DEGREES OF (NON)MONOTONICITY OF RRW-AUTOMATA

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

and

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

ABSTRACT

We introduce and study the notion of j-monotonicity of RR- and RRW-automata (special types of restarting automata without working symbols). We show that the j-monotonicity can be considered as a degree of non-context-freeness of languages recognized by RRW-automata and RR-automata. 1-monotonic RRW-automata recognize a subset of the class of context-free languages. For $j > 1$, the j-monotonic RRW-automata recognize also some non-context-free languages and the j-monotonic RRW-automata as well as the j-monotonic RR-automata form a strict hierarchy with respect to the parameter j.

Keywords: restarting automata, j-monotonicity, degrees of non-context-freeness

1. Introduction

RRW-automata are special restarting automata. Roughly speaking, a restarting automaton accepts a word by a sequence of changes where each change results in a shorter word for which the automaton restarts.

A computation of an RRW-automaton on some word has the degree of (non)monotonicity j (j-monotonicity) when the sequence of distances of the (places of) changes to the right end of the word during the computation can be obtained by a 'shuffle' of at most j decreasing (not strictly) sequences (of distances). E.g. 1-monotonic computation means that its sequence of distances of the (places of) changes to the right end of the word is decreasing (not strictly). A RRW-automaton is j-monotonic if all its computations are j-monotonic.

The j-monotonicity is a generalization of the monotonicity (i.e. 1-monotonicity) studied in [5]. Here we study two types of the j-monotonic RRW-automata.

Let us recall our motivations for the study of restarting automata. The motivations are of linguistic and formal nature. One important linguistic motivation is to model *analysis by reduction* of natural language sentences in a similar way as in [9]. The analysis by reduction consists of a stepwise simplification of an extended sentence so that the (in)correctness of the sentence is not affected. Thus, after a certain number of steps, a simple sentence is obtained or an error is found. The reduction analysis is used to handle word-order in Czech (dependency) syntax, cf. [10].

Each restart transfers an RRW-automaton into an initial configuration. This property of RRW-automata stresses the syntactic features of languages recognized by RRW-automata. Among the types of reductions, we consider the reduction by deletion as methodically significant. That is the reason for the study of RR-automata. From the linguistic point of view the j-monotonicity of an accepting computation helps to express the complexity of word-order of the accepted sentence. The j-monotonicity of an RRW-automaton M expresses the complexity of the word-order of the language recognized by M. It is a similar tool as the measures of nonprojectivity for free-order dependency grammars introduced in [2].

Works on restarting automata can be considered as a contribution to the study of regulated rewriting, cf. [8], [1]. From the point of view of the formal language theory, the j-monotonicity expresses the degree of non-context-freeness of languages recognized by RRW-automata. 1-monotonic RRW-automata recognize a subset of the class of context-free languages. For $j > 1$, the j-monotonic RRW-automata recognize also some non-context-free languages and their recognition power increases with increasing j.

In this paper we extend the taxonomy of language classes given in [3] by considering the mentioned degrees of monotonicity in a way which stresses the transparency of computations (rewriting, reductions) by RRW-automata.

2. Definitions

Throughout the paper, λ denotes the empty word. We start with a definition of restarting automata in special forms of which we shall be interested in.

A *restarting automaton* M, formally presented as a tuple $M = (Q, \Sigma, \Gamma, k, I, q_0, Q_A, Q_R)$, has a control unit with a finite set Q of *(control) states*, a distinguished *initial state* $q_0 \in Q$, and two disjoint sets $Q_A, Q_R \subseteq Q$ of *halting states*, called *accepting* and *rejecting* respectively. M further has one head moving on a finite linear list of items (cells). The first item always contains a special symbol ¢, i.e. the *left sentinel*, the last item always contains another special symbol \$, i.e. the *right sentinel*, and each other item contains a symbol either from an *input alphabet* Σ or a *working alphabet* Γ; we stipulate that Σ and Γ are finite disjoint sets and $¢, \$ \notin \Sigma \cup \Gamma$. We assume the head to have a *lookahead* window so that it always scans k consecutive items ($k \geq 1$) or maybe less when the right sentinel \$ appears in the window.

