On the Descriptional Complexity of Simple RL-Automata[⋆]

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

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

[2] 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

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 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. We focus on the *descriptional complexity* of these automata, establishing two *complexity measures* that are both based on the description of t-sRL-automata in terms of so-called *meta-instructions*. We present some hierarchy results as well as a non-recursive trade-off between deterministic 2-sRL-automata and finite-state acceptors.

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., [5]).

Here we continue the study of a new variant of the restarting automaton, the so-called *simple* RL-*automaton* (sRL-automaton) [9], 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, which yields an infinite hierarchy of automata and language classes. In [9] we studied the number of gaps generated during a reduction as a dynamic complexity measure for t-sRL-automata. A bounded number of gaps implies that only feasible languages are accepted, that is, languages that are recognizable in polynomial time, while with an unbounded number of gaps these automata accept NP-complete languages.

Here we concentrate on the descriptional complexity of sRL-automata. We introduce two descriptional complexity measures, the number of meta-instructions, which is a fairly rough measure, and the descriptional size. We establish an infinite

[⋆] F. Mráz and M. Plátek 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. F. Mráz was also partially supported by the Grant Agency of Charles University under Grant-No. 358/2006/A-INF/MFF.

hierarchy with respect to the first measure, and we show that even deterministic 2-sRL-automata allow very succinct representations of (certain) regular languages by presenting a non-recursive trade-off. Observe that deterministic 1-sRR-automata, that is, deterministic 1-sRL-automata that do not use move-left instructions, only accept regular languages. In [3] exponential trade-offs between nondeterministic and deterministic finite-state acceptors and deterministic 1-sRR-automata are given.

The paper is structured as follows. After introducing the simple RL-automaton in Section 2 and restating some of its basic properties, we show that deterministic t-sRL-automata are as expressive as nondeterministic t-sRL-automata that are correctness preserving. Then we define our descriptional complexity measures in Section 3. We illustrate them by various examples, and we present an infinite hierarchy with respect to our first measure. Then in Section 4 we establish the announced non-recursive trade-off between deterministic 2-sRL-automata and finite-state acceptors.

2 The t-sRL-Automaton

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 [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* and *move-left steps*, which shift the window one position to the right or to the left, respectively, thereby 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.

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 $q_0¢w\$$.

Each part of a computation of an sRL-automaton M from a restarting configuration to the next restarting configuration is called a *cycle*. The part after the last restart operation is called the *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* $\mathsf{RS}(M) := (\Sigma^*, \vdash_M^c, S(M))$ that is induced by M. Observe that, for each $w \in \Sigma^*$, we have $w \in L(M)$ if and only if $w \vdash_M^{c^*} v$ holds for some word $v \in S(M)$.

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. 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 words Σ^*, we consider the well-founded partial ordering $\leq$ that is defined by $u \leq v$ if and only if u is a scattered subword of v. By $<$ we denote the proper part of $\leq$.

For $L \subseteq \Sigma^*$, let $L_{\min} := \{ w \in L \mid u < w$ 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., [6]). 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.

An sRL-automaton working 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.

Theorem 1. [9] *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.

Based on Theorem 1 we can describe a t-sRL-automaton by meta-instructions of the form $(\text{¢} \cdot E_0, a_1, E_1, a_2, E_2, \ldots, E_{s-1}, a_s, E_s \cdot \$)$, 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 $q_0\text{¢}w\$$, 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:

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

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.

For each integer $n \in \mathbb{N}_+$, we consider the finite approximation of order $(n+1) \cdot t$ of the language LR_t which is defined as follows:

$$LR_t^{(n)} := \{\, c_0wc_1wc_2 \ldots c_{t-1}w \mid w \in \{a,b\}^*,\ |w| \leq n \,\}.$$

A t-sRR-automaton $M_t^{(n)}$ for $LR_t^{(n)}$, also working with minimal acceptance, is easily obtained from M_t by taking the following sequence of meta-instructions, where $\Sigma_0^{\leq n-1} := \{\, w \in \Sigma_0^* \mid |w| \leq n-1 \,\}$:

$$\begin{array}{l}
(1)\ (¢c_0, a, \Sigma_0^{\leq n-1} \cdot c_1, a, \Sigma_0^{\leq n-1} \cdot c_2, \ldots, \Sigma_0^{\leq n-1} \cdot c_{t-1}, a, \Sigma_0^{\leq n-1} \cdot \$),\\
(2)\ (¢c_0, b, \Sigma_0^{\leq n-1} \cdot c_1, b, \Sigma_0^{\leq n-1} \cdot c_2, \ldots, \Sigma_0^{\leq n-1} \cdot c_{t-1}, b, \Sigma_0^{\leq n-1} \cdot \$),\\
(3)\ (¢c_0c_1 \ldots c_{t-1}\$, \mathsf{Accept}).
\end{array}$$

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 $v \notin L(M)$.*

It is rather obvious that each t-sRL-automaton is error preserving, and that all deterministic t-sRL-automata are correctness preserving. On the other hand, one can easily construct examples of nondeterministic t-sRL-automata that are not correctness preserving.

