X})$.

The language $L_d$ of Proposition 8.11 is not context-free. On the other hand, we have the following containment result for $L_d$.

**Proposition 9.2.** $L_d \in L(\text{2-right-mon-RWW}) \cap L(\text{2-left-mon-RWW})$.

**Proof.** We use the technique developed in [10] for constructing an RWW-automaton for the Gladkij-language. Let $M$ be the RWW-automaton that proceeds as follows:

1. An input of the form $cucv$ ($u, v \in \{a, b\}^*$) is compressed into a word of the form $cu_1cv_1$. This compression is performed from left to right, encoding two symbols of $u$ into one (auxiliary) symbol of $u_1$ and encoding two symbols of $v$ into one (auxiliary) symbol of $v_1$. If this process succeeds, then $u$ and $v$ are both of even length, and $v$ is a subsequence of $u$. Obviously this part of the computation can be performed in a 2-right-monotone way.
2. If the compression succeeds, then it is verified whether $u_1$ and $v_1$ are of the same length. This is done by simply erasing the two symbols surrounding the rightmost occurrence of the symbol $c$ in each cycle.

It is obvious that $M$ accepts the language $L_d$. Further, the second part of the computation above can be combined with the leftmost subsequence of the first part into a right-monotone sequence. Thus, $M$ is 2-right-monotone. If, instead of doing the compression from left to right, the compression is performed from right to left, then we obtain a 2-left-monotone RWW-automaton for $L_d$. □

Together with Theorem 4.3, Proposition 8.11, and Corollary 9.1(b) this gives the following results.

**Corollary 9.3.** For each $Y \in \{ \text{right}, \text{left}, \text{right-left} \}$,

(a) $L(\text{det-Y-mon-RLWW}) \subset L(\text{det-2-Y-mon-RLWW})$.
(b) $L(\text{Y-mon-RWW}) \subset L(\text{2-Y-mon-RWW})$.
(c) $L(\text{Y-mon-RRWW}) \subset L(\text{2-Y-mon-RRWW})$.
(d) $L(\text{Y-mon-RLWW}) \subset L(\text{2-Y-mon-RLWW})$.

We have seen in Theorem 7.1 that for deterministic R(W)(W)- and RR(W)(W)-automata, the degree of right-monotonicity does not influence the expressive power. Here we will show that this result does not extend to left-monotonicity. In fact, we will separate the language classes defined by the various deterministic 2-left-monotone restarting automata from the classes defined by the corresponding deterministic left-monotone automata. For doing so we consider the language $L := L_1 \cup L_2$, where

$L_1 := \{ a_5^m a_4^m a_3^m a_2^n a_1^p \mid n, m, p > 0 \}$ and $L_2 := \{ a_5^m a_4^m a_3^m a_2^p a_1^l \mid n \neq l, n, l, m, p > 0 \}$.

First, we show that $L$ is accepted by a deterministic 2-left-monotone R-automaton.

**Proposition 9.4.** $L \in L(\text{det-2-left-mon-R})$.

**Proof.** We describe an R-automaton $M$ for $L$ through a sequence of meta-instructions (cf. the proof of Theorem 7.2):

1. $(q \cdot a_5^m, a_5^2 a_4^2 \rightarrow a_5 a_4)$,
(2) \((qa_5a_4 \cdot a_5^2a_2^2 \rightarrow a_3a_2)\),
(3) \((qa_5a_4a_3a_2 \cdot a_1^1 \cdot \$, Accept)\
(4) \((qa_5a_4 \cdot a_4^1 \cdot a_3^2 \cdot a_2^2 \rightarrow a_2a_1)\),
(5) \((qa_5 \cdot a_4^2 \cdot a_4 \cdot a_3^+ \cdot a_2 \cdot a_2^2a_1^2 \rightarrow a_2a_1)\),
(6) \((qa_5 \cdot a_4^2 \cdot a_3^2 \cdot a_2a_1\$, Accept)\
(7) \((qa_5 \cdot a_4 \cdot a_4^2 \cdot a_3^+ \cdot a_2a_1\$, Accept)\

