On Restarting Automata with Rewriting*

Petr Jančar†, František Mráz‡, Martin Plátek‡, Jörg Vogel§

July 4, 2000

Abstract

Motivated by natural language analysis we introduce restarting automata with rewriting. They are acceptors on the one hand, and (special) regulated rewriting systems on the other hand. The computation of a restarting automaton proceeds in cycles: in each cycle, a bounded substring of the input word is rewritten by a shorter string, and the computation restarts on the arising shorter word.

We show a taxonomy of (sub)variants of these automata taking into account (non)determinism and two other natural properties.

Theoretical significance of the restarting automata is also demonstrated by relating it to context-free languages (*CFL*), by which a characterization of deterministic *CFL* is obtained.

1 Introduction

Our motivation for introducing the restarting automata is to model so called elementary syntactic analysis of natural languages. The elementary syntactic analysis consists in stepwise simplification of an extended sentence until a simple sentence is got or an error is found. Let us show it on the sentence

'Martin, Peter and Jane work very slowly.'

We work with the wordforms in the examples; instead of wordforms, the elementary analysis uses their lexical characterizations (categories). This sentence can be simplified for example in this way:

*Supported by the Grant Agency of the Czech Republic, Grant-No. 201/96/0195

†University of Ostrava, Department of Computer Science, Bráfova 7, 701 03 OSTRAVA, Czech Republic, e-mail: jancar@osu.cz

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

§Friedrich Schiller University, Computer Science Institute, 07740 JENA, Germany, e-mail: vogel@informatik.uni-jena.de

'Martin, Peter and Jane work slowly.'

'Martin, Peter and Jane work.'

'Martin **and** *Peter work.'* or *'Peter and Jane work.'* or *'Martin and Jane work.'*

'Martin **works**.*'* or some other variant of the corresponding simple sentence.

Notice, that every simplification is realized by deleting and possible rewriting (marked by the bold face) of words.

The restarting automaton with rewriting (RW-automaton), introduced in [7], can be roughly described as follows. It has a finite control unit, a head with a lookahead window attached to a linear (doubly linked) list with sentinels, and it works in certain cycles. In a cycle, it moves the head from left to right along the word on the list (any item contains exactly one symbol); according to its instructions, it can at some point replace the scanned string by a shorter string and "restart" – i.e. reset the control unit to the initial state and place the head on the left end of the list (which now contains the shortened word). The computation halts in an accepting or a rejecting state.

Using cycles we define *yield relation* for a RW-automaton M. From a string α contained in the list by the start of a cycle M yields a string β remaining in the list after finishing the cycle (denoted by $\alpha \Rightarrow_M \beta$). Together with the yield relation, a RW-automaton can also be considered as a (regulated) rewriting system.

Formerly, in [4], we have introduced restarting automata without rewriting – R-automata – by which the replacing string is a proper subsequence of the replaced string. They are considered as a transparent model for grammar checker (of natural and formal languages as well). Having found an error in a sentence, the grammar checker should specify it – often by exhibiting the parts (words or more exactly their lexical characterizations) which do not match each other. The method can be based on stepwise leaving out some parts not affecting the (non)correctness of the sentence. E.g. applying it to the sentence

'The little boys I mentioned runs very quickly'

we get after some steps the "error core"

'boys runs'.

There are other paradigms modelling the elementary syntax in the generative way, e.g. *pure (generalized) grammars with strictly length-increasing rules* (c.f. [8]). These grammars work on strings of terminals and do not introduce any nonterminals. This type of grammars realize, similarly as Marcus grammars ([6]), a (generalized) rewriting system with an yield relation $\Rightarrow_G$ ($\alpha \Rightarrow_G \beta$ means that α can be rewritten to β in one step according to a grammar G),

which should capture the stepwise development from simple sentences. The yield relation has the so called *correctness preserving property*, which means that if α is a correct string according to some grammar G and $\alpha \Rightarrow_G \beta$, then β is a correct string again.

The yield relation corresponding to a RW-automaton has a dual property comparing to the yield relation corresponding to a pure (generalized) grammar with strictly length-increasing rules. RW-automaton yields strings in the strictly length-decreasing way, and has the *error preserving property*: if α contains error (is not in the language recognized by the RW-automaton M) and $\alpha \Rightarrow_M \beta$, then β contains an error. This duality will allow to consider RW-automata as another type of generalization of pure grammars with strictly length-increasing rules. The RW-automata allow to add regulation of the yielding by their control units.

Section 2 contains definitions of RW-automata and R-automata. There are also described some basic properties of these automata and their computations. As usual, we define nondeterministic and deterministic versions of the automata. In second subsection we show that RW-automata are stronger than pure grammars. In the third subsection we consider a natural property of monotonicity (during any computation, "the places of restarting do not increase their distances from the right end") and show that the monotonicity is a decidable property. In Section 3 we show that monotonic RW-automata recognize a subset of the class of context-free languages (CFL) and deterministic monotonic RW-automata recognize only deterministic CFL ($DCFL$). Moreover any deterministic context-free language can be recognized by a deterministic monotonic R-automaton. From this results we get two characterizations of $DCFL$. The paper continues in Section 4 with separation theorems for rewriting and nonrewriting classes of automata and some related results. In conclusions (Section 5) beside discussion of future directions of our study, also another type of automata with similar features as RW-automata – contraction automata – is mentioned.

2 Definitions and Basic Properties

We present the definitions informally; the formal technical details could be added in a standard way of the automata theory. In the first subsection we introduce restarting automata with rewriting, in the second subsection we relate RW-automata to pure grammars and in the last subsection we introduce the monotonicity property for RW-automata and a normal form of this automata – (strong) cyclic form.

2.1 Restarting automata with rewriting

A *restarting automaton with rewriting*, or a *RW-automaton*, M (with bounded lookahead) 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 ¢, the last one another special symbol \$, and each other item contains a symbol from a finite alphabet (not containing ¢, \$). The head has a lookahead "window"of length k (for some $k \geq 0$) – besides the current item, M also scans the next k right neighbour items (or simply the end of the word when the distance to \$ is less than k). In the *initial configuration*, the control unit is in a fixed, initial, state and the head is attached to the item with the left sentinel ¢ (scanning also the first k symbols of the input word).

