Lookahead Hierarchies of Restarting Automata*

František Mráz

Charles University, Dept. of Computer Science, Malostranské nám. 25, 118 00
PRAHA 1, Czech Republic, e-mail: mraz@ksvi.ms.mff.cuni.cz

Abstract. Automata with a restart operation are a special kind of the linear bounded automaton. They can model syntax analysis by reduction which consists in stepwise simplification of an extended sentence so that the (in)correctness of the sentence is not affected. An automaton with a restart operation is a one-tape device with a finite control and a read/write-window of fixed length. We study several versions of automata with a restart operation and we show hierarchies with respect to the size of the read/write-window.

1 Introduction

The introduction of automata with a restart operation [2] was motivated by modelling *analysis by reduction* of natural language sentences in a similar way as in [6]. The analysis by reduction consists of stepwise simplification of an extended sentence so that the (in)correctness of the sentence is not affected. Thus, after some number of steps, a simple sentence is got or an error is found.

We illustrate analysis by reduction on the sentence

'Martin, Peter and Jane work very slowly.'

This sentence can be simplified, for example, as follows:

'Peter and Jane work very slowly.'
'Jane works very slowly.'
'Jane works slowly.'
'Jane works.'

Notice that every simplification is realized by deleting and possible rewriting of words. E.g., the sentence *'Jane works slowly.'* is obtained from the sentence *'Peter and Jane work slowly.'* by deleting *'Peter and'* and rewriting *'work'* by *'works'*.

Let us also demonstrate how to use reductions in a stepwise localization of a syntactic inconsistency on an example which is similar to the previous one:

'Martin, Peter and Jane works slowly.'
'Peter and Jane works slowly.'
'Peter and Jane works.'

* Supported by the Grant Agency of the Czech Republic, Grant-No. 201/99/0236

The last sentence is not reduced further; it constitutes a core of an error.

The given examples should illustrate that, for the purposes of the natural language processing, it is useful to model reductions of correct as well as incorrect sentences when we are interested in checking (syntactic) correctness.

$RRWW$-automata can serve as a natural framework for a formalization of analysis by reduction. Such an automaton can be roughly described as follows. It has a finite control unit, a head with a scanning window attached to a list (see Fig. 1), and it works in certain cycles. In a cycle, it moves the head from the left to the right along the word on the list; according to its instructions, it can, at some place, rewrite – once in a cycle – the contents of its lookahead by a shorter string, continue by scanning some further symbols, and finally "restart" – i.e., reset the control unit into the initial state and place the head on the left end of the shortened word. The computation in a cycle halts when the automaton enters an accepting or a rejecting state.

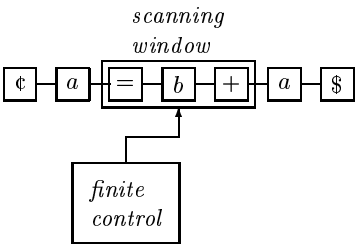

Fig. 1. An $RRWW$-automaton with the scanning window of size 3.

In a straightforward way, analysis by reduction can be modeled by the subclass of RRW-automata, which are the $RRWW$-automata using only the input alphabet symbols in rewriting. These automata enjoy an "error preserving property": if a word (sentence) is incorrect then the reduced word is also incorrect (after any number of cycles). In addition, deterministic RRW-automata have a "correctness preserving property": if a word is correct then the reduced word is correct as well. Similar properties for (general) $RRWW$-automata can not be stated so directly since also non-input symbols can be used for rewriting; nevertheless, this feature provides the power to recognize all context-free languages.

Besides RRW-automata, we introduce further subclasses of $RRWW$-automata. A taxonomy of (nondeterministic versions of) these automata is presented in [3].

The paper has several sections. In the next section we give definitions of all the studied classes of restarting automata and we present there some basic properties of these automata and some result from previous articles. Sect. 3 contains the main results – hierarchies of language classes recognized by restarting automata which does not use working symbols with respect to the size of their scanning window. Sect. 4 contains some final remarks concerning the (unsolved) problems about the restarting automata with working symbols.

2 Definitions and basic properties

In the first subsection we introduce $RRWW$-automata (nondeterministic and deterministic versions) and we show their basic properties. In the second subsection we summarize some results from other articles which we will use later.

By λ we denote the empty word, and by $\emptyset$ the empty set. $\subseteq$ denotes the inclusion relation and $\subset$ denotes the proper inclusion relation.

2.1 *RRWW*-automata

A *restarting automaton* is a tuple $M = (Q, \Sigma, \Gamma, k, I, q_0, Q_A, Q_R)$, the components of which are described in what follows. Q is a finite set of *states*, Σ and Γ are disjoint finite sets of symbols, called the *input alphabet* and the *work alphabet* respectively. We denote $V = \Sigma \cup \Gamma$, and suppose that V does not contain the special symbols $\mathbb{c}$, and \$ called the *left sentinel* and the *right sentinel* respectively. k is a positive integer called the *size of lookahead*, $q_0 \in Q$ is the *initial state*, $Q_A \subseteq Q$ is the set of *accepting states*, $Q_R \subseteq Q$ is the set of *rejecting states* ($Q_A \cap Q_R = \emptyset$). I is a finite set of *instructions* of the following three types (where $q, q' \in Q, a \in V \cup \{\mathbb{c}, \$\}$, $u \in V^* \cdot \{\lambda, \$\}$, $v \in \{\lambda, \mathbb{c}\} \cdot V^* \cdot \{\lambda, \$\}$, $k \geq |au| > |v| \geq 0$, s.t. either $|au| = k$, or $|au| \leq k$ and au ends by \$):