A *configuration* of M is a string $\alpha q \beta$ where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot (\Sigma \cup \Gamma)^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot (\Sigma \cup \Gamma)^*$ and $\beta \in (\Sigma \cup \Gamma)^* \cdot \{\$\}$; here q represents the current (control) state, $\alpha\beta$ the current contents of the list of items while it is understood that the head scans the first k symbols of β (or the whole β when $|\beta| < k$). A *restarting configuration*, for a word $w \in (\Sigma \cup \Gamma)^*$, is of the form $q_0 ¢ w \$$; if $w \in \Sigma^*$, $q_0 ¢ w \$$ is an *initial configuration*.

As usual, a *computation* of M (for an input word $w \in \Sigma^*$) is a sequence of configurations starting with an initial configuration where two consecutive configurations are in the relation $\vdash_M$ (denoted $\vdash$ when M is clear from the context) induced by a finite set I of instructions. Each *instruction* is of one of the following three types:

(1) $(q, \gamma) \rightarrow (q', MVR)$

(2) $(q, \gamma) \rightarrow (q', REWRITE(\gamma'))$

(3) $(q, \gamma) \rightarrow RESTART$

Here q is a nonhalting state ($q \in Q - (Q_A \cup Q_R)$), $q' \in Q$, and γ, γ' are "lookahead window contents" where $|\gamma| > |\gamma'|$ (i.e., rewriting must shorten the word).

Type (1) (moving right), where $\gamma = a\beta$, a being a symbol, induces $\alpha q a \beta \delta \vdash \alpha a q' \beta \delta$. Type (2) (rewriting) induces $\alpha q \gamma \delta \vdash \alpha \gamma' q' \delta$; but if $\gamma = \beta\$$, in which case $\gamma' = \beta'\$$, we have $\alpha q \beta \$ \vdash \alpha \beta' q' \$$. Type (3) (restarting) induces $\alpha q \gamma \delta \vdash q_0 \alpha \gamma \delta$.

We assume that there is an instruction with the left-hand side (q, γ) for each nonhalting state and each (possible) γ (i.e., in any configuration a further computation step is possible iff the current state is nonhalting). In general, the automaton is *nondeterministic*, i.e., there can be two or more instructions with the same left-hand side (q, γ), and thus there can be more than one computation for an input word. If this is not the case, the automaton is *deterministic*.

An input *word* $w \in \Sigma^*$ *is accepted by* M if there is a computation which starts with the initial configuration $q_0 ¢ w \$$ and finishes with an *accepting configuration* – where the current control state is accepting. $L(M)$ denotes the language consisting of all words accepted by M; we say that M *recognizes (accepts) the language* $L(M)$.

We observe that any computation of a restarting automaton M consists of certain phases. A phase called a *cycle* starts in a (re)starting configuration, the head moves to the right along the input list until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a *tail*. For technical convenience, we assume that M performs at least one rewrite operation during any cycle (this is controlled by the finite state control unit) – thus the new phase starts on a shortened word; it also means that any computation is finite (finishing in an accepting or rejecting configuration).

We use the notation $u \Rightarrow_M v$ meaning that there is a cycle of M beginning with the restarting configuration $q_0 ¢ u \$$ and finishing with the restarting configuration $q_0 ¢ v \$$; the relation $\Rightarrow_M^*$ is the reflexive and transitive closure of $\Rightarrow_M$. We sometimes (implicitly) use the following obvious fact referred as "the error preserving property":

Fact 2.1 *Let M be a restarting automaton, and u, v two words in its* input *alphabet. If $u \Rightarrow_M^* v$ and $u \notin L(M)$, then $v \notin L(M)$.*

Now we recall the subclasses of restarting automata relevant for our consideration.