The *computation* of M is controlled by a finite set of *instructions* of the following two types:

(1) $(q, au) \rightarrow_M (q', MVR)$

(2) $(q, au) \rightarrow_M RESTART(v)$

The left-hand side 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 contents of the lookahead window (u being a string of length k 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. In case (2), au is replaced with v, where v must be shorter than au, and M restarts – i.e. it enters the initial state and places the head on the first item of the list (containing ¢).

We say that M is a *restarting automaton (R-automaton)* if in each instruction of the form $(q, au) \rightarrow_M RESTART(v)$ the word v is a proper subsequence of the word au.

We will suppose that the control unit states of M are divided into two groups: the *regulating states* (nonhalting states – an instruction is always applicable when the unit is in such a state) and the *halting states* (a computation finishes by entering such a state); the *halting states* are further divided into the *accepting states* and the *rejecting states*.

In general, a *RW*-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* (*det-RW*-automaton).

An input *word w is accepted by* M if there is a computation which starts in the initial configuration with w (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* $L(M)$.

It is natural to divide any computation of a *RW*-automaton into *cycles*: in one cycle, the head moves right along the input list (with a bounded lookahead)

until a halting state is entered or something in a bounded space is rewritten – in that case the computation is resumed in the initial configuration on the shortened word (thus a new cycle starts). It immediately implies that any computation of any RW-automaton is finite (finishing in a halting state).

The notation $u \Rightarrow_M v$ means that there exists a cycle of M starting in the initial configuration with the word u and finishing in the initial configuration with the word v; the relation $\Rightarrow_M^*$ is the reflexive and transitive closure of $\Rightarrow_M$. We say that u *yields* v by M if $u \Rightarrow_M v$.

The next three claims express the basic properties of the yield relations corresponding to RW-automata.

Claim 2.1 (The error preserving property (for all RW-automata)) *Let M be a RW-automaton, and $u \Rightarrow_M^* v$ for some words u, v. If $u \notin L(M)$, then $v \notin L(M)$.*

Proof: Let $u \Rightarrow_M^* v$ such that $v \in L(M)$. Since $v \in L(M)$ there is some y such that $v \Rightarrow_M^* y$, where y can be accepted by M in one cycle. Because of $u \Rightarrow_M^* v$, the relation $u \Rightarrow_M^* y$ holds. Hence u is accepted by M. □

Claim 2.2 (The correctness preserving property (for *det*-RW-automata)) *Let M be a deterministic RW-automaton and $u \Rightarrow_M^* v$ for some words u, v. If $u \in L(M)$, then $v \in L(M)$.*

Proof: Let M be a deterministic RW-automaton, $u \in L(M)$ and $u \Rightarrow_M^* v$ for some word v. Because of determinism of M there exists exactly one accepting computation for u by M. The computation represented by sequence of cycles $u \Rightarrow_M^* v$ is a prefix of this computation. Thus the rest of this computation (starting in the initial configuration with v on the list) is an accepting computation for v and $v \in L(M)$. □

As a consequence of the previous two claims 2.1 and 2.2 we get:

Claim 2.3 *Let M be a deterministic R-automaton and $u \Rightarrow_M^* v$ for some words u, v. Then $v \in L(M)$ if and only if $u \in L(M)$.*

2.2 Pure grammars

Next we will show that RW-automata can be considered as regulated acceptors (analysers) for pure grammars with strictly length-increasing rules.

A *pure grammar* is a triple $G = (V, P, S)$ where V is a finite set of symbols, S is a finite subset of V^*, and P is finite set of productions of the form $v \rightarrow_G w$, $v, w \in V^*$.

For $x, y \in V^*$, the yield relation $x \Rightarrow_G y$ is defined by $x = z_1 v z_2$, $y = z_1 w z_2$, $v \rightarrow_G w \in P$, where $v, w, z_1, z_2 \in V^*$.

$\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$. The language generated by G is defined as $L(G) = \{y \mid x \Rightarrow_G^* y \text{ for some } x \in S\}$.

We can easily see that $\Rightarrow_G^*$ has the correctness preserving property, i.e. if $x \in L(G)$ and $x \Rightarrow_G^* y$ then $y \in L(G)$.

We say that a grammar G has strictly length-increasing rules if $|v| < |w|$ for each $v \rightarrow_G w \in P$.

We can easily see that for any pure grammar G with strictly length-increasing rules there is a RW-automaton M with one regulating state (the starting state), one rejecting and one accepting state only, such that $L(G) = L(M)$, and the relation $\Rightarrow_G$ is the opposite relation to $\Rightarrow_M$.

The opposite implication is not true. Let us show that the RW-automata with one regulating state are more powerful than pure grammars.

Let us take the following language $L_a = \{a^n b^n \mid n \geq 0\} \cup \{a^n \mid n \geq 0\}$.

Claim 2.4 *The language L_a cannot be generated by any pure grammar G with strictly length-increasing rules.*

Proof: Let us suppose that some pure grammar $G = (V, P, S)$ with strictly length-increasing rules generates L_a.

Let us consider a sufficiently long word $w = a^m$, where m is greater than the size of any string from S. We can see that there is $v = a^n$ such that $v \Rightarrow_G w$, and therefore P contains a rule of the form $a^p \rightarrow_G a^{p+q}$, where $p \geq 0$ and $q > 0$.

Let us consider word $z = a^m b^m$. We can see that $z \in L_a$ and $z \Rightarrow_G z'$, where $z' = a^{m+q} b^m$. Since $m + q > m > 0$ the word z' is not in L_a, that is a contradiction to the correctness preserving property of $\Rightarrow_G$. □

Claim 2.5 *There is a RW-automaton M with one regulating state recognizing the language L_a.*

Proof: Let us describe the automaton M: M has lookahead of the length 3, one regulating state (the initial state q_0), the accepting state q_a and the rejecting state q_r. The automaton in one cycle accepts the empty word, deletes a from the words containing only a's, deletes ab from the word ab and deletes ab from the words with the prefix a^+bb. The working alphabet of M is $\{a, b\}$ and M has the following instructions:

$(q_0, ¢\$) \rightarrow_M (q_a, MVR)$,
$(q_0, ¢a\$) \rightarrow_M RESTART(¢\$)$,
$(q_0, ¢ab\$) \rightarrow_M RESTART(¢\$)$,
$(q_0, ¢aa\$) \rightarrow_M RESTART(¢a\$)$,
$(q_0, ¢aaa) \rightarrow_M (q_0, MVR)$,
$(q_0, ¢aab) \rightarrow_M (q_0, MVR)$,
$(q_0, aaaa) \rightarrow_M (q_0, MVR)$,
$(q_0, aaab) \rightarrow_M (q_0, MVR)$,
$(q_0, aabb) \rightarrow_M RESTART(ab)$,
$(q_0, aaa\$) \rightarrow_M RESTART(aa\$)$,
$(q_0, u) \rightarrow_M (q_r, MVR)$ for any u of the length four not covered in the previous cases.