(1) $(q, au) \rightarrow (q', MVR)$
(2) $(q, au) \rightarrow (q', REWRITE(v))$
(3) $(q, au) \rightarrow RESTART$

The *restarting automaton* M is a device with a finite state control unit, and one head moving on a finite linear (doubly linked) list of items (cells). The first item always contains a special symbol $\mathbb{c}$, the last one contains another special symbol \$, and each other item contains a symbol from V. The head has a lookahead 'window' of length k ($k \geq 1$) – besides the current item, M also scans the next $k-1$ right neighbour items (or simply the end of the word when the distance to \$ is less than $k-1$).

We suppose that the set of states Q is divided into two classes – the set of *nonhalting states* $Q - (Q_A \cup Q_R)$ (there is at least one instruction which is applicable when the unit is in such a state) and the set of *halting states* $Q_A \cup Q_R$ (any computation finishes by entering such a state); the set of halting states is further divided into the set of accepting states Q_A and the set of rejecting states Q_R.

A *configuration* of the automaton M is (u, q, v), where $uv \in \{\mathbb{c}\}(\Sigma \cup \Gamma)^*\{\$\}$, u is the content of the list from the left sentinel to the position of the head (without the item under the head), $q \in Q$ is the current state and v is the content of the list from the scanned item to the right sentinel.

In the *restarting configuration* on a *word* $w \in (\Sigma \cup \Gamma)^*$, the word $\mathbb{c}w\$$ is stored in the items of the list, the control unit is in the initial state q_0, and the head is attached to that item which contains the left sentinel (scanning $\mathbb{c}$, looking also at the first $k-1$ symbols of the word w). An *initial computation* of M starts

in an *initial configuration* which is a restarting configuration on *an input word* ($w \in \Sigma^*$).

The *computation* of M is controlled by a finite set of instructions I of types (1), (2) and (3) from above. The left-hand side (q, au) of an instruction determines when it is applicable – q means the current state (of the control unit), a the symbol being scanned by the head, and u means the content of the lookahead window (u being a string of length $k-1$ or less if it ends with \$). The right-hand side describes the activity to be performed.

In case (1), M changes the current state to q' and moves the head to the right neighbour item of the item containing a. In particular, if $a = \$$ then q' must be a halting state.

In case (2), the activity consists of deleting (removing) some items (at least one) of the just scanned part of the list (containing au), and of rewriting some (possibly none) of the nondeleted scanned items (in other words au is replaced with v, where v must be shorter than au). After that, the head of M is moved to the right to the item containing the first symbol after the lookahead and the current state of M is changed to q'. There are two exceptions: if au ends by \$ then v also ends by \$ (the right sentinel cannot be deleted or rewritten) and after the rewriting the head is moved to the item containing \$; similarly, the left sentinel ¢ cannot be deleted or rewritten.

In case (3), $RESTART$ means entering the initial state and placing the head on the first item of the list (containing ¢).

Any computation of a restarting automaton M is composed of certain phases. A *phase* called *cycle* starts in a restarting configuration, the head moves to the right along the input list until a restart operation is performed, and M is resumed in a new restarting configuration. In this paper, we work with *$RRWW$-automata* which are restarting automata making exactly one $REWRITE$-instruction in each cycle – i.e., new phase starts on a shortened word. A phase of a computation called *tail* starts in a restarting configuration, the head moves to the right along the input list until one of the halting states is reached. M can also once rewrite during the tail.

It immediately implies that any computation of any $RRWW$-automaton is finite (finishing in a halting state).

In general, an $RRWW$-automaton is *nondeterministic*, i.e., there can be two or more instructions with the same left-hand side (q, au). If it is not the case, the automaton is *deterministic*.

An input *word w is accepted by M* if there is an initial computation which starts in the initial configuration with $w \in \Sigma^*$ (bounded by sentinels ¢,\$) on the list and finishes in an *accepting configuration* where the control unit is in one of the accepting states. $L(M)$ denotes the language consisting of all words accepted by M; we say that *M recognizes the language $L(M)$*.

By RRW-automata we mean $RRWW$-automata with empty working alphabet. RRW-automata use only the input symbols in the rewriting, i.e., in all instructions of the form (2) above, the string v is from Σ^*. Hence, all restarting configurations of an RRW-automaton are also its initial configurations.

By RR-automata we mean RRW-automata which use deleting without rewriting (i.e., in all instructions of the form (2) above, the string v can always be obtained by deleting some symbols from au).

By RWW-automata we mean $RRWW$-automata which do restart immediately after any $REWRITE$-instruction.

By RW-automata we mean RRW-automata which do restart immediately after any $REWRITE$-instruction.

By R-automata we mean RR-automata which do restart immediately after any $REWRITE$-instruction.

The notation $u \Rightarrow_M v$ means that there exists a cycle of M starting in the restarting configuration with the word u and finishing in the restarting configuration with the word v; the relation $\Rightarrow_M^*$ is the reflexive and transitive closure of $\Rightarrow_M$. If $u \Rightarrow_M v$ holds, we say that u is (can be) *directly reduced* to v by M. If $u \Rightarrow_M^* v$ holds, we say that u can be *reduced* to v by M.