Concerning the parameter t and the relation of t-sRL-automata to other language classes the following results have been obtained.

Theorem 2. [9] *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* $n \geq t$*.*

Theorem 3. [9]

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

(b) *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.*

Correctness preserving nondeterministic t-sRR-automata are strictly more expressive than deterministic t-sRR-automata. For example, the language $L := \{\, a^n b^n c, a^n b^{2n} d \mid n \geq 0 \,\}$ is accepted by the 3-sRR-automaton M that is given through the following meta-instructions:

$$(1)\ (\text{¢}, a, a^*, b, b^* \cdot c \cdot \$),\ (2)\ (\text{¢}, a, a^*, b, \{\lambda\}, b, b^* \cdot d \cdot \$),\ (3)\ (\text{¢} \cdot (c+d) \cdot \$, \mathsf{Accept}).$$

If M should select the wrong meta-instruction, then this is recognized at the right sentinel, and then M simply halts and rejects. Thus, M is correctness preserving. On the other hand, it is easily shown that L cannot be accepted by a deterministic sRR-automaton. Surprisingly, however, we have the following equivalence for sRL-automata.

Theorem 4. *For each correctness preserving* t-sRL*-automaton* M*, there exists a deterministic* t-sRL*-automaton* M' *such that* $L(M') = L(M)$*.*

Proof. Let M be a correctness preserving t-sRL-automaton that is given through meta-instructions $I_1, \ldots, I_i$. We will describe a deterministic t-sRL-automaton M' recognizing the same language as M. First, for each $j = 1, \ldots, i$, we construct a finite-state acceptor A_j for the set of words to which meta-instruction I_j is applicable. The automaton M' will then proceed as follows:

1. M' scans the current word w on its tape from left to right simulating all the acceptors $A_1, \ldots, A_i$ in parallel. At the right sentinel M' knows which meta-instructions of M are applicable to the current word. If none is applicable, then M' halts and rejects; if one of the applicable meta-instructions is accepting, then M' halts and accepts. Otherwise, any correct application of any of the applicable meta-instructions will yield a word w' such that $w' \in L(M)$ if and only if $w \in L(M)$, as M is correctness preserving. Thus, M' simply chooses one of the applicable meta-instructions, e.g., the one with the smallest index. By I we denote this meta-instruction.
2. M' simulates an application of I on its current tape content.

It remains to show how M' can simulate an application of I to the configuration q_0¢$w\$$. Let $w = y_1 \cdots y_n$, where $y_1, \ldots, y_n \in \Sigma$, and assume that $I =$

$(¢ \cdot E_0, x_1, E_1, x_2, E_2, \ldots, E_{s-1}, x_s, E_s \cdot \$)$, where $1 \leq s \leq t$. M' must determine a factorization of the form $w = v_0x_1v_1x_2 \ldots v_{s-1}x_sv_s$ such that $v_i \in E_i$ for all $i = 0, \ldots, s$, and remove the symbols $x_1, x_2, \ldots, x_s$. As w may have many such factorizations, M' must choose one of them deterministically. For this task M' will use finite-state acceptors $M_1, \ldots, M_s$ and $M_1^R, \ldots, M_s^R$, which accept the following regular languages:

$$\begin{array}{llr}
L(M_1) = E_0 \cdot x_1, & (E_1 \cdot x_2 \cdot E_2 \cdot x_3 \cdots E_{s-1} \cdot x_s \cdot E_s)^R = & L(M_1^R), \\
L(M_2) = E_1 \cdot x_2, & (E_2 \cdot x_3 \cdots E_{s-1} \cdot x_s \cdot E_s)^R = & L(M_2^R), \\
\vdots & & \vdots \\
L(M_{s-1}) = E_{s-2} \cdot x_{s-1}, & (E_{s-1} \cdot x_s \cdot E_s)^R = & L(M_{s-1}^R), \\
L(M_s) = E_{s-1} \cdot x_s, & (E_s)^R = & L(M_s^R).
\end{array}$$

After step (1) above (that is, when choosing the meta-instruction I), M' is at the right sentinel. Now it scans its tape again, this time from right to left, thereby simulating the finite-state acceptors $M_1^R, \ldots, M_s^R$ in parallel. For each $0 \leq j \leq s$ and $1 \leq \ell \leq n$, let $q(j, \ell)$ denote the state of M_j^R after reading the word $y_n \ldots y_{\ell+1}$. When reaching the left sentinel, M' changes direction again. Now, while moving to the right, M' simulates the finite-state acceptor M_1. Simultaneously, it recomputes the internal states of all the acceptors $M_1^R, \ldots, M_s^R$ on the respective tape symbol, that is, it runs these acceptors in reverse. This it can do due to the following technical result from [1] (pages 212–213).