We can see that M recognizes L_a and that M is actually a deterministic R-automaton. □

2.3 Monotonicity and cyclic forms of RW-automata

In this subsection the monotonicity property of RW-automata is defined, its decidability is shown and so called cyclic forms of restarting automata with rewriting are introduced.

The property of monotonicity (for a computation of a RW-automaton): All items which appeared in the lookahead window (and were not deleted) during one cycle will appear in the lookahead in the next cycle as well – if it does not finish in a halting state (i.e., during a computation, "the places of changes in the list do not increase their distances from the right endmarker \$").

By a *monotonic RW-automaton* we mean a RW-automaton where the property of monotonicity holds for all computations.

Theorem 2.6 *There is an algorithm which for any RW-automaton M decides whether M is monotonic or not.*

Proof: We sketch the idea briefly. Consider a given (nondeterministic) RW-automaton M; let its lookahead be of length k. Recall that all computations of a monotonic automaton have to be monotonic. The idea is to construct a (nondeterministic) finite automaton which accepts a nonempty language if and only if there is a nonmonotonic computation of M.

Suppose there is a nonmonotonic computation of M. Then there is a word w on which M can perform two cycles where in the second cycle it does not scan all (remaining) items scanned in the first cycle.

Now consider the construction of the mentioned finite automaton A; we can suppose that it has lookahead of length k. A supposes reading the described w. It moves right simulating two consecutive cycles of M simultaneously. At a certain moment, A decides nondeterministically that it has entered the area of rewriting in the second cycle – it guesses the appropriate contents of the lookahead window which would be encountered in the second cycle. Then it moves right coming to the place of the (guessed) rewriting in the first cycle and verifies that the previously guessed lookahead was guessed correctly; if so, A accepts. □

Considering a deterministic RW-automaton M, it is sometimes convenient to suppose it in the *strong cyclic form*; it means that the words of length less than k, k being the length of lookahead, are immediately (hence in the first cycle) accepted or rejected, and that M performs at least two cycles (at least one restarting) for any longer word.

For a nondeterministic RW-automaton M, we can suppose the *weak cyclic form* – any word from $L(M)$ longer than k (the length of lookahead) can be

accepted only by performing two cycles at least. The cyclic forms are justified by the following claim.

Claim 2.7 *For any RW-automaton (R-automaton) M, with lookahead k, there exists an RW-automaton (R-automaton) M', with some lookahead n, $n \geq k$, such that M' is in the weak cyclic form and $L(M) = L(M')$. Moreover, if M is deterministic then M' is deterministic and in the strong cyclic form, if M is monotonic deterministic then M' is monotonic deterministic and in strong cyclic form.*

Proof: To prove this claim we can proceed in the same way as for R-automata (see [5]).

First notice that we can easily force an RW-automaton M to visit all items of the input list before accepting or rejecting – instead of an "original" accepting (rejecting) state, it would enter a special state which causes moving to the right end and then accepting (rejecting).

Now suppose that (the modified) M accepts a long word w in the first cycle (without restarting). If w is sufficiently long then it surely can be written $w = v_1auv_2auv_3$ where M enters both occurrences of a in the same state q during the corresponding computation (as above, by a we mean a symbol and by u a string of length k). Then it is clear that the word v_1auv_3 (auv_2 has been deleted) is also accepted. In addition, we can suppose that the length of auv_2auv_3 is less than a fixed (sufficiently large) n.

We sketch a desired M' with lookahead n. Any word w shorter than n is immediately accepted or rejected by M' according to whether $w \in L(M)$ or not. On a longer w, M' simulates M with the following exception: when \$ appears in the lookahead window, M' checks whether M could move to the right end and accept; if so, M' deletes the relevant auv_2 (cf. the above notation) and restarts (recall that n has been chosen so that such auv_2 surely exists). Obviously, $L(M) = L(M')$ holds.

In case M is deterministic, M' can work as above; in addition it can safely delete the relevant auv_2 also when M would reject (due to determinism, the resulting word is also rejected by M).

It should be clear that monotonicity of M implies monotonicity of M' in the deterministic case.

Further it should be clear that we get by this construction from a R-automaton M a R-automaton M'. □

Remark: *For the nondeterministic monotonic RW-automata the construction does not ensure monotonicity of the resulted automaton.*

For brevity, we use the following obvious notation. RW denotes the class of all (nondeterministic) restarting automata (with rewriting and some lookahead). R denotes the class of all (nondeterministic) restarting automata without rewriting. Prefix *det-* denotes the deterministic version, similarly *mon-* the monotonic

version. For any class $\mathcal{A}$ of automata, $\mathcal{L}(\mathcal{A})$ denotes the class of languages recognizable by automata from $\mathcal{A}$, and $\mathcal{A}$-language is a language from $\mathcal{L}(\mathcal{A})$. E.g. the class of languages recognizable by deterministic monotonic R-automata is denoted by $\mathcal{L}(det\text{-}mon\text{-}R)$.

Throughout the article we will use the following notations for the inclusion relations: $A \subseteq B$ means that A is a subset of B and $A \subset B$ means that A is a proper subset of B ($A \subseteq B$ and $A \neq B$). $\emptyset$ denotes the empty set.

3 Characterization of *DCFL*

In this section, we show a (twofold) characterization of $DCFL$, namely $DCFL = det\text{-}mon\text{-}R = det\text{-}mon\text{-}RW$. In addition, we also get $mon\text{-}RW \subset CFL$.