In the proofs, we often use the next two propositions, two corollaries and one lemma. For their (simple) proofs see [4]:

Proposition 1 (The error preserving property). *Let $M = (Q, \Sigma, \Gamma, k, I, q_0, Q_A, Q_R)$ be an arbitrary $RRWW$-automaton, u, v be arbitrary input words from Σ^*. If $u \Rightarrow_M^* v$ and $u \notin L(M)$, then $v \notin L(M)$. [Equivalently, if $u \Rightarrow_M^* v$ and $v \in L(M)$, then $u \in L(M)$.]*

Corollary 2. *For RRW-automata (with $\Gamma = \emptyset$), $u \Rightarrow_M^* v$ and $v \in L(M)$ always implies $u \in L(M)$, since all its restarting configurations are initial.*

The next proposition is a useful complement of the previous one for deterministic $RRWW$-automata.

Proposition 3 (The correctness preserving property). *Let $M = (Q, \Sigma, \Gamma, k, I, q_0, Q_A, Q_R)$ be an arbitrary deterministic $RRWW$-automaton, u, v be arbitrary input words from Σ^*. If $u \Rightarrow_M^* v$ and $u \in L(M)$, then $v \in L(M)$.*

Corollary 4. *For deterministic RRW-automata (with $\Gamma = \emptyset$), $u \Rightarrow_M^* v$ and $u \in L(M)$ always implies $v \in L(M)$.*

Lemma 5. *Let $M = (Q, \Sigma, \Gamma, k, I, q_0, Q_A, Q_R)$ be an arbitrary $RRWW$-automaton, and $V = \Sigma \cup \Gamma$. There is a constant p such that for an arbitrary phase (cycle or tail) starting on a word $w_1 v w_2$ (for some words $w_1, v, w_2 \in V^*$), where $|v| \geq p$, the subword v can be written $v_1 a u v_2 a u v_3$ (for some words $v_1, u, v_2, v_3 \in V^*$ and symbol $a \in V$), $|u| = k$ and $|a u v_2| \leq p$ where both occurrences of a are entered in the same state by M during the given cycle or tail (including the possibility that both are not entered at all) and in the cycle or tail nothing in $a u v_2$ is deleted or rewritten.*

We will use the fact that for any $i \geq 0$ the given phase can be naturally extended for the 'pumped' word $w_1 v_1 (a u v_2)^i a u v_3 w_2$, where $i \geq 0$ (including also the case of removing – $i = 0$).

Next we introduce the monotonicity property of computations of $RRWW$-automata.

Any cycle Cyc performed by an $RRWW$-automaton M has a significant configuration $c_w = (\mathfrak{c}u, q, v\$)$ in which begins the rewriting step of the cycle Cyc. We call the number $|v|$ ($|v|$ means the length of the string v) the distance of the place of the rewriting in Cyc to the right sentinel and we denote it by $D_r(Cyc)$.

Let a computation $\mathcal{C}$ of an $RRWW$-automaton M consists of cycles Cyc_1, Cyc_2, ..., Cyc_n and a tail T. We say that the computation $\mathcal{C}$ is *monotonic* if the sequence $D_r(Cyc_1)$, $D_r(Cyc_2)$, ..., $D_r(Cyc_n)$ is not increasing, i.e., monotonic. Notice, that M can rewrite also during T, but the tails are not considered for the monotonicity.

By a *monotonic $RRWW$-automaton* we mean an $RRWW$-automaton for which all its computations (also the non-initial computations which can start in any restarting configuration) are monotonic.

The monotonicity of a given $RRWW$-automaton is effectively decidable (cf. [4]).

For brevity, prefix *det-* denotes the deterministic versions of $RRWW$-automata, similarly *mon-* the monotonic versions. $\mathcal{L}(\mathcal{A})$, where $\mathcal{A}$ is some class of automata, denotes the class of languages recognizable by automata from $\mathcal{A}$. E.g. the class of languages recognizable by deterministic monotonic R-automata is denoted by $\mathcal{L}(det\text{-}mon\text{-}R)$. A number in parentheses immediately following the type of a restarting automaton denotes the size of its scanning window – e.g. the class languages recognizable by deterministic RWW-automata with lookahead of at most k is denoted as $\mathcal{L}(det\text{-}RWW(k))$.

2.2 Some previous results about restating automata

The main results from [2], [3] and [4] are the characterizations of context-free languages (CFL) and deterministic context-free languages ($DCFL$) by restarting automata:

Theorem 6. *For any $X \in \{R, RR, RW, RRW, RWW, RRWW\}$ the following holds:*

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

Theorem 7. $CFL = \mathcal{L}(mon\text{-}RRWW) = \mathcal{L}(mon\text{-}RWW)$

In [2] there was given an *det-R*-automaton M_r recognizing the language

$$\begin{aligned} L(M_r) = &\{(ab)^{2^n-2m}(abb)^m \mid m, n \geq 0, 2m < 2^n\} \\ &\cup \{(abb)^{2^n-m}(ab)^m \mid m, n \geq 0, m < 2^n\} \end{aligned} \ .$$

Because

$$L(M_r) \cap \{(ab)^n \mid n \geq 1\} = \{(ab)^{2^k} \mid k \geq 0\} \tag{1}$$

the language $L(M_r)$ is non-context-free and cannot be recognized by a deterministic monotonic automaton (Theorem 6).

