On the Gap-Complexity of Simple RL-Automata*

F. Mráz[1], **F. Otto**[2], and **M. Plátek**[1]

[1] Charles University, Faculty of Mathematics and Physics
Department of Computer Science, Malostranské nám. 25
118 00 Praha 1, Czech Republic
mraz@ksvi.ms.mff.cuni.cz, Martin.Platek@mff.cuni.cz

[2] Fachbereich Mathematik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Abstract. *Analysis by reduction* is a method used in linguistics for checking the correctness of sentences of natural languages. This method is modelled by *restarting automata*. Here we introduce and study a new type of restarting automaton, the so-called t-sRL-*automaton*, which is an RL-automaton that is rather restricted in that it has a window of size 1 only, and that it works under a minimal acceptance condition. On the other hand, it is allowed to perform up to t rewrite (that is, delete) steps per cycle. Here we study the *gap-complexity* of these automata. The membership problem for a language that is accepted by a t-sRL-automaton with a bounded number of gaps can be solved in polynomial time. On the other hand, t-sRL-automata with an unbounded number of gaps accept NP-complete languages.

1 Introduction

The original motivation for introducing the restarting automaton was the desire to model the so-called *analysis by reduction* of natural languages. Analysis by reduction is usually presented by finite samples of sentences of a natural language and by sequences of their correct reductions (e.g., tree-banks) (see, e.g., [7]).

From a theoretical point of view the restarting automaton can be seen as a tool that yields a very flexible generalization of analytical grammars, that is, in a very flexible way it introduces a basic syntactic system (an approximation to the formalization of the analysis by reduction), which contains the full information about the input vocabulary (set of wordforms), the categorial vocabulary, the set of reductions (rewritings), the recognized language, the language of sentential forms, and the categorial language. On the other hand, the restarting automaton can be considered as a generalization and a refinement of the pushdown automaton (see, e.g., [9]) and the contraction automaton [14].

Up to now all models of restarting automata studied in the literature accept at least all deterministic context-free languages. Hence, they can be used to analyze only language classes above the level of deterministic context-free languages. In particular, regular languages or even finite languages are of too small complexity to be studied by restarting automata. However, in (corpus) linguistics essentially finite (though very large) approximations of infinite languages are often studied. As the

* The first and the third author were partially supported by the Grant Agency of the Czech Republic under Grant-No. 201/04/2102 and by the program 'Information Society' under project 1ET100300517.

motivation for restarting automata is derived from linguistic considerations, this is a shortcoming of the model.

Here we propose a way to remedy this situation. We introduce a new variant of the restarting automaton, the so-called *simple* RL-*automaton* (sRL-automaton), that is rather restricted in various aspects to ensure that its expressive power is limited. However, by admitting that t (≥ 1) delete operations may be performed in each cycle, the expressive power of the obtained model of the restarting automaton is parametrized by t, and hence, we obtain an infinite hierarchy of automata and therewith of language classes. Then we study the number of gaps generated during a reduction as a complexity measure for t-sRL-automata that is inspired by linguistic considerations (see, e.g., [2, 3]). This measure is related to the notion of j-monotonicity considered in [12], and it yields an infinite hierarchy of automata and language classes. In contrast to the situation for 2-monotone restarting automata, which accept NP-complete languages [5], it turns out that a bounded number of gaps implies that only feasible languages are accepted, that is, languages that are recognizable in polynomial time. However, with an unbounded number of gaps these automata still accept NP-complete languages.

The paper is structured as follows. After introducing the simple RL-automata in Section 2, we will establish an infinite hierarchy based on the parameter t of delete operations that are permitted per cycle. In Section 3 various notions of monotonicity are presented for sRL-automata, and it is shown that sRL-automata can simulate standard RL-automata, preserving the type of monotonicity. In particular, this implies that t-right-left-monotone sRL-automata accept NP-complete languages. Then in Section 4 the gap-complexity is introduced, and it is shown that with bounded gap-complexity sRL-automata only accept feasible languages. The paper closes with a characterization of regular languages in terms of a special type of sRL-automata.

2 t-sRL-Automata

Here we describe in short the type of restarting automaton we will be dealing with. More details on restarting automata in general can be found in [9, 10].

An sRL-*automaton* (*simple* RL-*automaton*) M is a (in general) nondeterministic machine with a finite-state control Q, a finite input alphabet Σ, and a head (window of size 1) that works on a flexible tape delimited by the left sentinel ¢ and the right sentinel \$. For an input $w \in \Sigma^*$, the initial tape inscription is ¢w\$. To process this input M starts in its initial state q_0 with its window over the left end of the tape, scanning the left sentinel ¢. According to its transition relation, M performs *move-right steps* which shift the window one position to the right, thereby possibly changing the state of M, *move-left steps* which shift the window one position to the left, thereby possibly changing the state of M, and *delete steps*, which delete the content of the window, thus shortening the tape, change the state, and shift the window to the right neighbour of the symbol deleted. Of course, neither the left sentinel ¢ nor the right sentinel \$ must be deleted. At the right end of the tape M either halts and accepts, or it halts and rejects, or it *restarts*, that is, it places its window over the left end of the tape and reenters the initial state. It is required that before the first restart step and also between any two restart steps, M executes at least one delete operation.

We say that M is an sRR-automaton if M does not use any move-left steps. By sRL (sRR) we denote the class of all sRL-automata (sRR-automata). A t-sRL-automaton ($t \geq 1$) is an sRL-automaton which uses at most t delete operations in a cycle, and similarly we obtain the t-sRR-automaton.