Lemma 3.1 $DCFL \subseteq \mathcal{L}(det\text{-}mon\text{-}R)$

Proof: We use the characterization of deterministic context-free languages by means of $LR(0)$–grammars and $LR(0)$–analysers (cf. e.g. [2]); generally $LR(1)$–grammars (lookahead 1) are needed, but it is not necessary when each word is finished by the special sentinel \$.

Let L' be a deterministic context-free language. Then there is an $LR(0)$–grammar G generating $L = L'\$$ (the concatenation $L' \cdot \{\$\}$ supposing \$ being not in the alphabet of L'), and there is a corresponding $LR(0)$–analyser P.

For any word $w \in L$ there is only one derivation tree T_w; it corresponds to the analysis of w by the analyser P. In fact, P simulates constructing T_w in the left-to-right and bottom-up fashion. Due to the standard pumping lemma for context-free languages, there are constants p, q s.t. for any w with length greater than p there are a (complete) subtree T_1 of T_w and a (complete) subtree T_2 of T_1 with the same root labelling; in addition, T_2 has fewer leaves than T_1 and T_1 has q leaves at most. (Cf. Fig. 1; A is a nonterminal of G). Replacing T_1 with T_2, we get the derivation tree for a shorter word w' (w could be written $w = u_1v_1u_2v_2u_3$ in such a way that $w' = u_1u_2u_3$).

Now we outline a $det\text{-}mon\text{-}R$-automaton M with lookahead of length $k > q$ which recognizes L'.

M stores the contents of the lookahead in a buffer in the control unit. Simulating the $LR(0)$–analyser P, it constructs (in a bounded space in the control unit) all maximal subtrees of the derivation tree which have all their leaves in the buffer. If one of the subtrees is like the T_1 above, M performs the relevant deleting (of at most two continuous segments) in the input list and restarts. If it is not the case then M forgets the leftmost of these subtrees with all its $n \geq 1$ leaves, and reads n new symbols to the right end of the buffer (shifting the contents left). Then M continues constructing the maximal subtrees with all leaves in the (updated) buffer (simulating P).

In general, the input word w is either shorter than p, such words can be checked using finite memory, or it is longer. If it is longer and belongs to L then

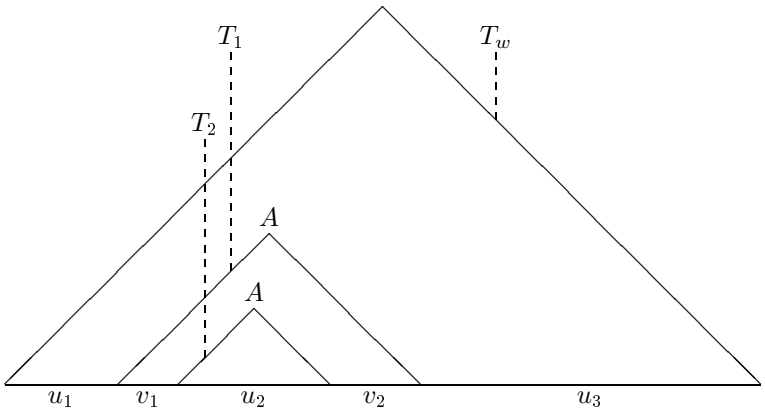

Figure 1:

M must meet the leftmost above described T_1 (with the subtree T_2); it performs the relevant deleting and restarts on a new, shorter, word. If the (long) input word w does not belong to L, M either meets \$ without restarting and stops in a rejecting state or performs some deleting and restarts. It suffices to show that the resulting shorter word (in both cases) is in L if and only if w is in L.

It can be verified using the following properties of the $LR(0)$–analyser.

a) For each word $w \in L$ there is exactly one derivation of w in G which corresponds to the analysis of w by P.

b) Let u be the prefix of the input word $w = uv$ which has been already read by the $LR(0)$–analyser P. If P did not reject the word until now, then there exists a suffix word v' s.t. uv' is in L, and the computation of P on the prefix u is independent w.r.t. the suffix.

Let u be the prefix of the input word, the last symbol of which corresponds to the last symbol in the lookahead window just before M performs a *RESTART*-operation; let $\overline{u}$ be the rest of u after performing the *RESTART*-operation. There exists a suffix v' such that uv' is in L. uv' has a derivation tree in which there is a complete subtree T_1 (with a subtree T_2; as above) corresponding to the place of cutting. Then in the computation of M on $\overline{u}$ the tree T_2 will appear in the buffer above the same terminal leaves as in the computation on u (it follows from the presence of T_2 in the derivation tree of $\overline{u}v'$ and from the independence of computation on the suffix).

Let $w = uv$ is in L, then obviously $\overline{u}v$ is in L. Conversely if uv is not in L then $\overline{u}v$ is not in L (otherwise, in the corresponding derivation tree of $\overline{u}v$, the

subtree T_2 appears over the corresponding terminal leaves of $\overline{u}$ and replacing the tree T_2 by T_1 yields a derivation tree for uv – a contradiction).

The monotonicity of M should be clear from the above description. □

Lemma 3.1 (together with Claim 2.7) is a means for short proving that some languages are not in *DCFL*. We illustrate it by the next two examples.

Example 3.2 Consider the language $L_1 = \{ww^R \mid w \in \{a,b\}^*\}$. If it were in *DCFL*, it would be recognized by a deterministic R-automaton M with the length of lookahead k (for some k); M can be supposed in the strong cyclic form. Let us now take a word $a^n b^m b^m a^n$ $(n, m > k)$, on which M performs two cycles at least. In the first cycle, M can only shorten the segment of $b's$. But due to determinism, it would behave in the same way on the word $a^n b^m b^m a^n a^n b^m b^m a^n$, which is a contradiction to the correctness preserving property (Claim 2.2).

Example 3.3 The language $L_2 = \{a^m b^n c^p \mid m, n, p \geq 0 : (m = n \text{ or } n = p)\}$ is not in *DCFL*. This can be shown in a similar way as in the previous example. But using the error preserving property for RW-automata we will show stronger result – the language L_2 cannot be recognized by any RW-automaton.

The next claim shows that RW-automata do not recognize all context-free languages and will be used in the proof of the next lemma.