- An *RRWW-automaton* is a restarting automaton which performs exactly one *REWRITE*-instruction in each cycle (this is controlled by the finite state control unit).
- An *RRW-automaton* is an *RRWW*-automaton whose working alphabet is empty. Note that each restarting configuration is initial in this case.
- An *RR-automaton* is an *RRW*-automaton whose rewriting instructions can be viewed as deletions, i.e., in the instructions of type (2), γ' is obtained by deleting some symbols from γ.

Another obvious fact to which we sometimes refer is the following version of a "pumping lemma":

Fact 2.2 *For any RRWW-automaton M there is a constant p such that the following holds. If $uvw \Rightarrow_M uv'w$ then in each subword u_2 with length p of the word u, we write $u = u_1u_2u_3$, we can find a nonempty subword z_2, we write $u_2 = z_1z_2z_3$, such that $u_1z_1(z_2)^iz_3u_3vw \Rightarrow_M u_1z_1(z_2)^iz_3u_3v'w$ for all $i \geq 0$ (i.e., z_2 is a "pumping subword" in the respective cycle). Similarly, such a pumping subword can be found in any subword with length p of the word w.*

Next, we recall the notion of monotonicity for a computation of an *RRWW*-automaton and we introduce the notion of j-monotonicity. Any cycle C contains a unique configuration $\alpha q \beta$ in which a rewriting instruction is applied. We call $|\beta|$ the *right distance*, *r-distance*, of C, and denote it $D_r(C)$. We say that a *sequence of cycles* $Sq = C_1, C_2, \ldots, C_n$ is *monotonic* iff $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$; A *computation* is *monotonic* iff the respective sequence of cycles is monotonic. (The tail of the computation does not play any role here.)

Let j be a natural number. We say that the sequence of cycles $Sq = C_1, C_2, \ldots, C_n$ is *j-monotonic* iff (informally speaking) there is a partition of Sq into p subsequences where each of them is a monotonic one.

More formally we say that the sequence of cycles Sq is *j-monotonic* iff there is a partition of $\{1, \ldots, n\}$ into j (naturally) ordered subsets such that $\{1, \ldots, n\} = \{i_1^1, i_2^1, \ldots, i_{p_1}^1\} \cup \{i_1^2, i_2^2, \ldots, i_{p_2}^2\} \ldots \cup \{i_1^j, i_2^j, \ldots, i_{p_j}^j\}$, where all the j sequences of cycles

$$\begin{gathered} C_{i_1^1}, C_{i_2^1}, \ldots, C_{i_{p_1}^1}, \\ C_{i_1^2}, C_{i_2^2}, \ldots, C_{i_{p_2}^2}, \\ \vdots \\ C_{i_1^j}, C_{i_2^j}, \ldots, C_{i_{p_j}^j} \end{gathered}$$

are monotonic.

We will use the following obvious fact:

Fact 2.3 *A sequence of cycles $C_1, C_2, \ldots, C_n$ is not j-monotonic iff there exist $1 \leq s_1 < s_2 < \ldots < s_{j+1} \leq n$ such that $D_r(C_{s_1}) < D_r(C_{s_2}) < \cdots < D_r(C_{s_{j+1}})$.*

A *computation* is *j-monotonic* iff the respective sequence of cycles is j-monotonic. (The tail of the computation does not play any role here.) We can see that the 1-monotonicity of a computation means the monotonocity of a computation.

Let j be a natural number. An *RRWW-automaton M* is *j-monotonic* iff all its computations are j-monotonic.

Notation. For brevity, prefix *j-mon* denotes j-monotonicity. $\mathcal{L}(A)$, where A is some class of automata, denotes the class of languages recognizable by the automata from A. E.g., $\mathcal{L}(j\text{-}mon\text{-}RRW)$ denotes the class of languages recognizable by j-monotonic RRW-automata. *CFL* denotes the class of context-free languages, *DCFL* the class of deterministic context-free languages. The sign $\subset$ means the proper subset relation, the sign *Nat* the set of natural numbers. We will sometimes write regular expressions instead of the respective regular languages.