Using instruction (1) \(M\) reduces a given input until only one occurrence of the letter \(a_5\) or until only one occurrence of the letter \(a_4\) is left on the tape. If now \(a_5a_4a_3\) is a prefix of the actual tape content, then it remains to check whether the current tape content belongs to the sublanguage \(L_1\), which is done by using instructions (2) and (3). If, however, \(a_5a_4^2\) or \(a_5^2a_4a_3\) for some \(s > 1\) is a prefix of the actual tape content, then it remains to check whether the current tape content belongs to the sublanguage \(L_2\), which is done by using instructions (4) to (7). It follows that \(M\) accepts the language \(L\), and that it is actually deterministic. Notice that \(M\) makes two sequences of Rewrite steps: first it rewrites on the border between \(a_5^2\) and \(a_4^2\) and then it rewrites on the border between \(a_5^2\) and \(a_3^2\) or on the border between \(a_5^2\) and \(a_1^1\). Thus, \(M\) is 2-left-monotone. □

On the other hand, we have the following negative result.

**Theorem 9.5.** \(L \notin \mathcal{L}\) (det-left-mon-RRWW).

**Proof.** Assume to the contrary that there exists a left-monotone deterministic RRWW-automaton \(M = (Q, \Sigma, \Gamma, s, \delta, 0, \alpha, \beta, \gamma)\) for the language \(L\). Let \(\overline{L}_M = \{y \in (\Sigma \times Q)^* \mid y = \text{reach}_M(w)\text{ for some }w \in L(M)\}\) (see Definition 5.7). By \(\overline{L}\) we denote the language \((\overline{L}_M)^R\). Then we see from Lemma 5.8 that \(\overline{L} \in \text{DCFL}\). Hence by Theorem 7.1, there exists a deterministic right-monotone R-automaton \(M'\) recognizing \(\overline{L}\).

Let \(p\) be the constant for the RRWW-automaton \(M\) from the Pumping Lemma (Fact 2.3), and let \(j\) be a number larger than \(k'\) that is divisible by \(p\), where \(k'\) is the size of the read/write window of \(M'\). The word \(x := a_4^4a_2^2a_2^2a_2^1\) belongs to \(L\), and therewith \(y := (\text{reach}_M(x))^R \in \overline{L}\). The word \(y\) cannot be accepted by \(M'\) in a tail computation, as otherwise we would get an accepting tail computation for some word not belonging to \(\overline{L}\) by pumping (using Fact 2.3). So let \(y' \in \overline{L}_M\). \(y'\) be the first cycle of the accepting computation of \(M'\) on input \(y\). Because of the Correctness Preserving Property (Fact 2.2), \(y' \in \overline{L}\). There are two possibilities for the place at which \(M'\) performs the deletion during this cycle:

(i) at the border between the parts corresponding to \(a_1^2j\) and \(a_2^2j\), or
(ii) somewhere inside the part corresponding to \(a_3^3a_4^1a_5^j\).

In case (i), \(\text{src}(y') = a_4^2j-a_2^2j-a_3^j-a_4^ja_5^j\) for some integer \(s\) satisfying \(0 < s < k'\). As \(M'\) is an R-automaton, \(M'\) will restart immediately after rewriting.

The RRWW-automaton \(M\) for the language \(L\) gets into a cyclic behaviour after at most \(|Q|\) many steps, when it moves across a long one-letter block. As \(M\) only executes a single Rewrite operation per cycle, it can perform this Rewrite operation inside a long one-letter block only before it gets into a cyclic behaviour, that is, within a short prefix of that block, or at the end of that block. Accordingly, in each block \((a_i, q_1)(a_i, q_2)\cdots(a_i, q_j)\) (for any \(i \in \{1, \ldots, 5\}\) or \(i = \text{reach}_M(x)\), we can find a pumping subword of length \(0 < r_i < |Q|\) (\(p\)). This subword can then be pumped \(j/r_i\) many times, in this way increasing the length of the block of symbols corresponding to \(a_i\) by \(j\). By performing this process on the block corresponding to \(a_3\) and on the block corresponding to \(a_5\), we obtain a cycle of \(M\) that starts with the tape inscription \(x' := a_2^2j^2a_4^ja_3^ja_2^j\). In addition, the suffixes of length \(4j\) of the words \(\text{reach}_M(x) \in \overline{L}^R\) and \(\text{reach}_M(x') \in \overline{L}^R\) coincide.