Lemma 1. *Let A be a deterministic finite-state acceptor. For each word x and each integer i, $1 \leq i \leq |x|$, let $q_A(x, i)$ be the internal state of A after processing the prefix of length i of x. Then there exists a deterministic two-way finite-state acceptor A' such that, for each input x and each $i \in \{2, 3, \ldots, |x|\}$, if A' starts its computation on x in state $q_A(x, i)$ with its head on the i-th symbol of x, then A' finishes its computation in state $q_A(x, i-1)$ with its head on the $(i-1)$-th symbol of x. During this computation A' only visits (a part of) the prefix of length i of x.*

As meta-instruction I is applicable to the configuration $q_0¢w\$$, w belongs to the set $E_0 \cdot x_1 \cdot E_1 \cdot x_2 \cdot E_2 \cdot x_3 \cdots E_{s-1} \cdot x_s \cdot E_s$. Hence, there is a smallest index ℓ_1 such that $y_1 \ldots y_{\ell_1} \in L(M_1)$ and $y_{\ell_1+1} \ldots y_n \in [L(M_1^R)]^R$. That is, after scanning $y_1 \ldots y_{\ell_1}$, the finite-state acceptor M_1 is in an accepting state, and simultaneously $q(1, \ell_1)$ is an accepting state of M_1^R. On reaching this position, M' deletes $y_{\ell_1} = x_1$, aborts the simulations of M_1 and M_1^R, and starts to simulate M_2 from its initial state. Now M' looks for an index $\ell_2 > \ell_1$ such that M_2 is in an accepting state after processing $y_{\ell_1+1} \ldots y_{\ell_2}$, and $q(2, \ell_2)$ is an accepting state of M_2^R. Once this position is reached, M' deletes the symbol $y_{\ell_2} = x_2$, aborts the simulations of M_2 and M_2^R, and starts to simulate M_3. This process is then continued for $i = 3, 4, \ldots, s$. In this way, M' deletes s symbols $y_{\ell_1}, \ldots, y_{\ell_s}$ such that $y_1 \ldots y_{\ell_1-1} \in E_0$, $y_{\ell_1} = x_1$, $y_{\ell_1+1} \cdots y_{\ell_2-1} \in E_1$, $y_{\ell_2} = x_2$, $\ldots$, $y_{\ell_s} = x_s$, and $y_{\ell_s+1} \cdots y_n \in E_s$.

It is easy to see that the t-sRL-automaton M' constructed in the way described above is deterministic, and that it accepts the same language as the given t-sRL-automaton M. □

3 Complexity Measures for sRL-Automata

A t-sRL-automaton M can be interpreted as a description of the language $L(M)$. Hence, the question about the succinctness of this description in comparison to other descriptions of the same language arises. Thus, we need to introduce a measure for the size of a t-sRL-automaton.

In [3] the descriptional complexity of various types of deterministic restarting automata is investigated. There the number of instructions in the transition relation of a restarting automaton is taken as the size of that automaton, that is, for an sRL-automaton M this would yield the number $\text{size}_\delta(M) := |Q| \cdot (|\Sigma| + 2) \cdot \mu$, where μ denotes the maximal degree of nondeterminism that M has in any situation. However, the description of sRL-automata in terms of transition relations is rather cumbersome. Therefore we prefer to consider measures that are based on descriptions of sRL-automata in terms of meta-instructions.

Definition 3. *Let M be a t-sRL-automaton that is given through meta-instructions $I_1, \ldots, I_r$. Then $\text{size}_I(M) := r$ is called the* instruction size *of M.*

The t-sRL-automata M_t and $M_t^{(n)}$ of Example 2 have instruction size 3. Obviously, each regular language L can be accepted by a restarting automaton of instruction size 1 that is given through the single meta-instruction $(¢ \cdot L \cdot \$, \mathsf{Accept})$. It is easy to construct a sequence of languages $L_1, L_2, \ldots$ with growing alphabets such that any t-sRL-automaton recognizing L_i has instruction size at least i. However, we have a sequence of 3-sRL-automata with growing instruction size and a fixed finite alphabet with this property.