3. Degrees of non-monotonicity

In this section we will show the main result that the j-monotonic RRW-automata as well as the j-monotonic RR-automata form a strict hierarchy with respect to the parameter j.

Next we introduce a sequence of languages $L_1, L_2, \ldots$, which will be used as witness languages for separations (e.g. of classes $\mathcal{L}(j\text{-}mon\text{-}RRW)$ for different j's).

Examples. Let $j \in Nat$, let us consider the following alphabet $A_j = \Sigma_{ab} \cup \Sigma_{cd} \cup \{c_0, c_1, \ldots, c_{j-1}\}$, where $\Sigma_{ab} = \{a, b\}$ and $\Sigma_{cd} = \{c, d\}$. Further, let $L_{ab} = (\Sigma_{ab} \cdot \Sigma_{cd})^*$ and $L_{cd} = \Sigma_{cd} \cdot (\Sigma_{ab} \cdot \Sigma_{cd})^*$. We define a sequence of languages in the following way: for each $j \in Nat$ let

$$L_j = \bigcup_{0 \leq i < j} \{c_0 x w c_1 x w \ldots c_{i-1} x w c_i w c_{i+1} w \ldots c_{j-2} w c_{j-1} w \mid \begin{array}{r}(x \in \Sigma_{ab} \text{ and } w \in L_{cd}) \text{ or} \\ (x \in \Sigma_{cd} \text{ and } w \in L_{ab})\}.\end{array}$$

For a word $c_0 w_1 c_1 w_2 c_2 \ldots c_{j-1} w_j$ from L_j, the words $w_1, \ldots, w_j$ are either empty words or words in which the symbols from Σ_{ab} and Σ_{cd} alternate and their last symbol is from Σ_{cd}. If at least one of the words $w_1, \ldots, w_j$ is nonempty, then

$$w_1 = w_2 = \ldots = w_i = xw \qquad w_{i+1} = w_{i+2} = \ldots = w_j = w$$

for some i, $1 \leq i \leq j$, some symbol $x \in \Sigma_{ab} \cup \Sigma_{cd}$ and some word w such that $xw \in L_{ab} \cup L_{cd}$.

Claim 3.1 *For any $j \in Nat$:*

$$L_j \in \mathcal{L}(j\text{-}mon\text{-}RR).$$

Proof. We outline a j-mon-RR-automaton M recognizing L_j. Let $u \in (A_j)^*$ be the current word in the list of M (between the sentinels). M looks at the first two symbols of u, if they equal to:

- c_0c_1, then M checks the equality $u = c_0c_1...c_{j-1}$. In the positive case M accepts, in the negative case M rejects.
- c_0x, for some $x \in \Sigma_{ab} \cup \Sigma_{cd}$, then M nondeterministically guesses i, $0 \leq i < j$, and checks whether u is of the form

$$c_0xw_1c_1xw_2c_2 \dots c_{i-1}xw_ic_iw_{i+1}c_{i+1}w_{i+2} \dots c_{j-2}w_{j-1}c_{j-1}w_j,$$

  where either $x \in \Sigma_{ab}$ and $w_1, \dots, w_j \in L_{cd}$ or $x \in \Sigma_{cd}$ and $w_1, \dots, w_j \in L_{ab}$. If the result of the check is negative, M rejects. In the positive case, during the corresponding cycle M deletes the first symbol after c_{i-1} (i.e. x) and at the right end of the current list it restarts.

Any computation of M can be divided into parts in which the same symbol x from $\Sigma_{ab} \cup \Sigma_{cd}$ is deleted. Each part comprising cycles deleting some symbol from Σ_{ab} can be followed by a part comprising cycles deleting some symbol from Σ_{cd} only, and vice versa. In an accepting computation the first part can have less than j cycles, all the remaining parts have exactly j cycles. In a rejecting computation the first and the last part can contain less than j cycles, all the remaining parts contain exactly j cycles.