The automaton M_r from [2] works as follows:

1. reading ¢*abab* or ¢*abba* it moves to the right;
2. reading *ababa* or *babab* it moves to the right;
3. reading *abab*\$ it deletes second a and restarts;
4. reading *ababb* it deletes the first a and restarts;
5. reading *abbab* or *bbabb* or *babba* it moves to the right;
6. reading *babb*\$ it deletes the second b and restarts;
7. reading *bbab*\$ or *bbaba* it deletes the first b and restarts;
8. reading ¢*abb*\$ it deletes the first b and restarts;
9. reading ¢*ab*\$ it accepts;
10. in all other cases the automaton halts in a nonaccepting state.

In [2] (Theorem 4.2) there was shown, the following:

Theorem 8. $\mathcal{L}(mon\text{-}R) - \mathcal{L}(det\text{-}RW) \neq \emptyset$.

Thus $\mathcal{L}(mon\text{-}R) - DCFL \neq \emptyset$. There was shown that the language $L = \{a^i b^j \mid 0 \leq i \leq j \leq 2i\}$ is from $\mathcal{L}(mon\text{-}R) - \mathcal{L}(det\text{-}RW)$ and L can be recognized by a $mon\text{-}R(5)$-automaton. Actually, L can be recognized even by $mon\text{-}R(4)$-automaton which:

- immediately accepts the empty word,
- immediately rejects any nonempty word starting by b,
- on a nonempty word starting by a, moves to the right to this a. If its scanning window contains *ab*\$ (*abb*\$, resp.) then deletes *ab* (*abb*, resp.) and restarts. Otherwise M moves through a's to the right until its head scans a followed immediately by a different symbol. If its scanning window does not contain *abbb*, the word is rejected, else M nondeterministically deletes *ab* or *abb* and restarts.

Obviously M recognizes L.

In [4] there is proved the following.

Theorem 9. $CFL - \mathcal{L}(RRW) \neq \emptyset$.

Niemann and Otto in [5] studied the relation between the class of languages recognized by RWW-automata (they used a different notation) and the class of Church-Rosser languages (CRL). CRL is a class of languages defined by a string rewriting system. They have shown the following theorem.

Theorem 10. $\mathcal{L}(det\text{-}RWW) = CRL$.

3 Lookahead hierarchy

In [2, 4] there were shown some relations between CFL and the classes of languages accepted by restarting automata. In this section we will show that the lookahead is substantial in the concept of restarting automata and show that all considered classes of restarting automata without working symbols build proper

hierarchies according to the amount of allowed lookahead. The second considered question is the amount of lookahead needed to recognize a non-context-free language by a (nondeterministic) restarting automaton. Finally, we show that the obvious inclusion relations between $R(k)$, $RW(k)$, $RR(k)$ and $RRW(k)$ (Fig. 2) are proper inclusion relations, for all $k \geq 2$.

At first we show, that restarting automata which must restart immediately after rewriting and simultaneously with the scanning window of size 1 are weak. Let *Reg* denote the class of regular languages.

Lemma 11. $\mathcal{L}(R(1)) = \mathcal{L}(RW(1)) = \mathcal{L}(RWW(1)) = Reg$.

Proof. Obviously, any regular language can be recognized by an $R(1)$-automaton (even without use of the *RESTART*-operation). I.e., $Reg \subseteq \mathcal{L}(det\text{-}mon\text{-}R(1))$. It remains to show that $\mathcal{L}(RWW(1)) \subseteq Reg$. Let $M = (Q, \Sigma, \Gamma, 1, I, q_0, Q_A, Q_R)$ be an $RWW(1)$-automaton. We will show how to construct a finite state automaton F recognizing the language $L(F) = L(M)$. Note, please, that M can be considered to be an $R(1)$-automaton: in a *REWRITE*-instruction it must shorten the substring from its scanning window, but because its scanning window is of size 1, it must delete the scanned symbol. Obviously, it cannot rewrite any input symbol by a symbol from its working alphabet.

At first, we construct a finite state automaton F'. The states of F' are the subsets of Q. The automaton F' preserves the following invariant:

> *After reading a word u, the automaton F' is in a state $A \subseteq Q$ such that A consists of all the states in which M can be after several (possibly none) cycles on u and in a phase in which M leaves the part of the list originally containing u.*

The initial state of F' is the set A_0 of states which can enter M after moving to the right from the item containing the left sentinel, i.e. $A_0 = \{q' \in Q \mid (q_0, ¢) \to (q', MVR) \in I\}$.

The invariant is kept by the following transition function of F' defined for each set $A \subseteq Q$ and $x \in \Sigma$:

$$\delta_{F'}(A, x) = A_1 \cup A_2$$

where

$$A_1 = \{q' \in Q \mid \exists q \in A : (q, x) \to (q', MVR) \in I\}$$

and

$$A_2 = \begin{cases} A \text{ if } \exists q \in A : (q, x) \to (q', REWRITE(\lambda)) \in I \\ \emptyset \text{ if such } q \text{ does not exists} \end{cases}$$

which can be shown by induction on the length of u. A_1 contains the states which can be entered by M when it moves from x being in some state from A. A_2 contains states in which can be M when x was already deleted in some previous cycle. Note, please, that after performing the instruction $(q, x) \to (q', REWRITE(\lambda))$ the automaton M restarts as it is an RWW-automaton.