Let $\Sigma := \{a, b\}$, let $i > 1$, and let $w_j := a^j b^{i+1-j}$ $(1 \leq j \leq i)$. For each $j \in \{1, \ldots, i\}$, let E^i_j denote the language $E^i_j := \{w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_i\} \subset \Sigma^{i+1}$, and let M_i be the 3-sRL-automaton with input alphabet $\Sigma := \{a, b\}$ that is given through the following meta-instructions:

1. $(¢ \cdot (w_1 \cdot a^* \cdot w_1 + \ldots + w_i \cdot a^* \cdot w_i + w_1 \cdot b^+ \cdot E^i_1 + \ldots + w_i \cdot b^+ \cdot E^i_i) \cdot \$, \mathsf{Accept})$,
2. $(¢ \cdot w_j \cdot a^*, a, \{\lambda\}, b, b^* \cdot w_j \cdot \$)$, $\quad j = 1, \ldots, i$,
3. $(¢ \cdot w_j \cdot a^*, a, \{\lambda\}, b, \{\lambda\}, b, b^* \cdot E^i_j \cdot \$)$, $\quad j = 1, \ldots, i$.

Obviously, $\text{size}_I(M_i) = 2i + 1$, and it is easily verified that

$$L(M_i) = \bigcup_{j=1}^{i} \{\, w_j a^n b^m w_j \mid n \geq m \geq 0 \,\} \,\cup \bigcup_{\substack{j,k=1 \\ j \neq k}}^{i} \{\, w_j a^n b^m w_k \mid m > 2n \geq 0 \,\}.$$

On the other hand, we have the following lower bound result.

Lemma 2. *If M is an* sRL*-automaton for $L(M_i)$, then $\text{size}_I(M) \geq 2i + 1$.*

Proof. Let $i > 1$ be an integer, and let M be an sRL-automaton recognizing the language $L(M) = L(M_i)$. Let $w_{k,l,\mu,\nu} := w_k a^\mu b^\nu w_l$, let p be the number of states of M, and let $n := p\,!$. None of the words $w_{k,k,n,n}$ $(1 \leq k \leq i)$ can be accepted by M

in a tail computation, as otherwise it could be shown by using pumping techniques that M will then also accept certain words which do not belong to $L(M_i)$.

Each meta-instruction which is used in an accepting computation on an input $w_{k,l,\mu,\nu} \in L(M_i)$ deletes only some of the symbols a and b in the middle. Here the fact is used that all words w_j $(1 \leq j \leq i)$ have the same length, and so neither the prefix w_k nor the suffix w_l can be converted into another word w_j for any index j by applying deletions.

If there are less than i meta-instructions that are used for accepting all words of the form $w_{k,k,cn,cn}$ $(1 \leq k \leq i,\ c \geq 1)$, then at least one of them applies to two different words $w_{k,k,cn,cn}$ and $w_{l,l,cn,cn}$, $l \neq k$. Hence, this instruction cannot distinguish between $w_k a^{cn} b^{cn} w_k$ and $w_l a^{cn} b^{cn} w_l$. As neither the prefixes nor the suffixes w_l and w_k are affected by this instruction, we see that this instruction also applies to the word $w_l a^{cn} b^{cn} w_k$.

Each meta-instruction which is used in an accepting computation for a word of the form word $w_l a^{cn} b^{cn} w_l$ (c a large constant) deletes at least as many symbols b as a. Now assume that, after some cycles of an accepting computation starting with the word $w_l a^{cn} b^{cn} w_l$, the number α of symbols a deleted and the number β of symbols b deleted satisfy the condition $\beta = \alpha + m$ for some integer m satisfying $m \geq n = p!$. Thus, $w_l a^{cn} b^{cn} w_l$ is reduced to a word of the form $w_l a^{cn-\alpha} b^{cn-\alpha-m} w_l$. The restarting automaton M cannot distinguish between $w_l a^{cn} b^{cn} w_l$ and $w_l a^{cn} b^{cn+n} w_l \notin L(M_i)$. However, by applying the same sequence of cycles to the latter word, M will derive the word $w_l a^{cn-\alpha} b^{cn-\alpha+n-m} w_l$, which belongs to $L(M_i)$, as $m \geq n$. This contradicts the Error Preserving Property.

Hence, for all sequences of cycles starting with a word of the form $w_l a^{cn} b^{cn} w_l$, the number α of symbols a deleted and the number β of symbols b deleted satisfy the restiction $\alpha \leq \beta < \alpha + n$. As seen above the word $w_l a^{cn} b^{cn} w_k$ can also be processed by the same sequence of cycles, and the same is true for the word $w_l a^{cn} b^{2cn} w_k \notin L(M_i)$. However, after a sufficient number of cycles $\alpha > n$ is obtained, and therewith $\beta < \alpha + n \leq 2\alpha$ holds. Hence, the resulting word is $w_l a^{cn-\alpha} b^{2cn-\beta} w_k$, which belongs to $L(M_i)$, again contradicting the Error Preserving Property.