A *configuration* of M is a string $\alpha q \beta$ where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Sigma^*$ and $\beta \in \Sigma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first symbol of β. A *restarting configuration* is of the form $q_0 ¢ w \$$, where $w \in \Sigma^*$.

We observe that any finite computation of an sRL-automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the window is moved along the tape by performing move-right, move-left, and (at least one) delete operations until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, each finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. We assume that no delete operation is executed in a tail computation.

We use the notation $u \vdash^c_M v$ to denote a cycle of M that begins with the restarting configuration $q_0 ¢ u \$$ and ends with the restarting configuration $q_0 ¢ v \$$; the relation $\vdash^{c^*}_M$ is the reflexive and transitive closure of $\vdash^c_M$.

An input $w \in \Sigma^*$ is *accepted* by M, if there is an accepting computation which starts with the (initial) configuration $q_0 ¢ w \$$. By $L(M)$ we denote the language consisting of all words accepted by M; we say that M *recognizes (accepts) the language* $L(M)$. By $S(M)$ we denote the *simple language* accepted by M, which consists of all words that M accepts by tail computations. Obviously, $S(M)$ is a regular sublanguage of $L(M)$. By $\mathsf{RS}(M)$ we denote the *reduction system* $\mathsf{RS}(M) := (\Sigma^*, \vdash^c_M, S(M))$ that is induced by M. Observe that, for each $w \in \Sigma^*$, we have $w \in L(M)$ if and only if $w \vdash^{c^*}_M v$ holds for some word $v \in S(M)$.

By $\mathcal{L}(\mathsf{A})$ we denote the class of languages that are accepted by automata of type A (A-automata), and by $\mathcal{L}_{\leq n}(\mathsf{A})$ we denote the class of finite languages that are accepted by automata of type A and that do not contain any words of length exceeding the number n.

On the set of input words Σ^*, we define a partial ordering $\leq$ as follows:

$$u \leq v \text{ if and only if } u \text{ is a scattered subword of } v.$$

By $<$ we denote the proper part of $\leq$. Obviously, $\leq$ is well-founded, that is, there do not exist infinite descending sequences with respect to $<$.

For $L \subseteq \Sigma^*$, let $L_{\min} := \{\, w \in L \mid u < w \text{ does not hold for any } u \in L \,\}$, that is, $L_{\min}$ is the set of minimal words of L. It is well-known that $L_{\min}$ is finite for each language L (see, e.g., [8]). We say that an sRL-automaton M accepting the language L works with *minimal acceptance* if it accepts in tail computations exactly the words of the language $L_{\min}$, that is, $S(M) = L_{\min}$. Thus, each word $w \in L \smallsetminus L_{\min}$ is reduced to a word $w' \in L_{\min}$ by a sequence of cycles of M. We will use the prefix min- to denote sRL-automata that work with minimal acceptance.

Here we will mainly be interested in t-sRL-automata with minimal acceptance. In this way we will achieve similarities between certain classes of finite and infinite languages recognized by sRL-automata, as an sRL-automaton with minimal acceptance is forced to perform sequences of cycles even for accepting a regular language. In fact, this is even true for most finite languages.

Example 1. Let $t \geq 1$, and let $L^{<t>} := \{a^t, \lambda\}$. Then $L^{<t>}_{\min} = \{\lambda\}$. Hence, an sRL-automaton for the language $L^{<t>}$ that works with minimal acceptance must execute the cycle $a^t \vdash^c \lambda$, which means that it must execute t delete operations during this cycle. Hence, it is a t-sRL-automaton.

Concerning the relationship between sRR- and sRL-automata, we have the following important result which generalizes a corresponding result for RLWW-automata from [11].

Theorem 1. *For each integer $t \geq 1$ and each t-*sRL-automaton *M, there exists a t-*sRR-automaton *M' such that the reduction systems* $\mathsf{RS}(M)$ *and* $\mathsf{RS}(M')$ *coincide.*

Observe that, in each cycle, M' executes its up to t delete operations strictly from left to right, while M may execute them in arbitrary order.

Proof. Each cycle of a computation of M consists of (up to) $2t + 2$ phases. In phases $1, 3, \ldots, 2i + 1, \ldots, 2t + 1$, M shifts its window across the tape by executing move-left and move-right steps, in phases $2, 4, \ldots, 2i, \ldots, 2t$, M executes a delete operation, and in phase $2t+2$, M performs a restart step. Thus, the sRR-automaton M' must guess the positions of the delete steps and the crossing sequences of M corresponding to the move-phases. As within a move-phase M need not visit the same tape cell twice while being in the same state, we see that the corresponding crossing sequence is bounded in length. Hence, M' can indeed guess the (up to) $t+1$ crossing sequences and verify that they are consistent with each other and with the chosen delete operations. □

Based on Theorem 1 we can describe a t-sRL-automaton by meta-instructions of the form

$$(¢ \cdot E_0, a_1, E_1, a_2, E_2, \ldots, E_{s-1}, a_s, E_s \cdot \$),$$

where $1 \leq s \leq t$, $E_0, E_1, \ldots, E_s$ are regular languages (often represented by regular expressions), called the *regular constraints* of this instruction, and $a_1, a_2, \ldots, a_s \in \Sigma$ correspond to letters that are deleted by M during one cycle. On trying to execute this meta-instruction starting from a configuration $q_0¢w\$$, M will get stuck (and so reject), if w does not admit a factorization of the form $w = v_0a_1v_1a_2 \ldots v_{s-1}a_sv_s$ such that $v_i \in E_i$ for all $i = 0, \ldots, s$. On the other hand, if w admits factorizations of this form, then one of them is chosen nondeterministically, and $q_0¢w\$$ is transformed into $q_0¢v_0v_1 \ldots v_{s-1}v_s\$$. In order to also describe the tails of accepting computations, we use accepting meta-instructions of the form $(¢ \cdot E \cdot \$, \mathsf{Accept})$, where E is a regular language. Actually we can require that there is only a single accepting meta-instruction for M. If M works with minimal acceptance, then this accepting meta-instruction is of the form $(¢ \cdot L(M)_{\min} \cdot \$, \mathsf{Accept})$.