It is easy to see that any computation (without the tail) can be partitioned into (at most) j monotonic subsequences of cycles, simply so that for the first subsequence are taken the first cycles from the parts, for the second one the second cycles of the parts and so on. This observation proves the claim. □

Claim 3.2 *For any $j \in Nat$*

$$L_{j+1} \notin \mathcal{L}(j\text{-}mon\text{-}RRW).$$

Proof. Let us suppose that $M = (Q, \Sigma, \emptyset, k, I, q_0, Q_A, Q_R)$ is a j-monotonic RRW-automaton recognizing L_{j+1}. Let p be the constant determined for M by the Fact 2.2 (pumping lemma). Let us take any word $z_0 = c_0wc_1wc_2...c_j \in L_{j+1}$ where $w \in L_{ab}$ and $|w| = n > 3kp$ (k is the size of the lookahead window of M).

Let us consider an accepting computation $\mathcal{C}$ of M on z_0. This computation contains at least one cycle C_0. Otherwise because of the length of z_0, using Fact 2.2 we can construct an accepting tail for a word outside L_{j+1}. Let the resulting word after C_0 be z_1. Because $z_1 \in L_{j+1}$ the only possible change performed by C_0 is the deletion of the first symbol after the symbol c_j. Therefore, $z_1 = c_0wc_1wc_2...w_1c_{j-1}wc_jw'$, where $w = xw'$ for some $x \in \Sigma_{ab}$.

We can see that $\mathcal{C}$ continues (for similar reasons) by another j cycles C_1,...C_j, where C_i deletes the first symbol after c_{j-i} (for $1 \leq i \leq j$). We can see that $D_r(C_0) < D_r(C_1) < ... < D_r(C_j)$. That is a contradiction with the assumption that M is a j-monotonic RRW-automaton (Fact 2.3). □

Theorem 3.3 *For any $j \in Nat$, the following proper inclusions hold:*

$$\mathcal{L}(j\text{-}mon\text{-}RRW) \subset \mathcal{L}((j{+}1)\text{-}mon\text{-}RRW),$$
$$\mathcal{L}(j\text{-}mon\text{-}RR) \subset \mathcal{L}((j{+}1)\text{-}mon\text{-}RR).$$

Proof. We can see from the definition of j-monotonicity that

$$\mathcal{L}(\textit{j-mon-RRW}) \subseteq \mathcal{L}(\textit{(j+1)-mon-RRW}),$$
$$\mathcal{L}(\textit{j-mon-RR}) \subseteq \mathcal{L}(\textit{(j+1)-mon-RR}).$$

From the previous two claims directly follow the proper inclusions. □

Lemma 3.4 *$\mathcal{L}$(1-mon-RRW) − $\mathcal{L}$(RR) is not empty.*

Proof. In fact, this lemma is proved by Lemma 4.2 in [5] using the following witness language $L_1 = \{ f, ee \} \cdot \{ c^n d^n \mid n \geq 0 \} \ \bigcup \ \{ g, ee \} \cdot \{ c^n d^m \mid m > 2n \geq 0 \}$. □

Theorem 3.5 *For any $j \in Nat$, the following proper inclusion holds:*

$$\mathcal{L}(\textit{j-mon-RR}) \subset \mathcal{L}(\textit{j-mon-RRW}).$$

Proof. Obviously, $\mathcal{L}(\textit{j-mon-RR}) \subseteq \mathcal{L}(\textit{j-mon-RRW})$. The proper inclusions follow from the previous lemma. □

Theorem 3.6 *For any $j \in Nat$, the following proper inclusions hold:*

$$\mathcal{L}(\textit{j-mon-RR}) \subset \mathcal{L}(RR),$$
$$\mathcal{L}(\textit{j-mon-RRW}) \subset \mathcal{L}(RRW).$$

Proof. For any $j \in Nat$ holds that

$$\mathcal{L}(\textit{j-mon-RR}) \subseteq \mathcal{L}(RR),$$
$$\mathcal{L}(\textit{j-mon-RRW}) \subseteq \mathcal{L}(RRW).$$