It follows that, for each value of $j \in \{1, \ldots, i\}$, there is at least one meta-instruction with prefix and suffix w_j that is involved in the accepting computations of M for the words of the form $w_{k,k,cn,cn}$ $(1 \leq k \leq i)$. Thus, there are at least i different meta-instructions of this form.

Essentially the same method also works for those meta-instructions that are used in accepting computations for words of the form $w_j a^n b^m w_k$ $(m > 2n)$. At least i different meta-instructions must be used in these computations. As they differ from the meta-instructions above, and as M needs at least one accepting meta-instruction, we see that $\text{size}_I(M) \geq 2i + 1$ holds. □

The instruction size does not allow to distinguish 'complicated' regular languages from 'simple' ones. Therefore we now define a finer complexity measure for sRL-automata.

Definition 4. *We measure the size of a t-sRL-automaton M through the size of its description in terms of meta-instructions. As meta-instructions contain regular* *expressions, we first have to assign a size to each regular expression. Let E be a regular expression over Σ. Then its size* $\mathrm{size}(E)$ *is defined inductively as follows:*

$$\begin{array}{lcl}
E = \lambda & : & \mathrm{size}(E) := 1, \\
E = a_1 a_2 \dots a_n \ (a_i \in \Sigma,\ n \geq 1) & : & \mathrm{size}(E) := n, \\
E = E_1 \cdot E_2 & : & \mathrm{size}(E) := \mathrm{size}(E_1) + \mathrm{size}(E_2), \\
E = E_1 \cup E_2 & : & \mathrm{size}(E) := \mathrm{size}(E_1) + \mathrm{size}(E_2) + 1, \\
E = E_1^* & : & \mathrm{size}(E) := \mathrm{size}(E_1) + 1, \\
E = E_1^n \ (n \geq 2) & : & \mathrm{size}(E) := \mathrm{size}(E_1) + \log(n), \\
E = E_1^{\leq n} \ (n \geq 2) & : & \mathrm{size}(E) := \mathrm{size}(E_1) + \log(n) + 1.
\end{array}$$

Now the size of a meta-instruction I of M is defined as follows:

$$\begin{array}{lcl}
I = (¢ \cdot E \cdot \$, \mathsf{Accept}) & : & \mathrm{size}(I) := \mathrm{size}(E) + 3, \\
I = (E_0, u_1, E_1, u_2, E_2, \dots, E_{s-1}, u_s, E_s) & : & \mathrm{size}(I) := \sum_{i=0}^{s} \mathrm{size}(E_i) + s.
\end{array}$$

Finally, if M is given through the meta-instructions $I_1, I_2, \dots, I_m$, then

$$\mathrm{size}(M) := \sum_{i=1}^{m} \mathrm{size}(I_i).$$

We illustrate this definition through an example.

Example 3. The t-sRL-automaton M_t of Example 2 has size $13\,t + 7$, as its accepting meta-instruction has size $t + 3$, and each of its other two meta-instructions has size $6\,t + 2$. Observe that $\mathrm{size}(\Sigma_0^*) = \mathrm{size}((a \cup b)^*) = 4$.

On the other hand, the t-sRL-automaton $M_t^{(n)}$ from the same example has size $(13 + 2 \cdot \log(n-1)) \cdot t + 7$, as each of its two deleting meta-instructions has size $(6 + \log(n-1)) \cdot t + 2$. Observe that $\mathrm{size}(\Sigma_0^{\leq n-1}) = \mathrm{size}((a \cup b)^{\leq n-1}) = 4 + \log(n-1)$.

Finally, the 3-sRL-automaton M_i of Lemma 2 has size $2i^3 + 8i^2 + 28i + 2$.

If a t-sRL-automaton M is given through a sequence of meta-instructions $I_1, I_2, \dots, I_m$, then the transition relation for M can simply be constructed by deriving nondeterministic finite-state acceptors for all the regular expressions occurring in these meta-instructions. Thus, $\mathrm{size}_\delta(M)$ will be bounded by the combined size of all these acceptors. Essentially, for all regular expressions E, the size of the resulting finite-state acceptor coincides with the number $\mathrm{size}(E)$ with the one exception that a finite-state acceptor for E^m has size $\mathrm{size}(E) \cdot m$ instead of $\mathrm{size}(E) + \log(m)$. Thus, we see that $\mathrm{size}(M)$ and $\mathrm{size}_\delta(M)$ are related to each other by a logarithmic factor.

We close this section with a couple of further examples.