At second, we will show how to modify F' to get F. The finite state automaton F simulates F' and checks whether during the current computation, some accepting state (from Q_A) appeared in some state entered by F'. At that moment it enters a special accepting state in which it scans the rest of the input word. Obviously, F accepts a word w iff M accepts w. Thus $\mathcal{L}(RWW(1)) \subseteq Reg$. □

On the other hand, the restarting automata which need not to restart immediately after each rewriting (RR-, RRW- and $RRWW$-automata) are more powerful – even an $RR(1)$-automaton can recognize a non-context-free language. Such automaton can be constructed from any R-automaton with an arbitrary size of lookahead, but satisfying the condition that in each cycle it deletes exactly 1 symbol from the current word. This condition is satisfied by the R-automaton M_r from Sect. 2. An $RR(1)$-automaton M_{rr} recognizing the language $L(M_{rr}) = L(M_r)$ can be constructed so that it simulates M_r. M_{rr} in each step guesses the content of the lookahead of M_r, simulates the proper action and later checks whether the guess was correct. Even if M_r is deterministic, M_{rr} is nondeterministic. We omit the detailed construction.

Proposition 12. $\mathcal{L}(RR(1)) - CFL \neq \emptyset$.

Proof. See the above considerations. □

Lemma 13. $R(1) = RW(1) = RWW(1) = Reg \subset RR(1) = RRW(1) = RRWW(1)$.

Proof. The first three equations follow from Proposition 11. The proper inclusion follows from Proposition 12. The last two equalities follow from the fact, that in any rewriting the part in the scanning window should be shortened. As the size of the scanning windows is 1, the only way how to shorten the contents of the scanning window is to delete the symbol in the scanning window. □

Let D_1 denote the Dyck language ([1]), i.e., the language over the alphabet $\{a_1, \bar{a}_1\}$ generated by the following context-free grammar $G = (\{S\}, \{a_1, \bar{a}_1\}, S, P)$ with the set of rules P:

$$S \to a_1 S \bar{a}_1 \mid SS \mid \lambda$$

An alternative way to describe D_1 is to code a_1 as the left parenthesis and $\bar{a}_1$ as the right parenthesis. Then D_1 is the set of well-balanced parentheses. D_1 is a deterministic context-free language which is not regular. It can be recognized by a deterministic $R(2)$-automaton.

Proposition 14. $D_1 \in \mathcal{L}(det\text{-}mon\text{-}R(2))$.

Proof. An *det-mon-*$R(2)$ automaton recognizing D_1 simply accepts the empty word, and on a nonempty word, M scans the current word from left-to-right, deletes the first occurrence of $a_1\bar{a}_1$ and restarts. If the (nonempty) scanned word does not contain any subword $a_1\bar{a}_1$, then M rejects. Obviously, M is monotonic, deterministic and recognizes D_1. □

Lemma 15. *For each $k \geq 1$:*

$$\mathcal{L}(\textit{det-mon-}R(k+1)) - \mathcal{L}(RRW(k)) \neq \emptyset \ .$$

Proof. We will give a sequence of languages $\{L_k\}_{k=1}^{\infty}$ which satisfy

$$L_k \in \mathcal{L}(\textit{det-mon-}R(k+1)) - \mathcal{L}(RRW(k)) \ .$$

1. Let $L_1 = D_1$. According Proposition 14, $D_1 \in \mathcal{L}(\textit{det-mon-}R(2))$. Let D_1 be recognized by a $RRW(1)$-automaton M_1. Then, for a sufficiently large n (e.g. $n > p$, where p is the constant from Lemma 5) the word $a_1^n \bar{a}_1^n$ from D_1 cannot be accepted by M_1 in a tail (otherwise according to the remark after Lemma 5 we can construct an accepting tail for a word $a_1^{n+m}\bar{a}_1^n$, for some $m > 0$, which is not from D_1) In the first cycle of an accepting computation of M_1 on $a_1^n\bar{a}_1^n$ word must be shortened, but all words from D_1 shorter than $2n$ have length at most $2n-2$, but $RRW(1)$-automaton can shorten the word only by 1 symbol in one cycle – a contradiction. Thus, D_1 cannot be recognized by a $RRW(1)$-automaton.
2. Let $k \geq 2$ be an integer and $L_k = \{a^n c^{k-1} b^n \mid n \geq 0\}$. Then the language L_k cannot be recognized by an RRW-automaton with the size of the scanning window less or equal to k. Let M_k be a RRW-automaton recognizing L_k. For a sufficiently large n there exists an input word $a^n c^{k-1} b^n$ which cannot be accepted by M_k in a tail. Otherwise using Lemma 5 we could construct a word outside L which will be accepted by M_k. Thus in the first cycle of an accepting computation the word must be shortened, but the closest shorter word is $a^{n-1}c^{k-1}b^{n-1}$, which we can get by deleting the last a and the first b. This is possible for R-automata (even *det-mon-R*-automata) with lookahead at least $k+1$, hence $L_k \in \textit{det-mon-}R(k+1) - RRW(k)$.