Example 2. Let $t \geq 1$, and let $LR_t := \{\, c_0wc_1wc_2\ldots c_{t-1}w \mid w \in \{a,b\}^* \,\}$, where $\Sigma_0 := \{a,b\}$ and $\Sigma_t := \{c_0, c_1, \ldots, c_{t-1}\} \cup \Sigma_0$. We obtain a t-sRR-automaton M_t for the language LR_t through the following sequence of meta-instructions:

$$\begin{aligned}&(1)\ (\cent c_0, a, \Sigma_0^* \cdot c_1, a, \Sigma_0^* \cdot c_2, \ldots, \Sigma_0^* \cdot c_{t-1}, a, \Sigma_0^* \cdot \$),\\ &(2)\ (\cent c_0, b, \Sigma_0^* \cdot c_1, b, \Sigma_0^* \cdot c_2, \ldots, \Sigma_0^* \cdot c_{t-1}, b, \Sigma_0^* \cdot \$),\\ &(3)\ (\cent c_0c_1\ldots c_{t-1}\$, \mathsf{Accept}).\end{aligned}$$

It follows easily that $L(M_t) = LR_t$ holds, and that M_t works with minimal acceptance. Actually, the automaton M_t is even deterministic.

We emphasize the following properties of restarting automata, which are used implicitly in proofs. They play an important role in linguistic applications of restarting automata (e.g., for the analysis by reduction, grammar-checking, and morphological disambiguation).

Definition 1. (Correctness Preserving Property)
A t-sRL-automaton M is (strongly) correctness preserving *if $u \in L(M)$ and $u \vdash_M^{c^*} v$ imply that $v \in L(M)$.*

Definition 2. (Error Preserving Property)
A t-sRL-automaton M is error preserving *if $u \notin L(M)$ and $u \vdash_M^{c^*} v$ imply that $v \notin L(M)$.*

The following facts are easily verified.

Fact 3 *1. Each t-sRL-automaton is error preserving.*
2. Each deterministic t-sRL-automaton is correctness preserving.
3. There exist nondeterministic t-sRL-automata which are not correctness preserving.

Observe, using Fact 3 (3), that the language LR_t cannot be accepted by an r-sRL-automaton for any $r < t$.

From the witness languages $L^{\langle t\rangle}$ of Example 1 and LR_t of Example 2 we obtain the following hierarchy results. Here the prefix det- is used to denote deterministic automata.

Corollary 1. *For each suffix $\mathsf{Y} \in \{\mathsf{sRR}, \mathsf{sRL}\}$, and each integer $t \geq 2$,*

(a) $\mathcal{L}(\mathsf{det}\text{-}(t-1)\text{-}\mathsf{Y}) \subset \mathcal{L}(\mathsf{det}\text{-}t\text{-}\mathsf{Y})$ *and* $\mathcal{L}((t-1)\text{-}\mathsf{Y}) \subset \mathcal{L}(t\text{-}\mathsf{Y})$.

(b) $\mathcal{L}(\mathsf{min}\text{-}\mathsf{det}\text{-}(t-1)\text{-}\mathsf{Y}) \subset \mathcal{L}(\mathsf{min}\text{-}\mathsf{det}\text{-}t\text{-}\mathsf{Y})$ *and* $\mathcal{L}(\mathsf{min}\text{-}(t-1)\text{-}\mathsf{Y}) \subset \mathcal{L}(\mathsf{min}\text{-}t\text{-}\mathsf{Y})$.

(c) $\mathcal{L}_{\leq n}(\mathsf{min}\text{-}\mathsf{det}\text{-}(t-1)\text{-}\mathsf{Y}) \subset \mathcal{L}_{\leq n}(\mathsf{min}\text{-}\mathsf{det}\text{-}t\text{-}\mathsf{Y})$ *and* $\mathcal{L}_{\leq n}(\mathsf{min}\text{-}(t-1)\text{-}\mathsf{Y}) \subset \mathcal{L}_{\leq n}(\mathsf{min}\text{-}t\text{-}\mathsf{Y})$ *for each $n > t$.*

Example 3. There exists a 1-sRR-automaton M_{copy} that accepts the language

$$LR_2' := LR_2 \cdot \{a,b,\lambda\} = \{\, c_0wc_1wx \mid w \in \{a,b\}^*, x \in \{a,b,\lambda\} \,\}.$$

M_{copy} is given through the following sequence of meta-instructions:

(1) $(¢c_0c_1\$, \mathsf{Accept})$,
(2) $(¢c_0 \cdot (\{a,b\}^2)^*, a, c_1 \cdot (\{a,b\}^2)^* \cdot a\$)$,
(3) $(¢c_0 \cdot (\{a,b\}^2)^*, b, c_1 \cdot (\{a,b\}^2)^* \cdot b\$)$,
(4) $(¢c_0 \cdot (\{a,b\}^2)^* \cdot c_1 \cdot (\{a,b\}^2)^*, a, \$)$,
(5) $(¢c_0 \cdot (\{a,b\}^2)^* \cdot c_1 \cdot (\{a,b\}^2)^*, b, \$)$,
(6) $(¢c_0 \cdot (\{a,b\}^2)^* \cdot \{a,b\}, a, c_1 \cdot (\{a,b\}^2)^* \cdot \{a,b\} \cdot a\$)$,
(7) $(¢c_0 \cdot (\{a,b\}^2)^* \cdot \{a,b\}, b, c_1 \cdot (\{a,b\}^2)^* \cdot \{a,b\} \cdot b\$)$,
(8) $(¢c_0 \cdot (\{a,b\}^2)^* \cdot \{a,b\} \cdot c_1 \cdot (\{a,b\}^2)^* \cdot \{a,b\}, a, \$)$,
(9) $(¢c_0 \cdot (\{a,b\}^2)^* \cdot \{a,b\} \cdot c_1 \cdot (\{a,b\}^2)^* \cdot \{a,b\}, b, \$)$.

It is easily seen that M_{copy} accepts the language LR_2' working with minimal acceptance. On the other hand, the language LR_2' is not even growing context-sensitive. Assume to the contrary that this language is growing context-sensitive. Then there exists a nondeterministic shrinking two-pushdown automaton A that accepts this language [1]. Consider the language $L_{\text{copy}} := \{\, ww \mid w \in \{a,b\}^* \,\}$. Given an input $x \in \{a,b\}^*$, a shrinking two-pushdown automaton B can simply insert c_0 at the beginning and c_1 somewhere inside the word x, producing the word $c_0x_1c_1x_2$, where the position for the latter insertion is chosen nondeterministically. While doing so the automaton verifies whether x is of even length. In the negative it will reject x, in the positive case it will simulate A for the input $c_0x_1c_1x_2$. This will lead to acceptance if and only if $x_1 = x_2$. Thus, the shrinking two-pushdown automaton B accepts the language L_{copy}. This, however, contradicts the fact that L_{copy} is not growing context-sensitive (see, e.g., [1]).

As a consequence we obtain the following incomparability results.

Theorem 2. *The language classes* $\bigcup_{t\in\mathbb{N}_+} \mathcal{L}(\mathsf{min}\text{-}t\text{-}\mathsf{sRL})$ *and* $\bigcup_{t\in\mathbb{N}_+} \mathcal{L}(t\text{-}\mathsf{sRL})$ *are incomparable under inclusion to the class* CFL *of context-free languages and to the class* GCSL *of growing context-sensitive languages.*

Proof. Already the class $\mathcal{L}(\mathsf{min\text{-}1\text{-}sRR})$ contains a language that is not growing context-sensitive, as shown by the example above. On the other hand, it is shown in [4] that the context-free language $L_2 := \{\, a^nb^n \mid n \geq 0 \,\} \cup \{\, a^nb^m \mid m > 2n \geq 0 \,\}$ is not accepted by any RRW-automaton. The argument is based on the observation that in each cycle an RRW-automaton M for L_2 would have to guess whether to remove a factor of the form a^ib^i or a factor of the form a^ib^{2i} for some integer $i > 0$. Using pumping techniques it can then be shown that M violates the error preserving property. The same argument also works for t-sRL-automata, that is, L_2 is not accepted by any t-sRL-automaton. □

As we have seen above, t-sRL-automata accept some languages that are quite complicated in comparison to the context-free languages. However, we do not yet have a characterization for the expressive power of t-sRL-automata in general. The following result shows that this question has been solved at least for the special case of a single-letter alphabet.

Theorem 3. *Let M be a t-*sRL*-automaton. If $L(M) \subseteq \{a\}^*$, then $L(M)$ is regular.*

Proof. By Theorem 1 we can assume that M is a t-sRR-automaton. Further, as M does not accept any word containing a letter other than a, we can assume that the tape alphabet of M consists of the letter a only.

Assume that M is defined by the meta-instructions

$$I_i := (\text{¢} \cdot E_0^{(i)}, a, E_1^{(i)}, a, E_2^{(i)}, \ldots, E_{s_i-1}^{(i)}, a, E_{s_i}^{(i)} \cdot \$) \quad (1 \leq i \leq r)$$

and $I_0 := (\text{¢} \cdot S_0 \cdot \$, \mathsf{Accept})$, where all $E_j^{(i)}$ and S_0 are regular expressions. We now define another t-sRR-automaton M' through the meta-instruction I_0 and the meta-instructions I_i', $1 \leq i \leq r$, where I_i' is defined as

$$I_i' := (\text{¢} \cdot E_0^{(i)} \cdot E_1^{(i)} \cdot E_2^{(i)} \cdots E_{s_i-1}^{(i)} \cdot E_{s_i}^{(i)}, a^{s_i} \to \lambda, \$).$$

Here $a^{s_i} \to \lambda$ is used as a shorthand for the fact that s_i copies of the symbol a are deleted that are next to each other.

To complete the proof we establish the following two claims.

Claim 1. $L(M') = L(M)$.