Example 4. Let $\Sigma := \{a, b\}$. We consider the language L_{pal} and its finite approximations $L_{\mathrm{pal}}^{(n)}$ $(n \in \mathbb{N}_+)$ that are defined as follows:

$$L_{\mathrm{pal}} := \{\, ww^R \mid w \in \Sigma^* \,\} \text{ and } L_{\mathrm{pal}}^{(n)} := \{\, ww^R \mid w \in \Sigma^{\leq n} \,\}.$$

Then L_{pal} is accepted by the deterministic 2-sRL-automaton M_{pal} that is specified by the following three meta-instructions:

$$(1) \quad (¢, a, (\Sigma^2)^*, a, \$), \qquad (2) \quad (¢, b, (\Sigma^2)^*, b, \$), \qquad (3) \quad (¢ \cdot \$, \mathsf{Accept}),$$

and $L_{\text{pal}}^{(n)}$ is accepted by the deterministic 2-sRL-automaton $M_{\text{pal}}^{(n)}$ that is specified by the following three meta-instructions:

$$(1) \quad (\mathrm{¢}, a, (\Sigma^2)^{\leq n-1}, a, \$), \quad (2) \quad (\mathrm{¢}, b, (\Sigma^2)^{\leq n-1}, b, \$), \quad (3) \quad (\mathrm{¢} \cdot \$, \mathsf{Accept}).$$

Observe that $\text{size}(M_{\text{pal}}) = 21$, while $\text{size}(M_{\text{pal}}^{(n)}) = 21 + 2 \cdot \log(n-1)$.

The situation is very different for the following example language.

Example 5. Let $\Sigma := \{a, b\}$. We consider the language L_{copy} and its finite approximations $L_{\text{copy}}^{(n)}$ $(n \in \mathbb{N}_+)$ that are defined as follows:

$$L_{\text{copy}} := \{ ww \mid w \in \Sigma^* \} \text{ and } L_{\text{copy}}^{(n)} := \{ ww \mid w \in \Sigma^{\leq n} \}.$$

Then $L_{\text{copy}}^{(n)}$ is accepted by the 2-sRR-automaton $M_{\text{copy}}^{(n)}$ that is specified by the following meta-instructions:

$$(\mathrm{¢}, c, \Sigma^m, c, \Sigma^m \cdot \$) \text{ for all } c \in \Sigma \text{ and } m = 0, 1, \ldots, n-1, \quad (\mathrm{¢} \cdot \$, \mathsf{Accept}),$$

which are of combined size $O(\sum_{m=1}^{n} \log m) = O(n \cdot \log n)$, while it can be shown that L_{copy} is not accepted by any 2-sRL-automaton at all.

The following language has been considered before in various contexts [8, 11].

Example 6. For $t \geq 1$, let $\Sigma_t := \{a_1, a_2, \ldots, a_t\}$. The language LE_t and its finite approximations $LE_t^{(n)}$ $(n \in \mathbb{N}_+)$ are defined as follows:

$$\begin{aligned} LE_t &:= \{ w \in \Sigma_t^* \mid |w|_{a_1} = |w|_{a_2} = \ldots = |w|_{a_t} \} \text{ and} \\ LE_t^{(n)} &:= \{ w \in \Sigma_t^* \mid |w|_{a_1} = |w|_{a_2} = \ldots = |w|_{a_t} \leq n \}. \end{aligned}$$

A deterministic t-sRR-automaton M_{LE_t} for LE_t is given through the following meta-instructions, where π varies over all permutations of the set $\{1, 2, \ldots, t\}$:

$$\begin{aligned} &(1.\pi) \ (\mathrm{¢}, a_{\pi(1)}, a_{\pi(1)}^*, a_{\pi(2)}, \{a_{\pi(1)}, a_{\pi(2)}\}^*, a_{\pi(3)}, \ldots, (\Sigma_t \smallsetminus \{a_{\pi(t)}\})^*, a_{\pi(t)}, \Sigma_t^* \cdot \$), \\ &(2) \quad (\mathrm{¢} \cdot \$, \mathsf{Accept}). \end{aligned}$$

In each cycle M_{LE_t} simply deletes the first occurrence of each of the t letters of Σ_t. Obviously, the size of M_{LE_t} is $O(t^2 \cdot t!)$.

In order to construct a t-sRR-automaton $M_{LE_t}^{(n)}$ for the finite approximation $LE_t^{(n)}$ of LE_t, we must ensure that no string of length exceeding the number $n \cdot t$ is accepted by changing the regular constraints of the meta-instructions of M_{LE_t} accordingly. This involves the incorporation of a counter that ensures that $M_{LE_t}^{(n)}$ rejects each tape content that exceeds this length bound. Accordingly, the size of the description of $M_{LE_t}^{(n)}$ will be even larger than the size of M_{LE_t}.