□

Let us note that in the previous proof the language $L_1' = \{a^n b^n \mid n \geq 0\}$ cannot be used for separation of $\mathcal{L}(R(1))$ from $\mathcal{L}(R(2))$, $\mathcal{L}(RW(1))$ from $\mathcal{L}(RW(2))$, resp.. The reason is that $L_1' \notin \mathcal{L}(RW(2))$. This can be proved by a contradiction. Let us suppose that L_1 is recognized by an $RW(2)$-automaton M_1. The language is not regular, thus deleting is necessary – for a sufficiently large n there exists an input word $a^n b^n$ which cannot be accepted in a tail. Thus in the first cycle of an accepting computation the word must be shortened, but the closest shorter word is $a^{n-1}b^{n-1}$, which we can get by deleting the last a and the first b. Thus, $a^n b^n \Rightarrow_{M_1} a^{n-1}b^{n-1}$. Since M_1 is an $RW(2)$-automaton, it can make the same computation on the word $a^n bab^n \notin L_1$ and then $a^n bab^n \Rightarrow_{M_1} a^n b^n$, where $a^n b^n \in L_1$ – a contradiction with the error preserving property (Proposition 1).

The previous lemma has the following corollary.

Corollary 16. *Let $k \geq 1$ be an integer. Then, for any $X, Y \in \{R, RR, RW, RRW\}$, $pref_X, pref_Y \in \{\lambda, mon, det, det\text{-}mon\}$ the following holds:*

$$\mathcal{L}(pref_X\text{-}X(k)) \subset \mathcal{L}(pref_X\text{-}X(k+1)).$$

$$\mathcal{L}(pref_X\text{-}X(k+1)) - \mathcal{L}(pref_Y\text{-}Y(k)) \neq \emptyset \ .$$

(E.g. for $pref_X = mon, X = R$ the expression $pref_X$-X denotes mon-R, for $pref_X = \lambda, X = RRW$ the expression $pref_X$-X denotes RRW.)

Proof. For each $k \geq 1, X, Y \in \{R, RR, RW, RRW\}$, $pref_X, pref_Y \in \{\lambda,$ $mon,$ $det,$ $det\text{-}mon\}$ the following holds: trivially, $\mathcal{L}(pref_X\text{-}X(k)) \subseteq \mathcal{L}(pref_X\text{-}X(k+1))$, $(\mathcal{L}(det\text{-}mon\text{-}R(k+1)) - \mathcal{L}(RRW(k))) \subseteq (\mathcal{L}(pref_X\text{-}X(k+1)) - \mathcal{L}(pref_Y\text{-}Y(k)))$ and according Lemma 15 $\mathcal{L}(det\text{-}mon\text{-}R(k+1)) - \mathcal{L}(RRW(k)) \neq \emptyset$. □

Using Theorem 6 and Corollary 16 we get that the classes of languages recognized by deterministic and simultaneously monotonic versions of R-, RW- RR- and RRW-automata create proper hierarchies in $DCFL$, covering all $DCFL$.

Corollary 17. *For any $X \in \{R, RR, RW, RRW\}$, $pref \in \{\lambda, mon,$ $det\}$ the following holds:*

$$for\ each\ k \geq 1 : \mathcal{L}(det\text{-}mon\text{-}X(k)) \subset DCFL$$

$$for\ each\ k \geq 1 : DCFL - \mathcal{L}(pref\text{-}X(k)) \neq \emptyset$$

$$for\ each\ L \in DCFL\ there\ exists\ k \geq 1 : L \in pref\text{-}X(k) \ .$$

Using Theorem 7, Theorem 9 and Corollary 16 we get that the classes of languages recognized by monotonic versions of R-, RW- RR- and RRW-automata create proper hierarchies in CFL which do not cover all CFL.

Corollary 18. *For any $X \in \{R, RR, RW, RRW\}$, $pref \in \{\lambda, mon,$ $det\}$ the following holds:*

$$for\ each\ k \geq 1 : \mathcal{L}(mon\text{-}X(k)) \subset CFL$$

$$CFL - \mathcal{L}(pref\text{-}X) \neq \emptyset \ .$$

Increasing the size of the lookahead increases the recognition power of even R-automata with respect to the Chomsky hierarchy:

- $\mathcal{L}(R(2)) - Reg \neq \emptyset$ (Proposition 14),
- $\mathcal{L}(R(4)) - DCFL \neq \emptyset$ (see the remark after Theorem 8),
- $\mathcal{L}(R(5)) - CFL \neq \emptyset$ (see the automaton M_r from Sect. 2.2).

Next we will study the relations between the classes of languages recognized by restarting automata without working symbols with the size of scanning window fixed to $k \geq 1$. The case $k = 1$ is solved by Lemma 13. For $k \geq 2$, the obvious inclusion relations depicted in Fig. 2 are all proper inclusion relations. Moreover we will show that the classes $RW(k)$ and $RR(l)$, for any $k \geq 2$, $l \geq 1$ are incomparable.

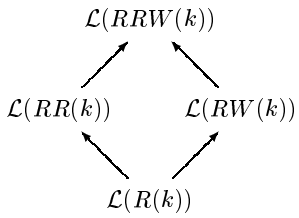

Fig. 2. Inclusion relations between $\mathcal{L}(R(k))$, $\mathcal{L}(RW(k))$, $\mathcal{L}(RR(k))$ and $\mathcal{L}(RRW(k))$, for $k \geq 1$. An arrow $A \longrightarrow B$ denotes $A \subseteq B$.

Lemma 19. $\mathcal{L}(RW(2)) - \mathcal{L}(RR) \neq \emptyset$.

