On Special Forms of Restarting Automata *

F. Mráz (mraz@ksvi.ms.mff.cuni.cz), M. Plátek
(platek@ksi.ms.mff.cuni.cz) and M. Procházka
Charles University, Department of Computer Science,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

Abstract. We study transformations of automata from some (sub)classes of restarting automata ($RRWW$-automata) into two types of special forms. We stress particularly the transformations into the linguistically motivated weak cyclic form. Special forms of the second type express a certain degree of determinism of such automata.

Keywords: restarting automaton, weak cyclic form, monotonicity

1. Introduction

This paper is a continuation of the paper (Plátek, 1999), which studied transformations into weak cyclic forms of automata from some subclasses of RWW-automata. These (sub)classes constitute a part of the taxonomy of restarting automata ($RRWW$-automata) presented in (Jančar et al., 1998).

The restarting automata serve as a formal tool to model analysis by reduction in natural languages. This means that restarting automata serve similarly as Marcus contextual grammars (Marcus, 1997; Păun, 1997) to cover (and to understand) the part of syntax which defines well- and ill-formedness of natural language sentences, but which is not (yet) burdened with the explicit effort to describe the relation between form and meaning (semantics). That is practically relevant e.g. in development of grammar-checkers (Kuboň et al., 1994), and in splitting the complex task to develop a parser of a natural language into several well defined (described) simpler tasks.

In this paper we concentrate on the property of a *weak cyclic form* which naturally corresponds to analysis by reduction for natural languages.

Let us give a closer look on the linguistic roots of our motivations. Analysis by reduction consists in 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. Let us illustrate it on the reduction of the sentence

* Supported by the Grant Agency of the Czech Republic, Grant-No. 201/99/0236, and by Grant Agency of Charles University, Grant-No. 157/1999/A INF/MFF

'Martin, Peter and Jane work very slowly.'

The sentence can be simplified for example in the following way:

'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 *'Peter works.'* or *'Jane works.'*

The last three sentences are all the simple sentences which can be reduced from the original sentence. We can see that each of them consist of two words only. The property of weak cyclic form for restarting automata corresponds to the general belief that for any natural language there exists an upper bound of the length of its simple sentences.

A restarting automaton (of the type of $RRWW$-automaton) can be roughly described as follows. It has a finite control unit, a head with a lookahead window attached to a list, and it works in certain phases. In each phase, it moves the head from left to right along the word on the list; according to its instructions, it can at some point once rewrite the contents of its lookahead by a shorter string and (possibly) move to the right-hand side again. At a certain place, according to its instructions it can 'restart' – i.e., reset the control unit to the initial state and place its head on the left end of the shortened word. The phase ending by restart is called a cycle. The computation ends in a phase by halting in an accepting or a rejecting state – such a phase is called a tail. A restarting automaton M is in the weak cyclic form if the length of any word accepted by M using a tail only (without restart) is not greater than the size of its lookahead.

Several classes of restarting automata were considered in (Jančar et al., 1998), the most general is that of $RRWW$-automata. For such automata, the property of *monotonicity of computations* has been introduced, which allows to derive three different definitions of *monotonicity of automata*; we call them monotonicity, a-monotonicity and g-monotonicity. The properties of monotonicity allow to characterize the class of context-free languages, and different types of their analysis by reduction.

Section 2 defines the $RRWW$-automata, their subclasses (*R-, RR-, RW-, RRW-,* RWW-automata), the mentioned three types of monotonicity properties as well as the weak cyclic form and three degrees of (non)determinism. Some basic properties of restarting automata are mentioned there.

The results are presented in Section 3. The transformation into weak cyclic form of RWW-automata and their subclasses (together with their monotonic, a-monotonic and g-monotonic versions) was studied in (Plátek, 1999). Here we show that the weak cyclic form can be constructed for any *RR*-, *RRW*- or *RRWW*-automaton and for any of its monotonic, a-monotonic and g-monotonic versions. We show further that the situation can be a bit more complex if we combine the transformation into weak cyclic form with a transformation into some other special forms of *RRWW*-automata, based on certain degree of determinism.

2. Definitions and basic properties

We present the definitions informally; the formal technical details can be added in a standard way of the automata theory. λ denotes the empty word, and $\emptyset$ denotes the empty set.