4 A Non-Recursive Trade-off

Let T be a single-tape Turing machine. Then the language VALC(T) of all valid computations of T consists of all words of the form $\omega_0\#\omega_1\#\dots\#\omega_n\#$, where ω_0 is an initial configuration of T, ω_n is an accepting configuration of T, and ω_{i+1} is an immediate successor configuration of the configuration ω_i for all $0 \le i \le n-1$. Each configuration ω_i is of the form $t_0t_1\dots t_{j-1}qt_jt_{j+1}\dots t_\ell$, where $t_0t_1\dots t_\ell$ is the support of the tape inscription and q is the current state of T, scanning t_j. Let Σ_T be the input alphabet of T, let $\Gamma_T = \{s_1,\dots,s_m\}$ be the tape alphabet of T containing Σ_T, let $Q_T = \{q_1,\dots,q_n\}$ be the set of internal states of T, where q_1 is the initial state and q_n is the only final state. We encode the language VALC(T) over $\Sigma := \{q, s, 0, 1, c, \#\}$ using the morphism φ that is defined as follows, where $k := \max\{m, n\} + 1$:

$$q_i \mapsto q0^i1^{k-i}c^3 \ (1 \le i \le n),\ s_i \mapsto s0^i1^{k-i}c^3 \ (1 \le i \le m),\ \# \mapsto \#c^3.$$

By $\mathrm{VALC}_\varphi(T)$ we denote the image $\varphi(\mathrm{VALC}(T))$.

Lemma 3. *From a single-tape Turing machine T we can construct a deterministic* 2-sRL-*automaton M_T and a regular language E_T over Σ such that* $\mathrm{VALC}_\varphi(T) = L(M_T) \cap E_T$.

Proof. First we define a number of auxiliary regular languages:

$$\begin{array}{lll}
\mathrm{CONF}_i & := \varphi_i(\Gamma_T^*)\cdot\varphi_i(Q_T)\cdot\varphi_i(\Gamma_T^+)\cdot\#c^i & \text{for } i=1,2,3;\\
\mathrm{CONF}_{\mathrm{init},i} & := \varphi_i(q_1)\cdot\varphi_i(\Sigma_T^+)\cdot\#c^i & \text{for } i=1,2,3;\\
\mathrm{CONF}_{\mathrm{final},i} & := \varphi_i(\Gamma_T^*)\cdot\varphi_i(q_n)\cdot\varphi_i(\Gamma_T^+)\cdot\#c^i & \text{for } i=1,2,3;\\
\mathrm{CONF}_{i,i+1} & := \varphi_i(\Gamma_T^*)\cdot\varphi_{i+1}(\Gamma_T^*)\cdot\varphi_{i+1}(Q_T)\cdot\varphi_{i+1}(\Gamma_T^*)\cdot\#c^{i+1} & \\
 & \quad\cup\ \varphi_i(\Gamma_T^*)\cdot\varphi_i(Q_T)\cdot\varphi_i(\Gamma_T^*)\cdot\varphi_{i+1}(\Gamma_T^*)\cdot\#c^{i+1} & \text{for } i=1,2.
\end{array}$$

Here φ_i is obtained from φ ($=\varphi_3$) by replacing each factor c^3 by the factor c^i, $i = 1, 2$. Without a restart the deterministic 2-sRL-automaton M_T will accept the regular language $S := \mathrm{CONF}_{\mathrm{init},1}\cdot\mathrm{CONF}_1^*\cdot\mathrm{CONF}_{\mathrm{final},2}$. In each cycle it will check that the current tape contents belongs to the regular language

$$\begin{aligned}
E := {} & \mathrm{CONF}_{\mathrm{init},3}\cdot\mathrm{CONF}_3^*\cdot\mathrm{CONF}_{\mathrm{final},3}\\
& \cup\ \mathrm{CONF}_{\mathrm{init},1}\cdot\mathrm{CONF}_1^*\cdot\mathrm{CONF}_{1,2}\cdot\mathrm{CONF}_{2,3}\cdot\mathrm{CONF}_3^*\cdot\mathrm{CONF}_{\mathrm{final},3},
\end{aligned}$$

where $\mathrm{CONF}_{\mathrm{init},1}$ can coincide with $\mathrm{CONF}_{1,2}$, and $\mathrm{CONF}_{\mathrm{final},3}$ can coincide with $\mathrm{CONF}_{2,3}$. In each cycle M_T compares the first letter from $\varphi_2(Q_T\cup\Gamma_T)$ of the factor from $\mathrm{CONF}_{1,2}$ with the first letter from $\varphi_3(Q_T\cup\Gamma_T)$ of the factor from $\mathrm{CONF}_{2,3}$. If these two symbols correspond to each other with respect to the transition relation of the Turing machine T, then from the suffix of each of these encoded letters, a single occurrence of the symbol c is deleted. Thus, M_T is a deterministic 2-sRL-automaton. Taking E_T as the regular language $E_T := \mathrm{CONF}_{\mathrm{init},3}\cdot\mathrm{CONF}_3^*\cdot\mathrm{CONF}_{\mathrm{final},3}$, it is easily verified that $\mathrm{VALC}_\varphi(T) = L(M_T)\cap E_T$ holds. □