Proof. Let $L_1 = \{c^n d^n, c^n f_1 d^n \mid n \geq 0\}$, $L_2 = \{c^n d^m, c^{n'} g_1 d^{m'}, c^n g_2 d^m \mid m > 2n \geq 0, m' > 2n'+1, n' \geq 0\}$ and $L = \{f, ee\} \cdot L_1 \cup \{g, ee\} \cdot L_2$. The language L can be recognized by a nondeterministic $RW(2)$-automaton M in the following way:

- M immediately accepts the word f, otherwise
- if the word starts by fc then M moves to the right until some symbol different from c appears in its scanning window, then:
  - seeing cd it rewrites it to f_1 and restarts,
  - seeing cf_1 it moves one symbol to the right,
  - seeing $f_1 d$ it rewrites it to d and restarts,
  - otherwise it rejects.
- if the word starts by gc then M moves to the right until some symbol different from c appears in its scanning window, then:
  - seeing cd it rewrites it to g_1 and restarts,
  - seeing cg_1 it moves one symbol to the right,
  - seeing $g_1 d$ it rewrites it to g_2 and restarts,
  - seeing cg_2 it moves one symbol to the right,
  - seeing $g_2 d$ it rewrites it to d and restarts,
  - otherwise it rejects.
- if the word starts by gd or gg_1 or gg_2 then M scans the rest of the word. If it contains only d's (at least one) then accepts otherwise rejects,
- if the word starts by ee then M nondeterministically rewrites ee by f or g and restarts.

It is easy to verify that $L(M) = L$.

L cannot be recognized by any RR-automaton. For a contradiction let us suppose $L = L(M)$ for some RR-automaton M with lookahead of length k. Let us choose (and fix) a sufficiently large n ($n > k$) s.t. n is divisible by $p!$ (and hence by all $p_1 \leq p$) where p is taken from Lemma 5. Now consider the first cycle C of an accepting computation of M on $eec^n d^n$. M can only shorten both segments of c's and d's in the same way, i.e. $eec^n d^n \Rightarrow_M eec^l d^l$, for some $l < n$. (Any accepting computation of M on $eec^n d^n$ has at least two cycles – otherwise

using Lemma 5 we can construct a word outside L which will be accepted by M in one cycle). Due to Lemma 5, d^n can be written $d^n = v_1auv_2auv_3$, $a = d$, $u = d^k$, $|auv_2| \leq p$, where M in the cycle C enters both occurrences of a in the same state.

Recall that nothing is deleted out of auv_2 in C and $|auv_2|$ divides n due to our choice of n. Then there is some i s.t. $d^{2n} = v_1(auv_2)^i auv_3$; hence $eec^n d^{2n} \notin L(M)$ but (by the natural extending of the cycle C) we surely get $eec^n d^{2n} \Rightarrow_M eec^l d^{n+l}$, where $2l < n + l$ and therefore $eec^l d^{n+l} \in \{ee\} \cdot L_2 \subseteq L(M)$ – a contradiction with the error preserving property (Proposition 1).

□

Because $\mathcal{L}(R) \subseteq \mathcal{L}(RR)$ and $\mathcal{L}(RW) \subseteq \mathcal{L}(RRW)$, from the above lemma it follows that $\mathcal{L}(RW(2)) - \mathcal{L}(R) \neq \emptyset$ and $\mathcal{L}(RRW(2)) - \mathcal{L}(RR) \neq \emptyset$. Thus we have the following corollary.

Corollary 20. *For each $k \geq 2$ the following holds:*

$$\mathcal{L}(R(k)) \subset \mathcal{L}(RW(k))$$
$$\mathcal{L}(RR(k)) \subset \mathcal{L}(RRW(k)) \ .$$

Next we will give a language which is recognized by a $RR(1)$-automaton but it cannot be recognized by any RW-automaton.

Lemma 21. $\mathcal{L}(RR(1)) - \mathcal{L}(RW) \neq \emptyset$.

Proof. Let $L = \{a^i b^j c^m \mid 0 \leq i \leq j \leq m \text{ and } m - i \leq 1\}$. Using standard pumping lemma for context-free languages it could be shown that L is not context-free language. We will show that $L \in \mathcal{L}(RR(1)) - \mathcal{L}(RW)$.

a) L can be recognized by an $RR(1)$-automaton M in the following way:
   - M immediately accepts the empty word;
   - on a nonempty word it works according to Table 1. M nondeterministically selects one row of the table and during cycle it scans the current word from the left to the right, checks that the word is of the form $a^*b^*c^*$, counts the parity of the numbers of a's, b's, c's and deletes one occurrence of the symbol from the middle column of the selected row of Table 1. At the right sentinel M knows $\overline{i}$ – the parity of the number of a's, $\overline{j}$ – the parity of the number of b's, $\overline{m}$ – the parity of the number of c's in the word at the start of the phase. If $\overline{i}$, $\overline{j}$, $\overline{m}$ equal to the three items of the first column of the selected row in the table, then M restarts, otherwise it rejects. M rejects the current word also when the word is not of the form $a^*b^*c^*$ and in the case M could not delete the symbol from the second column, i.e. the current list does not contain item with the symbol which should be deleted.