A *restarting automaton* $M = (Q, \Sigma, \Gamma, k, I, q_0, Q_A, Q_R)$ is a device with a finite state control unit, with the (finite) set of states Q, where $(Q_A \cup Q_R) \subseteq Q$, and with 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 the union of two disjoint finite alphabets Σ and Γ (not containing ¢, \$). Σ is called *the input alphabet*, Γ *the working alphabet*. The head has a lookahead 'window' of the length k ($k > 0$) – 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 $k-1$). M contains a fixed *initial* state q_0, ($q_0 \in Q$).

A *configuration* of the automaton M is (u, q, v), where $uv \in \{¢\}(\Sigma \cup \Gamma)^*\{\$\}$, 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 ¢$w$\$ 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 ¢, looking also at the first $k-1$ symbols of the word w). An initial computation of M starts in an *initial configuration*, that is a restarting configuration on *an input word* ($w \in \Sigma^*$). 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 that 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), where Q_A is the set of *accepting states* and Q_R is the set of *rejecting states*.

The *computation* of M is controlled by a finite set of *instructions* I. Instructions are of the following three types (where $q, q' \in Q, a \in$

$(\Sigma \cup \Gamma) \cup \{¢, \$\}$, $u \in (\Sigma \cup \Gamma)^* \cup (\Sigma \cup \Gamma)^*\{\$\}$, $v \in \{¢, \lambda\}(\Sigma \cup \Gamma)^*\{\$, \lambda\}$, $k \geq |au| > |v| \geq 0$):

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

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 (au 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 of the item containing a.

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 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 a transfer in the restarting configuration, i.e., entering the initial state and placing the head on the first item of the list (containing ¢).

– *Cycles,tails:* Any computation of a restarting automaton M is composed from certain phases. A phase called *cycle* starts in a restarting configuration, the head moves to the right along the input list until the restart operation is performed, and M is resumed in a new restarting configuration. M deletes at least once during a cycle – i.e., the new phase starts on a shortened word. A phase of a computation called *tail* starts in a restarting configuration, and the head moves to the right along the input list until one of the halting states is reached.

This immediately implies that any computation of any restarting automaton is finite (finishing in a halting state).

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

– *Accepting, recognizing:* An input *word w is accepted by M* if there is an (*initial*) computation that 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* $L(M)$.

– *RRWW-automata:* $RRWW$-automaton is a restarting automaton that makes exactly one *REWRITE*-instruction in each cycle. M can also once rewrite during any tail.

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

By RRW-automata we mean $RRWW$-automata with empty working alphabets. RRW-automata use only the input symbols in the rewriting, i.e., in all instructions of the form (2) above, the string v contains symbols from the input alphabet Σ only. We can see that 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 *RW*-automata (*R*-automata) we mean *RRW*-automata (*RR*-automata) which do restart immediately after any performance of a *REWRITE*-instruction.

– *Reductions:* 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) reduced to v by M. If $u \Rightarrow_M^* v$ holds, we say that u is stepwise reduced to v by M. If both u, v are from $L(M)$, we say that the *reduction* $u \Rightarrow_M v$ is *correct*.

We often (implicitly) use the next obvious claim:

CLAIM 2.1. (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 2.2. *For every* RRW*-automaton* M*,* $u \Rightarrow_M^* v$ *and* $v \in L(M)$ *always implies* $u \in L(M)$*, since all its restarting configurations are initial.*

– *Monotonicity properties*: Any cycle Cyc performed by an $RRWW$-automaton M has a significant configuration $c_w = (\cent u, q, v\$)$ in which it performs 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. We denote it $D_r(Cyc)$.

We say that a sequence of cycles $Cyc = Cyc_1, Cyc_2, \cdots Cyc_n$ of an $RRWW$-automaton M is *monotonic* if the sequence $D_r(Cyc_1)$,

$D_r(Cyc_2), \ldots, D_r(Cyc_n)$ is monotonic, i.e., not increasing. If Cyc is the (maximal) sequence of cycles of a computation $\mathcal{C}$, and Cyc is monotonic, we say that $\mathcal{C}$ is a monotonic computation. Notice that the tails are not considered for monotonicity.

We distinguish between three types of monotonicity of $RRWW$-automata:

By a *monotonic $RRWW$-automaton* we mean an $RRWW$-automaton where all its computations are monotonic.

By an *accepting-monotonic (a-monotonic) $RRWW$-automaton* we mean an $RRWW$-automaton where all the accepting computations are monotonic.