Based on this technical result we will now establish a non-recursive trade-off. For doing so we need the following general result, which is a generalization of Hartmanis' technique [2] for establishing non-recursive trade-offs. Here a descriptional system $\mathcal{D}$ is a recursive set of finite descriptions, where each descriptor $A \in \mathcal{D}$ describes a formal language $L(A)$, and there exists an effective procedure to convert A into a Turing machine that decides (or semi-decides) membership in $L(A)$, if $L(A)$ is recursive (recursively enumerable). For example, the class of finite-state acceptors and the class of sRL-automata constitute descriptional systems.

Theorem 5. [7] *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two descriptional systems. If for every Turing machine T, a language $L_T \in L(\mathcal{D}_1)$ and a descriptor $A_T \in \mathcal{D}_1$ for L_T can be effectively constructed such that $L_T \in L(\mathcal{D}_2)$ if and only if the language $L(T)$ accepted by T is finite, then the trade-off between $\mathcal{D}_1$ and $\mathcal{D}_2$ is non-recursive.*

Let $\mathcal{D} := (\mathsf{det\text{-}2\text{-}sRL}, \mathsf{REG})$ consist of pairs of the form (M, E), where M is a deterministic 2-sRL-automaton and E is a regular expression. A pair (M, E) describes the language $L(M, E) := L(M) \cap E$. Obviously, $\mathcal{D}$ is a descriptional system.

Given a (single-tape) Turing machine T, let L_T denote the language $\mathrm{VALC}_\varphi(T)$ of encodings of valid computations. From Lemma 3 we see that a descriptor $(M, E) \in \mathcal{D}$ can be constructed effectively for this language. Further, there exists a finite-state acceptor for the language $\mathrm{VALC}_\varphi(T)$ if and only if this language is regular, which in turn is the case if and only if the language $L(T)$ is finite (see, e.g, [4]). Thus, we obtain the following consequence.

Corollary 1. *There is a non-recursive trade-off between the descriptional system $\mathcal{D} = (\mathsf{det\text{-}2\text{-}sRL}, \mathsf{REG})$ and the finite-state acceptors.*

Thus, deterministic 2-sRL-automata (in combination with a regular expression) allow a very succinct representation of (certain) regular languages.

It currently remains open whether a corresponding non-recursive trade-off exists between the (deterministic) 2-sRL-automata and the finite-state acceptors.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. A general theory of translation. *Math. Systems Theory* 3 (1969) 193-221.
2. J. Hartmanis. On the succinctness of different representations of languages. *SIAM J. Comput.* 9 (1980) 114-120.
3. M. Holzer, M. Kutrib, and J. Reimann. Descriptional complexity of deterministic restarting automata. In: C. Mereghetti, B. Palano, G. Pighizzini, and D. Wotschke (eds.), *DCFS 2005, Proc.*, Università degli Studi di Milano, 2005, 158–169.
4. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.
5. M. Lopatková, M. Plátek, and V. Kuboň. Modeling syntax of free word-order languages: Dependency analysis by reduction. In: V. Matoušek, P. Mautner, and T. Pavelka (eds.), *TSD 2005, Proc., LNCS 3658*, Springer, Berlin, 2005, 140–147.
6. M. Lothaire. *Combinatorics on Words.* Encyclopedia of Mathematics, Vol. 17, Addison-Wesley, Reading, 1983.
7. A. Malcher. Descriptional complexity of cellular automata and its decidability questions. *J. Autom. Lang. Comb.* 7 (2002) 549–560.
8. F. Mráz, F. Otto, and M. Plátek. Degrees of free word-order and restarting automata. *Grammars*, to appear.

9. F. Mráz, F. Otto, and M. Plátek. On the gap-complexity of simple RL-automata. *DLT 2006*, to appear.
10. F. Otto. Restarting automata and their relations to the Chomsky hierarchy. In: Z. Ésik and Z. Fülöp (eds.), *DLT 2003, Proc.*, *LNCS 2710*, Springer, Berlin, 2003, 55–74.
11. G. Păun. *Marcus Contextual Grammars*, Studies in Linguistics and Philosophy, vol. 67. Kluwer, Dordrecht/Boston/London, 1997.