Proof. Obviously, M' and M accept the same words in tail computations. Thus, it suffices to show that M' and M execute the same cycles. Assume that $u \vdash_M^c v$ is a possible cycle of M. Then there exists an index $i \in \{1, \ldots, r\}$ such that the tape content $\text{¢} \cdot u \cdot \$$ is transformed into $\text{¢} \cdot v \cdot \$$ by meta-instruction I_i. Hence, u can be factored as $u = u_0 a u_1 a u_2 \ldots u_{s_i-1} a u_{s_i}$ such that $u_j \in E_j^{(i)}$ for all $1 \leq j \leq s_i$ and $v = u_0 u_1 u_2 \ldots u_{s_i-1} u_{s_i}$. It is now immediate that by applying meta-instruction I_i', M' can execute the cycle $u \vdash_{M'}^c v$. Analogously, if $u \vdash_{M'}^c v$ by meta-instruction I_i', then also $u \vdash_M^c v$ by meta-instruction I_i. Thus, the languages $L(M)$ and $L(M')$ coincide. □

Claim 2. $L(M') = L(A)$ for a nondeterministic finite-state acceptor A.

Proof. We present a nondeterministic finite-state acceptor A that, given a word $w = a^n$ as input, accepts if and only if there exists a sequence of cycles

$$w = w_m \vdash_{M'}^c w_{m-1} \vdash_{M'}^c \cdots \vdash_{M'}^c w_1 \vdash_{M'}^c w_0 \in S(M').$$

For each $i = 1, \ldots, m$, there exists a meta-instruction $I_{j_i}' = (\text{¢} \cdot E_{j_i}', a^{s_{j_i}} \to \lambda, \$)$ of M' such that $w_i = w_{i-1} \cdot a^{s_{j_i}}$ and $w_{i-1} \in E_{j_i}'$. Thus, scanning its input tape from left to right, A will simultaneously simulate finite-state acceptors for the languages E_i', $1 \leq i \leq r$, and $S(M')$. When it recognizes a prefix that belongs to $S(M')$, then it decides nondeterministically whether this is the string w_0. In the affirmative it aborts the simulation of the finite-state acceptor for $S(M')$, and guesses an index $k \in \{1, \ldots, r\}$. Then it continues with simulating the finite-state acceptors for the languages E_i' $(1 \leq i \leq r)$ for s_k steps. In case the finite-state acceptor for E_k' is now in a final state, A again guesses an index $k' \in \{1, \ldots, r\}$ and continues, otherwise, it halts and rejects. This process continues until A either rejects, or until the input has been read completely. If ℓ is the latest index guessed, then A accepts if, since passing through a final state of the finite-state acceptor for the language E_ℓ', exactly s_ℓ copies of letter a have been read. It is now immediate that $L(A) = L(M')$. □

Thus, the language $L(M)$ is regular. $\square$

Further, we have at least the following inclusion results.

Theorem 4. $\mathsf{DCFL} \subset \bigcup_{t\in\mathbb{N}_+} \mathcal{L}(\text{min-det-}t\text{-sRL}) \subset \bigcup_{t\in\mathbb{N}_+} \mathcal{L}(\text{min-}t\text{-sRL})$.

Proof. Each deterministic context-free language L is accepted by some right-monotone deterministic RR-automaton M [4]. If M has a window of size k, then it can be simulated by a deterministic k-sRL-automaton M'. M' scans its tape from left to right until it detects a factor that is to be rewritten by M. Then it moves its window back by $k-1$ positions, and deletes the up to k symbols that M deletes in this cycle. By some additional cycles each word from the regular language $S(M)$ can then be reduced to a minimal word. Thus, $\mathsf{DCFL} \subseteq \bigcup_{t\in\mathbb{N}_+} \mathcal{L}(\text{min-det-}t\text{-sRL})$.

The language $L := \{\, a^n b^n c, a^n b^{2n} d \mid n \geq 0 \,\}$, which is not deterministic context-free, is easily seen to be accepted by a deterministic 3-sRL-automaton that works with minimal acceptance. Thus, it follows that $\mathsf{DCFL} \subset \bigcup_{t\in\mathbb{N}_+} \mathcal{L}(\text{min-det-}t\text{-sRL})$.

The language $L := \{\, a^n b^m \mid 0 \leq n \leq m \leq 2n \,\}$ is accepted by the 3-sRL-automaton M that is given through the following meta-instructions:

$$(¢ \cdot a^*, a, \{\lambda\}, b, b^* \cdot \$), \quad (¢ \cdot a^*, a, \{\lambda\}, b, \{\lambda\}, b, b^* \cdot \$), \quad (¢ \cdot \$, \mathsf{Accept}).$$

However, L is not accepted by any deterministic t-sRL-automaton. Assume to the contrary that L is accepted by a deterministic t-sRL-automaton M'. For each integer $n \geq 1$, M' must accept the input $a^n b^{2n}$. However, if n is sufficiently large, then M' cannot do this in a tail computation, that is, the accepting computation of M' on input $a^n b^{2n}$ begins with a cycle of the form $a^n b^{2n} \vdash^c_{M'} u$. This is in particular the case if $n = 2t \cdot |Q|\,!$, where Q is the set of states of M'. By the correctness preserving property $u \in L$, that is, $u = a^{n-i} b^{2n-j}$ for some values of i, j satisfying $i \geq 0$, $j \geq 1$, $i + j \leq t$, and $j \geq 2i$.

Now consider the input of the form $a^n b^n$, on which M' will execute the cycle $a^n b^n \vdash^c_{M'} a^{n-i} b^{n-j}$, as it cannot distinguish between the two inputs $a^n b^{2n}$ and $a^n b^n$. While $a^n b^n \in L$, we see that $a^{n-i} b^{n-j} \notin L$, as $n - j < n - i$ holds, which violates the correctness preserving property. This shows that $\bigcup_{t\in\mathbb{N}_+} \mathcal{L}(\text{min-det-}t\text{-sRL}) \subset \bigcup_{t\in\mathbb{N}_+} \mathcal{L}(\text{min-}t\text{-sRL})$. $\square$

3 Monotonicity

Finally we turn to the notion of monotonicity for t-sRL-automata. Let M be a t-sRL-automaton. A configuration $C = \alpha q \beta$ of M in which a delete operation is to be applied is called a *delete configuration* of M. The number $|\beta|$ is called the *right distance* of C, denoted by $D_r(C)$, and the number $|\alpha|$ is the *left distance* of C, denoted by $D_l(C)$.