As \(M'\) is a deterministic R-automaton, it will perform the same Rewrite operation when starting with the tape inscription \(z := (\text{reach}_M(x'))^R \in \overline{L}\) as it performs on the tape inscription \(y\), that is, we obtain a cycle \(z \circ M' \circ z'\), where \(\text{src}(z') = a_2^2j-a_2^2j-a_3^ja_2^ja_5^j\). However, \(\text{src}(z') \notin \overline{L}^R\) implying that \(z' \notin \overline{L}\), which contradicts the Correctness Preserving Property for \(M'\).
In case (ii), the automaton $M'$ performs a Rewrite operation inside the part corresponding to $a^j_1 a^j_2 a^j_3$, when it works on the tape content $y = (\text{reach}_M(x))^R$, where $\text{src}(y) = a^j_1 a^j_2 a^j_3 a^j_4 a^j_5$. As $M'$ is deterministic, any deletion of symbols within the block corresponding to $a^j_2$ in subsequent cycles must start by deleting symbols at the right end of this block. However, in this way $M'$ would obtain a word $z'$ that does not belong to the language $L$, because $\text{src}(z') = a^j_1 a^j_2 a^j_3 a^j_4 a^j_5$, where $2j' - k' = 2j$ and $m_2 > j > m_3$. Hence, in case (ii), $M'$ cannot perform any Rewrite steps within the prefix corresponding to $a^j_1 a^j_2$. This means, however, that by applying pumping to the prefix of $y$ corresponding to $a^j_1$, we obtain an accepting computation of $M'$ for a word that does not belong to $L$.

Thus, $L$ is not accepted by any deterministic right-monotone R-automaton, which contradicts the assumption that $L \in \mathcal{L}$(det-left-mon-RWW). \hfill \Box

From this theorem and Proposition 9.4 we obtain the following proper inclusions.

**Theorem 9.6.**

(a) $\mathcal{L}$(det-left-mon-RWW) $\subsetneq$ $\mathcal{L}$(det-2-left-mon-RWW).

(b) $\mathcal{L}$(det-left-mon-RWW) $\subsetneq$ $\mathcal{L}$(det-2-left-mon-RWW).

10. **Beyond $j$-monotonicity**

In Corollary 8.7 we have seen that deterministic RL-automata are more powerful than the corresponding deterministic RR-automata. Here we study the relationship between the deterministic types of RR- and RL-automata and the nondeterministic $j$-monotone variants. For that we consider the language $L_{1r} := L_1 \cup L_r$, where

$L_1 := \{a^{2^n-2i}la^i \mid n \geq 1, 0 \leq 2i < 2^n\},$

$L_r := \{a^i ra^{2^n-2i} \mid n \geq 1, 0 \leq 2i < 2^n\}.$

**Proposition 10.1.** $L_{1r} \in \mathcal{L}$(det-RW) $\setminus$ $\mathcal{L}$(RL).

**Proof.** In [17] a deterministic RW-automaton $M_{1r}$ for $L_{1r}$ is presented. Hence, it remains to show that $L_{1r}$ is not accepted by any RL-automaton. Assume that $M$ is an RL-automaton with read/write window of size $k$ that recognizes $L_{1r}$. A sufficiently long input word $w = a^{2^n}l$ cannot be accepted directly by a tail computation. After the first cycle of an accepting computation on $w$, $M$ obtains a word $w_1$. This word has the form $w_1 = a^{2^n-i}l$, where $0 < i < k$, or the form $w_1 = a^{2^n-i}a^j$, where $0 < j < k$. As $L_{1r}$ does not contain any words of either of these forms, this contradicts our assumption that $M$ recognizes $L_{1r}$. \hfill \Box

This technical result yields the following consequences.