   For any two words $u, w \in \{a, b, c\}^*$, such that $u \Rightarrow_M v$, the following holds:

$$u \in L \quad \text{iff} \quad v \in L \ .$$

   I.e., the automaton M has the correctness preserving property although it is nondeterministic automaton (cf. Proposition 3).

original parities			symbol	resulted parities		
$\overline{i}$	$\overline{j}$	$\overline{m}$	to delete	$\overline{i'}$	$\overline{j'}$	$\overline{m'}$
0	0	0	a	1	0	0
1	0	0	b	1	1	0
1	1	0	c	1	1	1
1	1	1	a	0	1	1
0	1	1	b	0	0	1
0	0	1	c	0	0	0

Table 1.

b) L cannot be recognized by any RW-automaton. For a contradiction, let us suppose $L = L(M')$, for some RW(k)-automaton M', $k \geq 1$. For a sufficiently large n ($n > p$, where p is the constant from Lemma 5) M' must make at least one cycle in an accepting computation on the word $w = a^n b^n c^n \in L$ – otherwise using Lemma 5 we can construct a word outside L which will be accepted by M' in a tail (e.g. by 'pumping' in a's). W.l.o.g. we can suppose n being greater than k. Thus, $a^n b^n c^n \Rightarrow_{M'} a^i b^j c^m$, for some $0 < i \leq j \leq m$, $m - i \leq 1$, $i + j + m < 3n$. Clearly, $i < n$ and during the rewriting step, the scanning window must contain at least one a and the automaton restarted immediately after the rewriting (M' is an RW(k)-automaton). Hence, the last b did not appear in the lookahead of M' during the corresponding cycle. This implies $m = n$ and $i = n - 1$ (otherwise $a^i b^j c^m \notin L$). M' can make the same steps on the words $w_1 = a^n b^{n-1} c^{n-1}$ and $w_2 = a^n b^n c^{n-1}$. I.e.

$$w_1 = a^n b^{n-1} c^{n-1} \Rightarrow_{M'} a^i b^{j-1} c^{n-1} = w_3 \tag{2}$$

and

$$w_2 = a^n b^n c^{n-1} \Rightarrow_{M'} a^i b^j c^{n-1} = w_4 \tag{3}$$

There are two possibilities for the value of j. If $j = n$ then (2) contradicts the error preserving property (Proposition 1) – $w_1 \notin L, w_3 \in L$ and $w_1 \Rightarrow_{M'} w_3$. Else, $j = n-1$ then (3) contradicts the error preserving property (Proposition 1) – $w_2 \notin L, w_4 \in L$ and $w_2 \Rightarrow_{M'} w_4$.

□

Because $\mathcal{L}(R) \subseteq \mathcal{L}(RW)$ and $\mathcal{L}(RR) \subseteq \mathcal{L}(RRW)$, from the above lemma it follows that $\mathcal{L}(RR(1)) - \mathcal{L}(R) \neq \emptyset$ and $\mathcal{L}(RRW(1)) - \mathcal{L}(RW) \neq \emptyset$. Thus we have the following corollary.

Corollary 22. *For each $l \geq 1$ the following holds:*

$$\mathcal{L}(R(l)) \subset \mathcal{L}(RR(l))$$
$$\mathcal{L}(RW(l)) \subset \mathcal{L}(RRW(l)) \ .$$

From Lemma 19, Lemma 21, Corollary 20 and Corollary 22 it follows:

Corollary 23. *For each $k \geq 2$, $l \geq 1$ the classes $\mathcal{L}(RW(k))$ and $\mathcal{L}(RR(l))$ are incomparable by inclusion.*

4 Conclusions

We have shown that increasing the size of lookahead for any of R-, RW-, RR-, and RRW-automata increases their recognition power. This is important e.g. for natural language processing. I believe that for a given natural language there exists a fixed size of lookahead which is necessary and sufficient for its recognition by an RW- or RRW-automaton. We have not studied the question whether such hierarchies are also in the classes of languages recognized by restarting automata which use auxiliary (working) symbols (RWW-, and $RRWW$-automata). In [4] there was shown that $\mathcal{L}(mon\text{-}RWW) = CFL$. Actually, there was presented a construction which for a given context-free language constructs a $mon\text{-}RWW(3)$-automaton recognizing it. From this follows that the classes of languages recognized by monotonic RWW-automata ($RRWW$-automata resp.) does not create infinite hierarchy with respect to the size of their scanning window.

Even the open problem whether $\mathcal{L}(RWW) = \mathcal{L}(RRWW)$ from [3] remains still unsolved.

References

1. M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, 1st edition, 1978.
2. P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On restarting automata with rewriting. In G. Păun and A. Salomaa, editors, *New Trends in Formal Language Theory (Control, Cooperation and Combinatorics)*, volume 1218 of *LNCS*, pages 119–136. Springer, 1997.
3. P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Different types of monotonicity for restarting automata. In *FST&TCS'98*, volume 1530 of *LNCS*, pages 343–354. Springer, 1998.
4. P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics*, 4(4):287–311, 1999.
5. G. Niemann and F. Otto. Restarting automata, church-rosser languages, and representations of r.e. languages. In W. Thomas, editor, *Developments in Language Theory, 1999*, pages 49–62, Aachen, Germany, July 1999. Fachgruppe Informatik der RWTH.
6. M. Novotný. *With Algebra from Language to Grammar and back (S algebrou od jazyka ke gramatice a zpět).* Academia, Praha, 1988. *In Czech.*