By a *general-monotonic (g-monotonic) $RRWW$-automaton* we mean an $RRWW$-automaton where for an arbitrary accepting computation on some word w there is an monotonic accepting computation on w.

– *Weak cyclic form*: We say that an $RRWW$-automaton M is in the *weak cyclic form* if for each word $w \in (\Sigma \cup \Gamma)^*$, $|w| > k$ no accepting tail can start by the configuration $(\lambda, q_0, ¢w\$)$.

We introduce some further special forms of $RRWW$-automata. These forms represent certain levels of determinism of $RRWW$-automata.

Three types of det-MVR-forms: We say that an $RRWW$-automaton M is in the *det-MVR*$_1$-form if it has the following two properties:

1) For any couple (q, v), where q is a state of M and v is a string of its symbols (of the length at most k), there is at most one instruction of the form $(q, v) \to (q', MVR)$.

2) If M rewrites in a tail, the tail is a rejecting one. Each accepting tail ends when the lookahead window moves completely past the right sentinel.

M is in the *det-MVR*$_2$-form if it is in the *det-MVR*$_1$-form, and for any pair of its instructions of the form $(q, v) \to (q', REWRITE(u))$, $(q, v) \to (q_1, MVR)$ (a pair of instructions with the same left-hand sides) it holds that the move-instruction $(q, v) \to (q_1, MVR)$ cannot be used in a cycle. This means, in other words that, the place of rewriting on a given word w is fixed for all possible cycles by an $RRWW$-automaton M in the *det-MVR*$_2$-form.

M is in the *det-MVR*$_3$-form if it is in the *det-MVR*$_2$-form, and for any of its instructions of the form $(q, v) \to (q', REWRITE(u))$ there is no MVR-instruction with the same left-hand side (i.e., (q, v)) in M. This means, in other words, that an $RRWW$-automaton M in the *det-MVR*$_3$-form can nondeterministically choose between several rewriting steps only.

LEMMA 2.3. *For any $RRWW$-automaton M there is an $RRWW$-automaton M_1 in the det-MVR_1-form such that $L(M) = L(M_1)$, and $u \Rightarrow_{M_1} v$ exactly if $u \Rightarrow_M v$.*

Proof. The construction of M_1 is based on two ideas:

1. On the well known construction of an equivalent deterministic finite automaton for a given nondeterministic one. In this way we get from M an automaton M_0 with the property that for any couple (q, v), where q is a state of M and v is a string of its symbols, there is at most one instruction of the form $(q, v) \to (q', MVR)$.

2. On the idea that M_1 simulating a phase Ph of M_0 decides on the place of rewriting given by Ph (if there is one) whether M_1 further simulates exactly the Ph (performs the rewriting and so on), or it transforms the Ph into a tail which does not use any rewriting. Naturally, this transformation must preserve the language recognized by M_0.

   In the latter case, M_1 will behave in the following way: It will simulate Ph using the moves to the right only, without performing the rewrite operation. If Ph finishes with accepting, M_1 accepts. In any other case M_1 rejects.

   In the case that Ph does not contain any rewriting, M_1 simulates exactly Ph.

□

Using the fact that $u \Rightarrow_{M_1} v$ exactly if $u \Rightarrow_M v$ the following consequence of Lemma 2.3 can be shown.

COROLLARY 2.4. *Let $X \in \{Y, mon\text{-}Y, a\text{-}mon\text{-}Y, g\text{-}mon\text{-}Y\}$, where $Y \in \{R, RW, RWW, RR, RRW, RRWW\}$. For any X-automaton M there is an X-automaton M_1 in the det-MVR_1-form such that $L(M) = L(M_1)$, and $u \Rightarrow_{M_1} v$ if and only if $u \Rightarrow_M v$.*

Remark. A bit more complex situation concerns the other *det-MVR*-forms. We will show some results about these forms in the next section.

– *Notation:* For brevity, prefix *wcf-* denotes the weak cyclic form versions of $RRWW$-automata, similarly *mon-* the monotonic, *a-mon-* the *a*-monotonic, and *g-mon-* the *g*-monotonic versions. Recall that det-MVR_i, for $i \in \{1, 2, 3\}$, denotes the three special forms introduced above. $\mathcal{L}(A)$, where A is some class of automata, denotes the class of languages recognizable by automata from A. E.g. the class of languages recognizable by monotonic RR-automata in the weak cyclic form is denoted by $\mathcal{L}(wcf\text{-}mon\text{-}RR)$.