Claim 3.4 *The language* $L_2 = \{a^m b^n c^p \mid m, n, p \geq 0 : (m = n \text{ or } n = p)\}$ *is not a* RW*-language.*

Proof: Let us suppose that L_2 is recognized by some RW-automaton M_2 in the weak cyclic form with the size of lookahead k. Let us consider an accepting computation on a word $a^r b^r$, where $r > 2(k+1)$. In the first cycle of the accepting computation, M_2 can only shorten both segments of a's and b's in such way that after the first cycle the resulting word will be $a^{r-m} b^{r-m}$ for some $0 < m < k$ (i.e. the restart with rewriting occurs in the middle of the word; M_2 cannot rewrite a suffix of the word in order to get a word of the form $a^r b^m c^m$ for some $m \geq 0$). But M_2 can behave in the same way also on the word $a^r b^{r+m} c^r \notin L_2$ from which it can get after the first cycle $a^{r-m} b^r c^r \in L_2$. This is a contradiction to the error preserving property of RW-automata (Claim 2.1). □

Lemma 3.5 *a)* $\mathcal{L}(mon\text{-}RW) \subset CFL$,

b) $\mathcal{L}(det\text{-}mon\text{-}RW) \subseteq \mathcal{L}(DCFL)$.

Proof: At first we show that $\mathcal{L}(mon\text{-}RW)$ is a subclass of *CFL*. Let L be a language recognized by *mon*-RW-automaton M, with lookahead of length k. We show how to construct a pushdown automaton P which simulates M.

The construction is similar to the construction of a deterministic pushdown automaton which should simulate a given *det-mon-R*-automaton (see [4]). The difference is following: Instead of one step simulation of a single *MVR*-step, P will simulate in one step all possible "nondeterministic" *MVR*-steps performed on the simulated scanned item simultaneously, and the simulation of a restart without rewriting is replaced by a simulation of a restart with rewriting.

P is able to store in its control unit in a component CSt the set of possible current states of M (i.e. any subset of the set of states of M) and in a component B a word of length at most $1 + 2k$. P starts by storing $\{q_0\}$, where q_0 is the initial state of M, in CSt and pushing $\not\vdash$ (the left endmarker of M) into the first cell of the buffer B and the first k symbols of the input word of M into the next k cells of the buffer B (cells $2, 3, \ldots, k + 1$).

During the simulation, the following conditions will hold invariantly:
– CSt contains the set of all states of M, in which can be M visiting the simulated (currently scanned) item, with the current left-hand side, and the current lookahead,
– the first cell of B contains the current symbol of M (scanned by the head) and the rest of B contains m right neighbour symbols of the current one (lookahead of length m) where m varies between k and $2k$,
– the pushdown contains the left-hand side (w.r.t. the head) of the list, the leftmost symbol ($\not\vdash$) being at the bottom. In fact, any pushdown symbol will be composed – it will contain the relevant symbol of the input list and the set of states of M in which this symbol (this item) could be entered (from the left) by the situation, which corresponds to the last simulated visit.

The mentioned invariant will be maintained by the following simulation of instructions of M; the left-hand side (q, au) of the instruction to be simulated is determined by the information stored in the control unit. The activity to be performed depends on the right-hand sides of applicable instructions of M. P can

either (1) nondeterministically simulate one of $RESTART$ instructions of M,

or (2) simulate all possible MVR instructions in one step.

(1) $RESTART(v)$ is simulated by deleting and rewriting in the buffer B (some of the first $k + 1$ symbols are deleted and the rest is pushed to the left and possibly rewritten). Then $k+1$ (composed) symbols are successively taken from the pushdown and the relevant symbols are added from the left to B (shifting the rest to the right). The state parts of k (composed) symbols are forgotten, the state part of the $(k + 1)$-th symbol (the leftmost in the buffer) is stored in CSt. Thus not only the $RESTART(v)$ - operation is simulated but also the beginning part of the next cycle, the part which was prepared in the previous cycle.

(2) P puts the contents of the first cell of B and CSt as a composed symbol on the top of the pushdown, stores the set $\{q' \mid (q, au) \rightarrow_M (q', MVR), q \in CSt\}$

of simulated new states which can be entered after MVR-step from some state in the original CSt with the lookahead au, and shifts the contents of B one symbol to the left; if the $(k+1)$-th cell of B is then empty, then P reads the next input symbol into it.

It should be clear that due to monotonicity of M the second half of B (cells $k+2$,k+3,...,$2k$) is empty at the time of simulating a $RESTART(v)$–operation. Hence the described construction is correct which proves $\mathcal{L}(mon\text{-}RW) \subseteq CFL$. To finish the proof of the first part of the proposition can show a context-free language which cannot be recognized by any RW-automaton. But this was already done in Claim 3.4.

Obviously the above construction applied to a *det-mon-R*-automaton yields a deterministic push-down automaton – this proves part b) of the statement. □

Theorem 3.6 $\mathcal{L}(det\text{-}mon\text{-}RW) = DCFL = \mathcal{L}(det\text{-}mon\text{-}R)$

Proof: The statement is a consequence of Lemma 3.1, Lemma 3.5 and the trivial inclusion $\mathcal{L}(det\text{-}mon\text{-}R) \subseteq \mathcal{L}(det\text{-}mon\text{-}RW)$. □

It can be worth noting that the closure of deterministic RW-languages under complement is immediately clear when considering deterministic RW-automata (*det-R-*, *det-RW-*, *det-mon-R-* and *det-mon-RW*-automata). Since all computations of deterministic RW-automata are finite it suffices to exchange the accepting and the rejecting states to get a deterministic automaton of the same type (*det-R-*, *det-RW-*, *det-mon-R-* or *det-mon-RW*-automaton) recognizing the complementary language.

Claim 3.7 *The classes of languages* $\mathcal{L}(det\text{-}mon\text{-}R)$, $\mathcal{L}(det\text{-}mon\text{-}RW)$, $\mathcal{L}(det\text{-}R)$ *and* $\mathcal{L}(det\text{-}RW)$ *are closed under complement.*

4 Taxonomy of RW-languages

In this section we will study relations between different subclasses of $\mathcal{L}(RW)$. The resulting relations are depicted in Figure 2.