We say that a *sequence of delete configurations* $S = (C_1, C_2, \cdots, C_n)$ is *right-monotone* if $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$, that it is *left-monotone* if $D_l(C_1) \geq D_l(C_2) \geq \ldots \geq D_l(C_n)$, and that it is *right-left-monotone* if it is simultaneously right- and left-monotone. It is called *j-right-monotone*, *j-left-monotone* or *j-right-left-monotone* for some integer $j \geq 1$, if it can be partitioned into at most j subsequences $S_1, S_2, \ldots, S_j$ such that each of these subsequences is right-monotone, left-monotone or right-left-monotone, respectively. A computation of M is called *j-right-monotone*, *j-left-monotone* or *j-right-left-monotone* if the corresponding sequence of delete configurations is j-right-monotone, j-left-monotone or j-right-left-monotone, respectively.

An sRL-automaton M is called *t-right-monotone* (*t-left-monotone, t-right-left-monotone*) if it is a t-sRL-automaton and each of its computations is t-right-monotone (t-left-monotone, t-right-left-monotone). We will use the prefixes t-right-mon-, t-left-mon-, and t-right-left-mon- to denote the classes of t-right-monotone, t-left-monotone and t-right-left-monotone sRL-automata, respectively.

For example, the deterministic t-sRR-automaton M_t of Example 2 is t-right-left-monotone. Concerning the j-right-, j-left-, and j-right-left-monotone (standard) RL-automata considered in [12] we have the following result.

Theorem 5. *For each prefix* $\mathsf{Y} \in \{\text{right-left-mon}, \text{right-mon}, \text{left-mon}\}$ *and all integers $j, k \geq 1$, if M is a j-Y-RL-automaton with a window of size k, then there exists a $(j \cdot k)$-Y-sRL-automaton M' such that $L(M) = L(M')$ and* $\mathsf{RS}(M) = \mathsf{RS}(M')$ *hold.*

Proof. Let M be a j-Y-RL-automaton over Σ with a window of size k. Then the sequence of cycles of each computation of M can be divided into j interleaved subsequences that are all Y-monotone. We describe an sRL-automaton M' that simulates the computations of M cycle by cycle. Within its finite-state control M' realizes a buffer of size k that it will use to store the content of the window of M. When M performs a rewrite step $u \to v$, then $|u| \leq k$, and v is obtained from u by deleting up to k symbols of u. Hence, M' can simulate this rewrite step by executing up to k delete steps, each deleting a single symbol. It follows that M' is a k-sRL-automaton, and that $\mathsf{RS}(M') = \mathsf{RS}(M)$. Further, if M is j-Y-monotone, then we consider M' as a $(j \cdot k)$-sRL-automaton, and it is obvious that as such M' is $(j \cdot k)$-Y-monotone. □

From the proof above we see that a (standard) RL-automaton with a window of size k can be simulated by a k-sRL-automaton.

It is known that, for all three types of monotonicity, the language classes $(\mathcal{L}(j\text{-Y-RL}))_{j \geq 1}$ form a strict hierarchy (see [13] Theorem 7 (a)). In fact, this remains true when we restrict our attention to RL-automata with read/write windows of size two. Further, it is shown in [5] that already 2-right-monotone R-automata accept NP-complete languages. In fact, this result extends to right-left-monotonicity, as there are even 2-right-left-monotone R-automata that accept NP-complete languages [6]. However, neither the proof in [5] and nor the proof in [6] gives the size of the window of the R-automaton constructed explicitly (it is rather large due to the encoding used). Hence, we only obtain the following complexity result for sRL-automata.

Corollary 2. *There exists an integer $t \geq 1$ such that* $\mathcal{L}(t\text{-right-left-mon-sRL})$ *contains* NP*-complete languages.*

4 The Gap-Complexity

Let M be a t-sRL-automaton, let $w \in \Sigma^*$ be an input word, and let $w \vdash_M^c w_1 \vdash_M^c \cdots \vdash_M^c w_n$ be an initial part of a computation of M on input w. In each cycle M

deletes up to t symbols from the tape content of the current restarting configuration. Instead of deleting symbols we can replace them by a special symbol $\blacklozenge$, thus obtaining a word w'_i for each $i = 1, \ldots, n$. The word w_i is obtained from w'_i by deleting all $\blacklozenge$-symbols. The *gap-number* of w_i denotes the number of factors from $\blacklozenge^+$ that occur in w'_i. The *gap-number* of the above computation of M is the maximum of the gap-numbers of $w_1, \ldots, w_n$. The *gap-complexity* of M on input $w \in L(M)$ is the minimal gap-number of M over all accepting computations of M on input w. Finally, the *gap-complexity* of M is the maximal gap-complexity of M on input w over all words $w \in L(M)$.