3. Results

In the first part of this section we show that for any RR-automaton (RRW-, $RRWW$-automaton resp.) M with any type of monotonicity there exists an equivalent automaton M' of the same type as M in the weak cyclic form.

THEOREM 3.1. *Let $X \in \{Y, mon\text{-}Y, a\text{-}mon\text{-}Y, g\text{-}mon\text{-}Y\}$, where $Y \in \{RR, RRW, RRWW\}$. For any X-automaton M there is an X-automaton M_1 in the weak cyclic form such that $L(M) = L(M')$. I.e., $\mathcal{L}(X) = \mathcal{L}(wcf\text{-}X)$.*

Proof. We will outline a common transformation (construction) for all the above mentioned types X of $RRWW$-automata. Let M be an X-automaton. Because of Corollary 2.4, we can suppose M being in the $det\text{-}MVR_1$-form with the lookahead window of the size k, for some $k > 0$.

Let us also suppose that Rgl is the set of words which can be accepted by M in a tail (without restart). Let m be the number of MVR-instructions of M, and $p = m + k$. Let us consider the following fact.

Fact I. For any words v, w, y such that $vwy \in Rgl$, $|w| \geq p$ there is u_1, $u_1 \neq \lambda$ such that $w = xu_1z$, and $vxzy \in Rgl$ for some x, z.

In the case that the Rgl is finite, M can be simulated by an automaton M' which works in the same way as M and has the lookahead window of a size greater than the length of any word from Rgl (and at least k).

In the case that Rgl is an infinite language we will construct M' with the lookahead window of the size $p + k + 1$. M' simulates M using the first k symbols of the lookahead window. The suffix part of the length $p + 1$ serves for performing reductions based on Fact I (which does not change the (non)membership of the reduced word to the language Rgl). We need to use the suffix greater than p, because we need to reduce by the Fact I also with the right sentinel in the lookahead window.

We construct an X-automaton M' simulating M in such a way that if $u \Rightarrow_{M'} v$ by a cycle C' then

(r1) $Dr(C') > k + p$, and $u \Rightarrow_M v$, or

(r2) $Dr(C') > k + p$, both u and v are from Rgl, and there is a cycle C of M starting on the word u such that $Dr(C) = Dr(C')$, or

(r3) $Dr(C') \leq k + p$, $|u| \geq k + p$, and either u and v are from Rgl, or $u \Rightarrow_M v$.

The reductions of the types (r1), (r3) ensure fully the transformation into the weak cyclic form and at the same time the simulation of M by M'. The reductions of the type (r2) are used to ensure that M' rewrites in all places where rewrites M to preserve the possible monotonicities.

M' accepts or rejects (immediately) in a tail the words of the length at most $p+k$. Let us assume that $|w| > p+k$. M' will perform either a cycle or a rejecting tail on the string w. M' works in the following four modes: M'_0, M'_1, M'_2, M'_3.

M'_0: Any phase (i.e. cycle or tail) of M' starts in the mode M'_0. M being in the mode M'_0 moves its lookahead window from the left-hand side to the right-hand side, simulating the moving part of a (nondeterministically choosen) phase C of M. This mode ends by reaching the place p_1 in which C rewrites or the right sentinel appears in the lookahead of M'.

If $Dr(C) > k+p$, and if cannot use any MVR -instruction on p_1 then M' switches into the mode M'_1.

If $Dr(C) > k+p$, M can use (also) a *MVR*-instruction on the place p_1 then M' switches nondeterministically into the mode M'_1, or into the mode M'_2.

If the right sentinel appears in the lookahead of M' then M'_0 switches into the mode M'_3.

M'_1: The task of M'_1 is to perform the reductions of the type (r1). M'_1 continues in a plain simulation of C (performing its rewriting). If C ends by the restart, M' restarts. If C ends by rejecting, M' rejects. According to the *det-MVR*$_1$-form of M another possibility (an accepting) cannot occur.

M'_2: The task of M'_2 is to perform the reductions of the type (r2). M'_2 performs two subtasks at the same time in a parallel (and nondeterministic) way.

The first subtask continues in the simulation of C (not performing its rewriting), in order to check whether C is a cycle of M or not.