Next we will prove all the relations depicted in this figure. We will start by proving that a *det-R*-automaton can recognize a language which is not *mon-RW*-language.

Theorem 4.1 $\mathcal{L}(det\text{-}R) - \mathcal{L}(mon\text{-}RW) \neq \emptyset$

Proof: To prove the theorem, it is sufficient to give a *det-R*-automaton M which recognizes a non context-free language $L = L(M)$. L cannot be recognized by any *mon-RW*-automaton, because according Lemma 3.5 all languages recognized by *mon-R*-automata are context-free. We will use a *det-R*-automaton M which recognizes a non context-free language from [5].

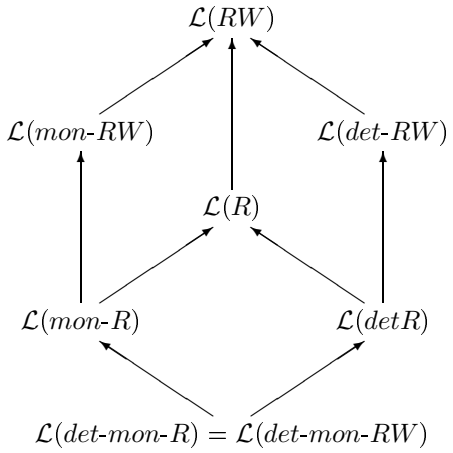

Figure 2: Taxonomy of RW-languages. Solid arrows show the proper inclusion relations, depicted classes not connected (by an oriented path) are incomparable.

The main idea is to start with a non context-free language $L' = \{a^{2^k} \mid k \geq 0\}$. The automaton M will work in phases. A phase starts with a word from L' on the list and consists of several cycles in which the length of the current word is reduced by factor 2 and simultaneously the parity of the length of the word on the start of the phase is checked. But restarting automaton can shorten the word by at most constant number of symbols in one cycle. Thus we must modify the language to enable to "mark" already shortened part of the working list – instead of one symbol a we use a pair ab. Working on a word of the form $(ab)^n$ for some $n > 1$, M at first deletes the second a out of each subword $abab$, proceeding stepwise from the right to the left. Then M deletes one b out of each abb and proceeds also stepwise from the right to the left. The automaton recognizes the language $L(M)$ for which

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

Thus the language $L(M)$ is not context-free and also is not a $mon\text{-}RW$-language (Lemma 3.5).

The automaton M works as follows:

1. reading $\mathfrak{c}abab$ or $\mathfrak{c}abba$ 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.

Clearly the automaton M is deterministic. To prove (1) let us consider a word of the form $(ab)^n$ for some $n \geq 1$. Then the following two conditions hold

$$\text{(i)} \quad (ab)^n \Rightarrow_M^* (ab)^{\frac{n}{2}} \qquad \text{when } n \text{ is even,}$$

$$\text{or (ii)} \quad (ab)^n \Rightarrow_M^* b(abb)^{\frac{n-1}{2}} \qquad \text{when } n \text{ is odd.}$$

In the case (i) the automaton makes $\frac{n}{2}$ cycles (using points 1, 2–4), and gets the word $(abb)^{\frac{n}{2}}$, then after another $\frac{n}{2}$ cycles it gets the word $(ab)^{\frac{n}{2}}$ (using points 1, 5–8)

In the case (ii) the automaton makes $\frac{n-1}{2}$ cycles (using points 1, 2–4), and gets the word $b(abb)^{\frac{n-1}{2}}$, which will be rejected in the next cycle.

Thus let the input word be of the form $(ab)^n$, where $n = l2^k$ for some integer $k \geq 0$ and some odd integer $l \geq 1$.

- If n is a power of 2 (i.e. $l = 1$ and $k \geq 1$), then according (i) $(ab)^n \Rightarrow_M^* (ab)^{2^{k-1}} \Rightarrow_M^* (ab)^{2^{k-2}} \Rightarrow_M^* \ldots \Rightarrow_M^* ab$ and according point 9 the input word will be accepted.
- If n is not a power of 2 (i.e. $l > 1$), then according (i) $(ab)^n \Rightarrow_M^* (ab)^l$, $l > 1$ and odd. Further according (ii) $(ab)^l \Rightarrow_M^* b(abb)^{\frac{l-1}{2}}$ and this word will be rejected by 10.

□

This theorem implies some relations between several classes of languages from Figure 2. These relations are depicted in Figure 3. Solid arrow from A to B means that $A \subset B$, dotted arrow from A to B means that $B - A \neq \emptyset$ (and $A \subset B$ is still not excluded).

The next theorem shows a symmetric statement to the previous theorem.

Theorem 4.2 $\mathcal{L}(mon\text{-}R) - \mathcal{L}(det\text{-}RW) \neq \emptyset$

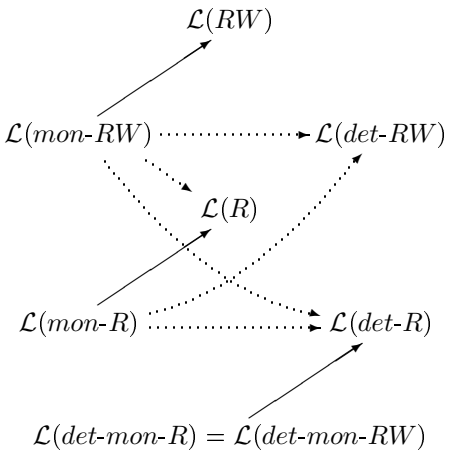

Figure 3: Relations which follow from the Theorem 4.1. Solid arrows depict proper inclusion relations, dotted arrows depict non-inclusion relations (in the following shown as incomparable by inclusion) – dotted arrow from A to B means $A \not\supseteq B$.

Proof: We will show that the language

$$L = \{a^i b^j \mid 0 \leq i \leq j \leq 2i\}$$

is a *mon-R*-language and is not a *det-RW*-language.

L is recognized by a (nondeterministic) *mon-R*-automaton M with lookahead 4 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 the lookahead contains $b\$$ ($bb\$$, $abb\$$, resp.) then deletes ab (abb, $aabb$ 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 lookahead is not $bbbb$ or $bbb\$$, the word is rejected, else M nondeterministically deletes ab or abb and restarts.

Obviously M recognizes L.