The proper inclusions follow from Theorem 3.3. □

4. Degrees of non-context-freeness

In [5], it was shown that

$$\mathcal{L}(\textit{mon-RRWW}) = CFL,$$
$$\mathcal{L}(\textit{1-mon-RRW}) \subset CFL,$$
$$DCFL \subset \mathcal{L}(\textit{1-mon-RR}).$$

On the other hand we can see that already the second witness language L_2 is not a context-free language. These observations and the presented results give us the opportunity to consider the j-monotonicity as the degree of non-context-freeness of RR-languages and RRW-languages.

5. Additional Remarks

Motivated by the syntax of free-word-order natural languages we will apply the j-monotonicity to such $RRWW$-automata which allow to write only a limited number of non-input symbol during a computation in the future. Some other considerations of the (descriptional) complexity nature, like in [6], can be taken into the account – all the sample languages $L_1, L_2, \ldots$ from Sect. 3 can be recognized by RR-automata with the size of lookahead 1.

It turns out that the 1-monotonicity property is decidable for (all) $RRWW$-automata, see [4]. We are also able to show that the 2-monotonicity property is decidable for (all) $RRWW$-automata. We believe that we will be able to show the decidability of j-monotonicity for any natural j in the future.

We also believe that we will be able to show a polynomial algorithm for recognition of j-monotonic $RRW(W)$-languages.

6. Acknowledgements

The research reported in this paper has been supported by the Grant Agency of the Czech Republic, Grant-No. 201/99/0236, and by the Grant Agency of Charles University, Grant-No. 157/1999/A INF/MFF.

References

[1] J. Dassow and G. Păun, *Regulated rewriting in formal language theory.* Number 18 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1989.

[2] T. Holan, V. Kuboň, K. Oliva, and M. Plátek, Two useful measures of word order complexity. In: A. Polguére and S. Kahane (eds), *Proceedings of the Workshop 'Processing of Dependency-Based Grammars'.* Université de Montréal, Montréal, Quebeck, Canada, August 1998, 21–28.

[3] P. Jančar, F. Mráz, M. Plátek, and J. Vogel, On restarting automata with rewriting. In: G. Păun and A. Salomaa (eds), *New Trends in Formal Language Theory (Control, Cooperation and Combinatorics).* LNCS **1218**, Springer-Verlag, 1997, 119–136.

[4] P. Jančar, F. Mráz, M. Plátek, and J. Vogel, Different types of monotonicity for restarting automata. In: V. Arvind and R. Ramanujam (eds), *FST&TCS'98.* LNCS **1530**, Springer-Verlag, 1998, 343–354.

[5] P. Jančar, F. Mráz, M. Plátek, and J. Vogel, On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics*, **4**(4), 1999, 287–311.

[6] F. Mráz, Lookahead hierarchies of restarting automata. In: O. Boldt and H. Jürgensen (eds), *Pre-Proceedings of DCAGRS 2000 Descriptional Complexity of Automata, Grammars and Related Structures.* Dept. of Comp. Science, The University of Western Ontario, London, Canada, 2000. As Report No. 255. Extended version of [7].

[7] F. Mráz, Lookahead hierarchies of restarting automata. *Journal of Automata, Languages and Combinatorics*, 2001. Accepted for publication.

[8] G. Niemann and F. Otto, Restarting automata, Church-Rosser languages, and representations of r.e. languages. In: G. Rozenberg and W. Thomas (eds), *Developments in Language Theory - Foundations, Applications, and Perspectives, Proceedings DLT 1999.* World Scientific, Singapore, 2000, 103–114.

[9] M. Novotný, *With Algebra from Language to Grammar and back (S algebrou od jazyka ke gramatice a zpět).* Academia, Praha, 1988. *In Czech.*

[10] M. Straňáková, Selected types of pg-ambiguity. *The Prague Bulletin of Mathematical Linguistics*, **72** (1999), 29–57.