The second subtask simulates a another branch (continuation) of C, denoted as Br. The simulation of Br serves for two purposes: first for checking whether Br is an accepting tail or not; secondly for a detection and deletion of a nonempty substring u_1 in the first step of M'_2 by Fact I (see above).

If the phase Br is an accepting tail, simultaneously M'_2 performs some deleting, and the phase C is a cycle, then M' restarts. Otherwise M' rejects.

M_3'**:** The task of M_3' is to perform the reductions of the type (r3). M_3' nondeterministically chooses (having the right sentinel in the lookahead) one of the following three actions.

If the simulated phase C is recognized as an accepting tail then due to Fact I M_3' deletes some nonempty substring u_1 from the last p symbols of the word, and performs the restart.

If the simulated phase C is recognized as a cycle then M_3' simulates exactly this cycle (performing also its rewriting).

If the simulated phase C is recognized as a rejecting tail then M' rejects.

Obviously, the presented construction can be applied on any X-automaton M and yields an X-automaton M'. In particular, it is not hard to see that the monotonicity of M causes the monotonicity of M'.

We can also see that M' is in the weak cyclic form, and that $L(M) = L(M')$. □

In what follows, we are looking for a transformation of automata of the above mentioned classes into the *det-MVR*$_3$-form, and simultaneously into the weak cyclic form. This is possible for *RR-*,*RRW-* and *RRWW*-automata in the *det-MVR*$_2$-form.

THEOREM 3.2. *The following equality holds for* $X \in \{\lambda, W, WW\}$*:* $\mathcal{L}(det\text{-}MVR_2\text{-}mon\text{-}RRX) = \mathcal{L}(wcf\text{-}det\text{-}MVR_3\text{-}mon\text{-}RRX)$.

Proof. Let M be a *mon-RRX*-automaton in the *det-MVR*$_2$-form with the lookahead window of the size k. For the main part of the proof we take the slightly modified construction from the proof of Theorem 3.1 applied on a slightly modified M. We can suppose w.l.o.g. that M always halts with the head on the right sentinel.

When the automaton M is a *det-MVR*$_2$-*RRX*-automaton, then the automaton M' constructed according to the (non-modified) construction from the proof of Theorem 3.1 is also a *det-MVR*$_2$-*RRX*-automaton: There is only one place p_1 of rewriting for all phases of M' on a word w – this is the place where the mode M_0' ends. The added property of M ensures that M' always rewrites on p_1 in the modes M_1' and M_2'. It is not hard to see (with this assumption) that M' can be modified in such a way that it rewrites (deletes) also by rejecting in the mode M_3'. In this way the mode M_0' ends at the only place where (the modified) M' always rewrites, thus M' is in the *det-MVR*$_3$-form. The weak cyclic form and the monotonicity of the constructed *RRX*-automaton is obvious. □

Let us consider the following language $L_{RR} = L^1_{RR} \cup L^2_{RR} \cup L^3_{RR}$, where $L^1_{RR} = \{ww^R \mid w \in (cd{+}dc)^*\}$, $L^2_{RR} = \{wddw^R \mid w \in (cd{+}dc)^*\}$ and $L^3_{RR} = (cd)^*(dc)^*a$.

LEMMA 3.3. *$L_{RR} \in \mathcal{L}(wcf\text{-}mon\text{-}RR)$.*

Proof. A *wcf-mon-RR*-automaton M_{RR} recognizing L_{RR} can be sketched as follows.

M_{RR} contains two (sub)modules M_1, M_2.

M_{RR} at the beginning of any phase nondeterministically chooses one of the two modules M_1, M_2.

M_1 serves for checking of the membership to L^1_{RR} or to L^2_{RR}. M_1 is looking for a subword $v_1 = cddc$, or for $v_2 = dccd$, or for z in the form $(cddddc + dcddcd)$ at odd positions from the left end of the input word w of the form $(cd + dc)^*(\lambda + dd)(cd + dc)^*$.

Finding z, M_1 rewrites z into dd (using only deleting). After this rewriting M_1 checks, moving to the right, the form $(cd + dc)^*$ of the remainder of the word. If the check is positive M_{RR} restarts, otherwise it rejects. Finding v_1, or v_2, M_1 nondeterministically decides between rewriting and moving further to the right in the above described searching mode of M_1. M_1 rewrites (by deleting of c's) the chosen string v_1 (resp. v_2) into dd. Further it checks, moving to the right, the remainder of w, whether it has the form $(cd + dc)^*$. If the check is positive M_{RR} restarts, otherwise it rejects. M_1 directly accepts the empty word and dd.