On the other hand the language L cannot be accepted by a deterministic *RW*-automaton (in the strong cyclic form). Working on the word $a^n b^n$ for sufficiently large n (greater than lookahead of the automaton) this deterministic automaton should shorten the word in a cycle by keeping the correctness

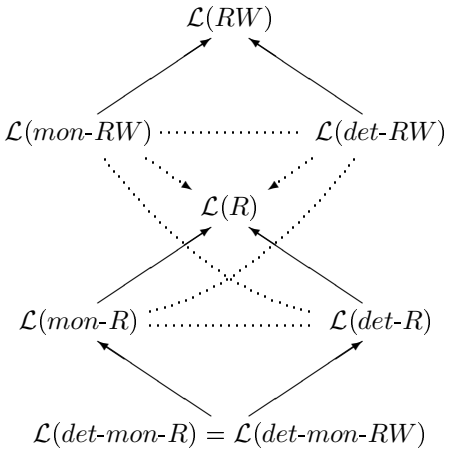

Figure 4: Relations which follow from Theorem 4.1 and Theorem 4.2. Dotted arcs (not the dotted arrows) denote already proved incomparability relations.

preserving property. Thus $a^n b^n \Rightarrow_M a^r b^s$ for some $r < n$ and $s \geq r$, i.e the automaton must decrease the number of a's in the word, thus rewriting can occur only when the head is visiting some a in the word. Because of determinism and fixed size of lookahead the automaton must work in the same way on the word $a^n b^{2n}$. But the resulting word at the end of the first cycle is $a^r b^{s+n}$, where $s + n > 2r$, which is not in L. This is a contradiction to the correctness preserving property of *det-RW*-automata (Claim 2.2). □

This theorem implies new proper inclusion relations and non-inclusion relations. We compose them with the relations depicted in Figure 3. In the resulting Figure 4 we use the same notation as in Figure 3 except that the incomparability relations which follow from the already proved theorems are depicted by dotted arcs.

As a consequence of the next theorem and Theorem 4.2 we get the incomparability of classes $\mathcal{L}(det\text{-}RW)$ and $\mathcal{L}(R)$.

Theorem 4.3 $\mathcal{L}(det\text{-}RW) - \mathcal{L}(R) \neq \emptyset$

Proof: We will construct a deterministic RW-automaton M such, that $L(M)$ cannot be recognized by any R-automaton. The idea of the construction is similar to that one used in the proof of Theorem 4.1. We are going out of the language $\{a^{3^n} \mid n \geq 0\}$. The automaton M will work in phases. One phase consists of several cycles. During a phase the working word will be reduced by factor 3 and its length will be checked whether it is divisible by 3. A marking

of the already reduced part of the word is done by a single symbol b in the list. Let us describe the automaton M: M has lookahead of the length 3, and three states only – the initial state q_0, the accepting state q_a and the rejecting state q_r. The working alphabet of M is $\{a, b\}$ and M has the following instructions:

1. $(q_0, ¢a\$) \rightarrow_M (q_a, MVR)$,
2. $(q_0, ¢ba\$) \rightarrow_M RESTART(¢a\$)$,
3. $(q_0, ¢baa) \rightarrow_M RESTART(¢aa)$,
4. $(q_0, ¢aaa) \rightarrow_M (q_0, MVR)$,
5. $(q_0, aaaa) \rightarrow_M (q_0, MVR)$,
6. $(q_0, aaab) \rightarrow_M RESTART(ba)$,
7. $(q_0, aaa\$) \rightarrow_M RESTART(ba\$)$,
8. $(q_0, u) \rightarrow_M (q_r, MVR)$ for any other u of the length four not covered in the previous cases.

Actually

$$L(M) \cap \{a^i \mid i \geq 1\} = \{a^{3^n} \mid n \geq 0\} \tag{2}$$

i.e. an intersection of $L(M)$ and the regular language a^* is a non context-free language, thus $L(M)$ cannot be from CFL. We can show that:

$$a^{3i+j} \Rightarrow_M^* a^j b a^i \text{ for } i > 0 \text{ and } 2 \geq j \geq 0$$

The automaton M makes cycles (using instructions $4 - 7$) until the symbol b appears in the lookahead window in the initial configuration of a cycle. If it is the first symbol after ¢ then this b is deleted (according 3 or 2). Otherwise the current word is rejected. From this the observation (2) directly follows.

$L(M)$ cannot be recognized by any R-automaton: Suppose that an R-automaton M_R in the weak cyclic form recognizes $L(M)$ and the length of its lookahead is k. Then for a sufficiently large m (e.g. greater than k) such automaton accepts the word a^{3^m} and $a^{3^m} \Rightarrow_{M_R} a^l$ in the first cycle of an accepting computation on the word a^{3^m}. But l cannot be a power of 3 ($3^{m-1} < 3^m - k - 1 \leq l < 3^m$). This fact contradicts the error preserving property (Claim 2.1). □

Moreover the previous proof shows, that RW-automata are stronger than R-automata outside CFL.

As a consequence of the next theorem and Theorem 4.1 we get the incomparability of classes $\mathcal{L}(mon\text{-}RW)$ and $\mathcal{L}(R)$.

Theorem 4.4 $\mathcal{L}(mon\text{-}RW) - \mathcal{L}(R) \neq \emptyset$.

Proof: Obviously $\mathcal{L}(R)$ is a subclass of $\mathcal{L}(RW)$.

At first, we will show that the language

$$L_c = \{ww^R \mid w \in \{a, b\}^*\} \cup \{wcw^R \mid w \in \{a, b\}^*\}$$

can be recognized by a $mon\text{-}RW$-automaton M_c:

1. M_c immediately accepts the empty word and the word c.

2. Working on a word (of length greater than 1) containing one symbol c the automaton M_c can scan the word until this symbol and to check whether it is surrounded by the same symbols (a or b). If it is so, then M_c deletes the left and right neighbour of c and restarts, otherwise rejects.

3. Working on a word without c the automaton can "guess aa or bb in the center" of the word, replace it by c and restart.

4. M_c is nondeterministic, but when it makes a mistake – inserts c in a word already containing c or inserts c not in its center, then the test according the point 2 above will fail later.