**Corollary 10.2.** For each $X \in \{R, RR, RL\}$,

$\mathcal{L}(\text{det-X}) \subseteq \mathcal{L}(\text{det-XW})$ and $\mathcal{L}(X) \subseteq \mathcal{L}(XW).$

The next proposition says that the language $L_{1r}$ is essentially non-monotone.

**Proposition 10.3.** Let $j \in \mathbb{N}_+$ and $Y \in \{\text{right, left}\}$. Then $L_{1r} \notin \mathcal{L}(j$-$Y$-mon-RLW).

**Proof.** Assume that there is an integer $i_0$ such that some $i_0$-right-mon-RLW-automaton $M$ with read/write window of size $k$ accepts the language $L_{1r}$. Let $2^n > p \cdot (k \cdot i_0 + 1)$, where $p$ is the constant from the Pumping Lemma (Fact 2.3), and let $w := a^{2^n}l \in L_{1r}$. Obviously, $w$ cannot be accepted directly by a tail computation. After the first cycle $C_1$ of an
accepting computation of $M$ on $w$, $M$ obtains a word $w_1$ of the form $w_1 = a^{2^n-2j_1}l_1a^{j_1}$, where $j_1 > 0$. Through the next cycle $C_2$ a word $w_2 = a^{2^n-2j_2}l_2a^{j_2}$ is obtained, where $j_1 < j_2$. We can continue in this way for $k \cdot i_0 + 1$ cycles. For each cycle $C_i$, $1 \leq i \leq k \cdot i_0 + 1$, we obtain a word $w_i := a^{2^n-2j}l_i$, where $j_1 < j_2 < \cdots < j_i$.

For each $i \geq 1$, the rewrite step performed in cycle $C_i$ increases the distance of the position of the symbol $l$ from the right sentinel from $j_i - 1$ to $j_i$, where we take $j_0 := 0$.

Thus, we obtain the following incomparability result.

Corollary 10.4. For all $j \in \mathbb{N}_+$, $X \in \{R, RR, RL\}$, and $Y \in \{right, left, right-left\}$,

$$L(det-XW) \subset L(j-Y-mon-RLW).$$

11. Concluding remarks

We close the paper by listing a number of problems that are related to the notions and results presented here, but that are still open:

- For the various types of restarting automata with auxiliary symbols, does the degree $j$ of right-monotonicity (left-monotonicity, right-left-monotonicity) yield infinite hierarchies?
- Is there a characterization of the language class $L(det-left-mon-RWW)$ in terms of more classical language families and operations?
- Can each context-free language be accepted by a deterministic RLWW-automaton?
- Are shrinking restarting automata in general more powerful than the corresponding (length-reducing) restarting automata of the same type? Or does Lemma 5.2 generalize to all types of (deterministic) restarting automata?

For completeness we include the following result which has been obtained recently, and which shows that Theorem 8.13 does not carry over to the case $j = 1$.

Theorem 11.1 (Jurdiński et al. [11,13]). For each $Y \in \{right, left, right-left\}$,


Appendix

Figs. 2–7 below depict the inclusion relations between the language classes that are defined by the various types of (right-, left-, right-left-) monotone restarting automata. Here $\longrightarrow$ denotes a proper inclusion, $\rightarrow$ denotes equality, and $\longrightarrow-$ denotes an inclusion, for which it is still open whether it is proper or not. The edge labels refer to the theorem or corollary, in which the corresponding equality or strict inclusion, respectively, is proved.
Fig. 2. Taxonomy of language classes accepted by deterministic $j$-right-monotone restarting automata.

Fig. 3. Taxonomy of language classes accepted by deterministic $j$-left-monotone restarting automata.
Fig. 4. Taxonomy of language classes accepted by deterministic $j$-right-left-monotone restarting automata.

Fig. 5. Taxonomy of language classes accepted by $j$-right-monotone restarting automata.
Fig. 6. Taxonomy of language classes accepted by $j$-left-monotone restarting automata.

Fig. 7. Taxonomy of language classes accepted by $j$-right-left-monotone restarting automata.
References