M_2 checks the membership to L^3_{RR} in the following way: On an input word of the form $w = (cd)^*(dc)^*a$ of the length at least three it performs a cycle consisting of the deleting of the first pair of symbols from w. Beside this deleting M_2 checks during the cycle the form $(cd)^*(dc)^*a$. M_2 accepts the string a.

M_{RR} does not accept any string not mentioned above.

We can see that M_{RR} is a *wcf-mon-RR*-automaton recognizing L_{RR}.

We can also see that the place of rewriting in a cycle of M_{RR} on word w is not determined by the input word of the cycle. □

LEMMA 3.4.

$L_{RR} \in \mathcal{L}(wcf\text{-}mon\text{-}RR) - \mathcal{L}(wcf\text{-}det\text{-}MVR_3\text{-}mon\text{-}RRW)$.

Proof. The first part of this lemma is proved by Lemma 3.3. It remains to prove the second part.

Let us suppose that some *wcf-det-MVR$_3$-mon-RRW*-automaton M with the size of its lookahead k recognizes L_{RR}. Because of the weak cyclic form for any word of the form $w = (cd)^n(dc)^n$, where $n \geq k$, there is a correct reduction. Any correct reduction of w by M must be of the following form: $w \Rightarrow_M w_1$, where $w_1 = (cd)^{n-i}v(dc)^{n-i}$, for some i, v, $2i \leq k, |v| < k$. Let us also consider in a similar way the word $ww^R = (cd)^n(dc)^n(cd)^n(dc)^n$. Also for this word exists a correct reduction. Any correct reduction of ww^R by M must be of the following form: $ww^R \Rightarrow_M w_2$, where $w_2 = (cd)^n(dc)^{n-j}z(cd)^{n-j}(dc)^n$, for some j, z, $2i \leq k, |z| < k$. But that is a contradiction with the *det*-MVR_3-form of M (otherwise M could not choose nondeterministically between *MVR*- and *REWRITE*- steps in the middle of w). □

The following inequalites are a direct consequence of the previous assertions, and the equality follows from (a construction showed in) (Plátek, 1999) .

THEOREM 3.5. *The following proper inequality holds for $X \in \{\lambda, W\}$:*
$\mathcal{L}(wcf\text{-}det\text{-}MVR_3\text{-}mon\text{-}RRX) \subset \mathcal{L}(wcf\text{-}mon\text{-}RRX)$.
Further the following equality holds:
$\mathcal{L}(wcf\text{-}det\text{-}MVR_3\text{-}mon\text{-}RRWW) = \mathcal{L}(wcf\text{-}mon\text{-}RRWW) = CFL$.

Concluding remark. We have concentrated on the classification of languages recognized by restarting automata in the weak cyclic form. In the future, we will be interested also in some other types of special forms which promise to serve as a suitable paradigm for grammar checking.

References

Jančar, P., F. Mráz, M. Plátek and J. Vogel. On Monotony for Restarting Automata, In M.Kudlek, editor, *Proc. MFCS'98 workshop 'Mathematical Linguistics'*, 71-84, Bericht 213, Universität Hamburg, Fachbereich Informatik, 1998.

Kuboň, V. and M. Plátek. A Grammar Based Approach to a Grammar Checking of Free Word Order Languages, In *Proceedings of the 15th International Conference on Computational Linguistics COLING 94*, Vol. II, 906-910, Kyoto, 1994.

Marcus, S. Contextual Grammars and Natural Languages. In G. Rozenberg, A. Salomaa, editors, *Handbook of Formal Languages*, Volume 2, 215-235, Springer Verlag, 1997.

Plátek, M. Weak Cyclic Forms of Restarting Automata (extended abstract), In W.Thomas, editor, *Preproceedings of DLT 99*, 63-72, Aachen, 1999.

Plátek, M. Weak Cyclic Forms of Restarting Automata, *Technical Report* No 99/4, KTI MFF of Charles University, Prague, 1999.

Păun, Gh. *Marcus Contextual Grammar*, Kluver Academic Publishers, Dodrecht, 1997.