M_c can be constructed in such a way that the following properties holds:

(i_1) $xaay \Rightarrow_{M_c} xcy$, for any words $x, y \in \{a, b\}^*$.

(i_2) $xbby \Rightarrow_{M_c} xcy$, for any words $x, y \in \{a, b\}^*$.

(i_3) $xacay \Rightarrow_{M_c} xcy$, for any words $x, y \in \{a, b\}^*$.

(i_4) $xbcby \Rightarrow_{M_c} xcy$, for any words $x, y \in \{a, b\}^*$.

(i_5) M_c accepts the one-symbol-word c immediately.

(i_6) M_c rejects in a cycle any word of the form cy or yc, where y is any nonempty word.

(i_7) Any cycle performed by M_c is one of the types $i_1, .., i_6$.

Secondly, we will show that L_c cannot be recognized by any R-automaton by a contradiction. W.l.o.g. let us suppose that L_c is recognized by some R-automaton M in the weak cyclic form. Let us consider an accepting computation on a sufficiently long word $a^n b^m b^m a^n$, where m, n are greater than the size of lookahead of M. In the first cycle of the accepting computation, M can only shorten the segment of b's. We will get a word of the form $a^n b^{2m'} a^n$, for some $m' < m$, after the first cycle. But M can make the same first cycle in the computation on the word $a^n b^{2m} a^n a^n b^{2m'} a^n$, which is not in $L(M)$ and get the word $a^n b^{2m'} a^n a^n b^{2m'} a^n$ which is from $L(M)$. This is a contradiction to the error preserving property of RW-automata (cf. Claim 2.1; R-automata are a special type of RW-automata). □

The language L_c used in the previous proof is a context-free language. Thus we have proved that RW-automata are stronger than R-automata even inside CFL.

The last two theorems imply the relations depicted in Figure 5.

Composing Figure 4 and Figure 5 we get the complete picture in Figure 2.

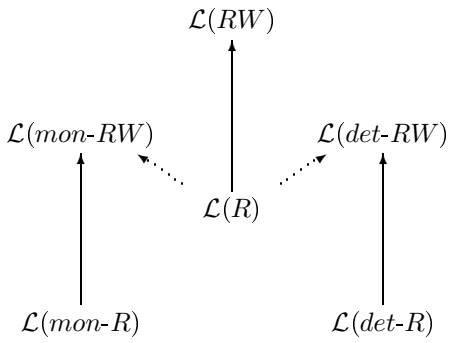

Figure 5: Relations which follow from Theorem 4.3 and Theorem 4.4.

5 Conclusions

In the previous sections we have shown typical results concerning RW-languages and R-languages. We have compared deterministic, nondeterministic, monotonic, nonmonotonic, rewriting and nonrewriting RW-languages. The nondeterministic RW-languages can be possibly studied in a similar way by (regulated) generative tools. On the other hand the results concerning the deterministic RW-languages can be hardly achieved by generative tools in a natural way.

Considering RW-automata as analysers we are interested in properties of the corresponding yield relations as well. There is exactly one difference between the presented taxonomy of languages and the corresponding taxonomy of yield relations. The class of yield relations corresponding to *det-mon-RW*-automata is greater than the class of yield relations corresponding to *det-mon-R*-automata. We have omitted the part about the taxonomy of yield relations here, because the nonequality results can be obtained by simple observations of finite yield relations (with finite number of pairs $u \Rightarrow v$).

In [4], we have introduced restarting automata as a further model of list automata (in particular forgetting automata (see [3])). Later, by reading the book by Dassow and Paun (see [1]) we have met contraction automata (introduced by von Solms in [9]). The contraction automata have (some) similar features as RW-automata. A contraction automaton works as a restricted linear bounded automaton. It simulates the operation deleting using a special symbol, and works in cycles. Any cycle starts on the right sentinel, and uses one reversal on the left sentinel. To any contraction automaton M, a complexity (n, k) is associated where, roughly speaking, n means the number of non-input symbols which can be used by M, and k is the maximal number of changes, which can be performed on the tape during a cycle. The contraction automata work also with some technical restrictions, which allow characterizations of matrix languages,

certain class of random context languages and a type of ETOL languages.

Inspired by contraction automata we propose to study measures of regulation of (generalized) RW-automata in the future. Also we propose to compare (generalized) RW-automata with different types of regulated generative tools. Some such steps been already done for (generalized) R-automata.

We plan to take RW-automata as theoretical background for a program for (robust) syntactic analysis of Czech sentences. That can be a nice tool to learn the basic syntax of the Czech language.

References

[1] J. Dassow, G. Păun: *Regulated Rewriting in Formal Language Theory*; EATCS Monographs on Theoretical Computer Science, Springer Verlag, 1989

[2] J. Hopcroft, J. Ullman: *Introduction to Automata Theory, Languages, and Computation*; Addison-Wesley, 1979

[3] P. Jančar, F. Mráz, M. Plátek: *A Taxonomy of Forgetting Automata*; in: *Proc. of MFCS 1993*; Gdańsk, Poland, LNCS 711, Springer Verlag, 1993, pp. 527–536

[4] P. Jančar, F. Mráz, M. Plátek, J. Vogel: *Restarting Automata*; in: *Proc. of FCT'95*; Dresden, Germany, LNCS 965, Springer Verlag, 1995, pp. 283–292

[5] P. Jančar, F. Mráz, M. Plátek, M. Procházka, J. Vogel: *Restarting Automata, Marcus Grammars and Context-Free Languages*; in: J. Dassow, G. Rozenberg, A. Salomaa (eds.): *Developments in Language Theory*; World Scientific Publ., 1995, pp. 102–111

[6] S. Marcus: *Contextual grammars*; Revue Roum. Math. Pures Appl.,14, 10, 1969, pp.1525–1534

[7] F.Mráz, M.Plátek, J.Vogel: *Restarting Automata with Rewriting*; in *Proc. of SOFSEM'96*; LNCS 1175, Springer Verlag 1996, to appear

[8] M.Novotný: *S algebrou od jazyka ke gramatice a zpět*; Academia, Praha, 1988, (*in Czech*)

[9] S.H. von Solms: *The Characterization by Automata of Certain Classes of Languages in the Context Sensitive Area*; Inform. Control 27, 1975, pp. 262–271