The 1-sRR-automaton M_{copy} for the language LR'_2 of Example 3 has gap-complexity 2, while the deterministic t-sRR-automaton M_t of Example 2 has gap-complexity t. Indeed, the accepting computation of M_t on the input word $w = c_0abac_1abac_2 \ldots c_{t-1}aba$ can be described by the following sequence of words with $\blacklozenge$-symbols:

$$c_0\blacklozenge bac_1\blacklozenge bac_2 \ldots c_{t-1}\blacklozenge ba, c_0\blacklozenge\blacklozenge ac_1\blacklozenge\blacklozenge ac_2 \ldots c_{t-1}\blacklozenge\blacklozenge a, c_0\blacklozenge\blacklozenge\blacklozenge c_1\blacklozenge\blacklozenge\blacklozenge c_2 \ldots c_{t-1}\blacklozenge\blacklozenge\blacklozenge.$$

If a t-sRL-automaton M has gap-complexity c, then each word w of $L(M)$ admits an accepting computation that is c-right-left-monotone. In this way the gap-complexity can be seen as a restriction of the notion of j-right-left-monotonicity. From the examples above we obtain the following hierarchy result.

Proposition 1. *For all $c \in \mathbb{N}_+$, the class of languages accepted by* sRL*-automata with gap-complexity c is properly contained in the class of languages accepted by* sRL*-automata with gap-complexity $c + 1$.*

On the other hand, the t-sRL-automaton of Corollary 2, although being t-right-left-monotone, has unbounded gap-complexity. Thus, having bounded gap-complexity is a much stronger restriction than a bounded degree of monotonicity.

In fact, we have the following positive result on the complexity of languages that are accepted by t-sRL-automata with a bounded gap-complexity.

Theorem 6. *If M is a t-*sRL*-automaton with gap-complexity $c \in \mathbb{N}$, then the membership problem for the language $L(M)$ can be solved in time $O(n^{2c+1})$.*

Proof. Let $w \in \Sigma^*$ be an input word of length n. With M and w we associate a graph $G_M(w) = (V, E)$ as follows. The set V of vertices corresponds to the set of all words over $\Sigma \cup \{\blacklozenge\}$ of length n that can be obtained from w by replacing symbols from Σ by $\blacklozenge$-symbols, and that have gap-number at most c. Within a word of length n there are $\binom{n+1}{2c}$ positions to place c gaps. Hence, we see that the number of vertices V is bounded from above by the number n^{2c}. The vertex w is called the *initial vertex* of $G_M(w)$, and a vertex $v' \in V$ is a *final vertex* if the word $v := \Pi_\Sigma(v')$ (that is, the projection onto Σ^*) belongs to the set $S(M)$.

Now a directed edge leads from a vertex $u' \in V$ to a vertex $v' \in V$, if M can execute the cycle $u \vdash^c_M v$, where u and v are obtained from u' and v', respectively, by deleting the $\blacklozenge$-symbols. As the gap-complexity of M is bounded by the constant c, each of the up to t delete operations of M starting from the configuration $q_0\cent u\$$ has to be applied at either the left or the right border of one of the c gaps in u'.

It follows that the outdegree of each vertex is bounded from above by the number $(2c)^t$, that is, $G_M(w)$ contains $O(n^{2c})$ many edges. Thus, the graph $G_M(w)$ can be computed from w in time $O(n^{2c+1})$.

Now $w \in L(M)$ if and only if a final vertex v' can be reached from the initial vertex w in the graph $G_M(w)$. This can be checked in time $O(|V| + |E|)$, that is, in time $O(n^{2c})$. □

Thus, for t-sRL-automata, bounded gap-complexity separates those automata that accept feasible languages from those that accept NP-complete languages.

Corollary 3. *It is decidable in polynomial time whether a t-sRL-automaton M accepts a given word w with gap-complexity at most c.*

Proof. As in the proof of Theorem 6 we can associate a graph $G_M(w, c)$ with a t-sRL-automaton M, a positive integer c, and a word w. Now M has an accepting computation for input w with gap-complexity at most c if and only if a final vertex v' can be reached from the initial vertex w in the graph $G_M(w, c)$. As this can be checked in polynomial time, this proves our result. □

For a t-sRL-automaton M and an integer $c \geq 1$ we can thus define the language $L_{\text{gap}}(M, c) := \{ w \in L(M) \mid M \text{ accepts } w \text{ with gap-complexity at most } c \}$. From Corollary 3 we see that the membership problem for languages of this form is decidable in polynomial time.

5 Analyzing Regular Languages

It is straightforward to see that each finite language is accepted by some t-sRL-automaton working with minimal acceptance. However, as seen in Example 1 the value of t depends on the particular language considered. Also all regular languages are accepted by t-sRL-automata. In fact, we have the following result, where an sRL-automaton M is said to have the $\mathsf{mr}(k)$*-property* if it executes at most k right-move operations in any cycle or tail.

Theorem 7. *A language L is regular if and only if there exists an integer $t \geq 1$ and a deterministic t-sRL-automaton M with the $\mathsf{mr}(t)$-property such that M accepts L working with minimal acceptance.*

Proof. For each regular language $L \subseteq \Sigma^*$, there exists a finite-state acceptor $A = (Q, \Sigma, q_0, F, \delta)$ that recognizes L. Let n be the number of states of A. Reading a prefix of a word from Σ^* of length at least n, the automaton A must pass some state at least twice. Hence, for each word z of length n, there exist words $u, v, w \in \Sigma^*$ such that $z = uvw$, $v \neq \lambda$, and the automaton A is in the same state after reading both prefixes u and uv. Hence, for each word $x \in \Sigma^*$, $uvwx \in L$ if and only if $uwx \in L$. Now a word $v \in \Sigma^{\leq n-1}$ is called *cycle-free* if $\delta(q_0, x) \neq \delta(q_0, xy)$ holds for all prefixes x and xy of v satisfying $y \neq \lambda$.

Take $t := n$. We construct a t-sRL-automaton M for L as follows. M will have the single accepting meta-instruction $(¢ \cdot L_{\min} \cdot \$, \mathsf{Accept})$. Each word $z \in \Sigma^n$ has a unique factorization of the form $z = uvaw$ such that uv is cycle-free, $a \in \Sigma$, and $\delta(q_0, u) = \delta(q_0, uva)$. If $z \cdot \Sigma^* \cap L \neq \emptyset$, then we introduce the meta-instruction

$$(\text{¢} \cdot u, b_1, \{\lambda\}, b_2, \{\lambda\}, \ldots, \{\lambda\}, b_r, \Sigma^* \cdot \$),$$

where $va = b_1 b_2 \ldots b_r$ $(b_1, \ldots, b_r \in \Sigma)$. Finally, for $z \in L \cap \Sigma^{\leq n-1}$, if z is not minimal in L, then we introduce a meta-instruction of the form

$$(\text{¢} \cdot z_1, a_1, z_2, a_2, z_2, \ldots, z_r, a_r, z_{r+1} \cdot \$),$$

where $z = z_1 a_1 z_2 a_2 z_2 \ldots z_r a_r z_{r+1}$, $z_1 z_2 \ldots z_r z_{r+1} \in L_{\min}$, and $a_1, a_2, \ldots, a_r \in \Sigma$. It can be verified that

– if $|z| \leq n - 1$, then $z \in L$ if and only if $z \in L(M)$,
– if $|z| \geq n$, then there exists a word $z' \in \Sigma^*$, $|z'| < |z|$, such that $z \vdash_M^c z'$ and $z \in L$ if and only if $z' \in L$.

From this it follows that $L(M) = L$. Further, it is easily seen that M is deterministic, and that in any cycle or tail it executes at most t move-right steps.

Conversely, let M be a deterministic t-sRL-automaton with the mr(t)-property working with minimal acceptance. We need to show that $L(M)$ is a regular language. It is not hard to see that a finite automaton M_f with a buffer of the size $2t$ can simulate the computations of M. The buffer serves as a storage for the prefix of the tape content of M during the simulation of the individual cycles of M. □

The above result implies in particular that all regular languages are accepted by sRL-automata with bounded gap-complexity. It should be interesting to classify regular languages with respect to the smallest value t for which they are accepted in this way, and with respect to the size of the description of these automata.

Acknowledgement. The authors wish to thank Hartmut Messerschmidt for many valuable discussions concerning the notions and results of this paper.

References

1. G. Buntrock, F. Otto. Growing context-sensitive languages and Church-Rosser languages. *Information and Computation* 141 (1998) 1–36.
2. T. Holan. Dependency analyser configurable by measures. In: P. Sojka, I. Kopeček, K. Pala (eds.), *TSD 2002, Proc.*, *LNCS 2448*, Springer, Berlin, 2002, 81–88.
3. T. Holan, V. Kuboň, K. Oliva, M. Plátek. Two Useful Measures of Word Order Complexity. In: A. Polguere, S. Kahane (eds.), *Workshop 'Processing of Dependency-Based Grammars,' COLING'98, Proc.*, University of Montreal, 1998, 21–28.
4. P. Jančar, F. Mráz, M. Plátek, J. Vogel. On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics* 4 (1999) 287–311.
5. T. Jurdziński, F. Otto, F. Mráz, M. Plátek. On the complexity of 2-monotone restarting automata. In: C.S. Calude, E. Calude, M.J. Dinneen (eds.), *DLT 2004, Proc.*, *LNCS 3340*, Springer, Berlin, 2004, 237–248.
6. T. Jurdziński, F. Otto, F. Mráz, M. Plátek. On the complexity of 2-monotone restarting automata. *Mathematische Schriften Kassel* 4/04, Universität Kassel, 2004.
7. M. Lopatková, M. Plátek, V. Kuboň. Modeling syntax of free word-order languages: Dependency analysis by reduction. In: V. Matoušek, P. Mautner, T. Pavelka (eds.), *TSD 2005, Proc.*, *LNCS 3658*, Springer, Berlin, 2005, 140–147.

8. M. Lothaire. *Combinatorics on Words.* Encyclopedia of Mathematics, Vol. 17, Addison-Wesley, Reading, 1983.
9. F. Otto. Restarting automata and their relations to the Chomsky hierarchy. In: Z. Esik, Z. Fülöp (eds.), *DLT 2003, Proc.*, *LNCS 2710*, Springer, Berlin, 2003, 55–74.
10. F. Otto. Restarting Automata - Notes for a Course at the 3rd International PhD School in Formal Languages and Applications. *Mathematische Schriften Kassel* 6/04, Universität Kassel, 2004.
11. M. Plátek. Two-way restarting automata and j-monotonicity. In: L. Pacholski, P. Ružička (eds.), *SOFSEM 2001, Proc.*, *LNCS 2234*, Springer, Berlin, 2001, 316–325.
12. M. Plátek, F. Otto, F. Mráz. Restarting automata and variants of j-monotonicity. In: E. Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil (eds.), *DCFS 2003, Proc.*, MTA SZTAKI, Budapest, 2003, 303–312.
13. M. Plátek, F. Otto, F. Mráz, T. Jurdziński. Restarting automata and variants of j-monotonicity. *Mathematische Schriften Kassel* 9/03, Universität Kassel, 2003.
14. S. H. von Solms. The characterization by automata of certain classes of languages in the context sensitive area. *Information and Control* 27 (1975) 262–271.