Shrinking Restarting Automata[*]

Tomasz Jurdziński[1] and **Friedrich Otto**[2]

[1] Institute of Computer Science, University of Wrocław
51-151 Wrocław, Poland
tju@ii.uni.wroc.pl

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

Abstract. Restarting automata are a restricted model of computation that was introduced by Jančar et. al. to model the so-called *analysis by reduction*. A computation of a restarting automaton consists of a sequence of cycles such that in each cycle the automaton performs exactly one rewrite step, which replaces a small part of the tape content by another, even *shorter* word. Thus, each language accepted by a restarting automaton belongs to the complexity class $\mathsf{CSL} \cap \mathsf{NP}$. Here we consider a natural generalization of this model, called *shrinking restarting automaton*, where we do no longer insist on the requirement that each rewrite step decreases the length of the tape content. Instead we require that there exists a *weight function* such that each rewrite step decreases the weight of the tape content with respect to that function. The language accepted by such an automaton still belongs to the complexity class $\mathsf{CSL} \cap \mathsf{NP}$. While it is still unknown whether the two most general types of one-way restarting automata, the RWW-automaton and the RRWW-automaton, differ in their expressive power, we will see that the classes of languages accepted by the shrinking RWW-automaton and the shrinking RRWW-automaton coincide. As a consequence of our proof, it turns out that there exists a reduction by morphisms from the language class $\mathcal{L}(\mathsf{RRWW})$ to the class $\mathcal{L}(\mathsf{RWW})$. Further, we will see that the shrinking restarting automaton is a rather *robust* model of computation. Finally, we will relate shrinking RRWW-automata to finite-change automata. This will lead to some new insights into the relationships between the classes of languages characterized by (shrinking) restarting automata and some well-known time and space complexity classes.

1 Introduction

The restarting automaton was introduced by Jančar et. al. as a formal tool to model the *analysis by reduction*, which is a technique used in linguistics to analyze sentences of natural languages [9]. This technique consists in a stepwise

[*] This work was supported by a grant from the Deutsche Forschungsgemeinschaft. It was performed while T. Jurdziński was visiting the University of Kassel.

simplification of a given sentence in such a way that the correctness or incorrectness of the sentence is not affected. It is applied primarily in languages that have a free word order. Already several programs used in Czech and German (corpus) linguistics are based on the idea of restarting automata [19, 21].

A (one-way) restarting automaton, RRWW-automaton for short, is a device M that consists of a finite-state control, a flexible tape containing a word delimited by sentinels, and a read/write window of a fixed size. This window moves from left to right along the tape until the control decides (nondeterministically) that the content of the window should be rewritten by some *shorter* string. In fact, the new string may contain auxiliary symbols that do not belong to the input alphabet. After a rewrite, M can continue to move its window to the right until it either halts and accepts, or halts and rejects, or restarts, that is, it places its window over the left end of the tape, and reenters the initial state. Thus, each computation of M can be described through a sequence of cycles.

Accordingly, an RRWW-automaton M can be defined by a finite set of meta-instructions of the form $(E_1, u \to v, E_2)$ and (E_1, Accept), where E_1 and E_2 are regular expressions, called the *constraints* of the meta-instruction, and u, v are strings over the tape alphabet of M satisfying the condition $|u| > |v|$. In each cycle M nondeterministically chooses a meta-instruction to be applied next. If the chosen meta-instruction is of the form $(E_1, u \to v, E_2)$, then M halts and rejects, if the current tape content does not admit a factorization of the form xuy with $x \in L(E_1)$ and $y \in L(E_2)$; if, however, the tape content does admit one or more factorizations of this form, then one such factorization is chosen nondeterministically, and the tape content is transformed into xvy. If the chosen meta-instruction is of the form (E_1, Accept), then M halts and accepts if the current tape content belongs to the language $L(E_1)$; otherwise it halts and rejects. Thus, it is easily seen that M can be simulated by a nondeterministic single-tape Turing machine that runs in quadratic time using only linear space, that is, the language $L(M)$ accepted by M belongs to the complexity class $\mathsf{CSL} \cap \mathsf{NP}$.

By requiring that a restarting automaton must always perform a restart step immediately after executing a rewrite operation, we obtain the so-called RWW-automaton. Within any cycle such an automaton cannot scan the suffix of the tape content that is to the right of the position where the rewrite operation is performed. This restriction can be expressed by meta-instructions of the form $(E_1, u \to v)$, which can be applied successfully if the tape content admits a factorization of the form xuy satisfying $x \in L(E_1)$, that is, no restriction is placed on the corresponding suffix y. Although the definition of the RWW-automaton is clearly much more restricted than that of the RRWW-automaton, it is a long-standing open problem whether the class of languages $\mathcal{L}(\mathsf{RWW})$ accepted by RWW-automata is a proper subclass of the class of languages $\mathcal{L}(\mathsf{RRWW})$ accepted by RRWW-automata.

Many well-known classes of formal languages admit characterizations in terms of restricted variants of the restarting automaton. For example, the class of Church-Rosser languages CRL of McNaughton et al. [15] coincides with the class of languages that are accepted by the deterministic variant of the RWW- and the RRWW-automaton [17, 18], the class of context-free languages CFL is characterized by the monotone variants of the RWW- and the RRWW-automaton [10], and the class of deterministic context-free languages DCFL is characterized by several different variants of monotone deterministic RWW- and RRWW-automata [10]. In addition, the class of growing context-sensitive languages GCSL considered by Dahlhaus and Warmuth [8] coincides with the class of languages that are accepted by the weakly monotone variant of the RWW- and the RRWW-automaton [11]. Observe that in all these particular cases the considered variant of the RWW-automaton is just as powerful as the corresponding variant of the RRWW-automaton. On the other hand, it is known that for some types of restarting automata without auxiliary symbols, the RWW-variant is strictly less powerful than the RRWW-variant [10]. For a recent survey on restarting automata see [20].

In the present paper we consider a generalization of the restarting automaton, called *shrinking restarting automaton*. A shrinking restarting automaton M is defined just as a restarting automaton with the one exception that it is no longer required that each rewrite step $u \to v$ of M must be length-reducing. Instead there must exist a weight function ω that assigns a positive integer $\omega(a)$ to each letter a of M's tape alpabet Γ such that, for each rewrite step $u \to v$ of M, $\omega(u) > \omega(v)$ holds. Here the function ω is extended to a morphism $\omega : \Gamma^* \to \mathbb{N}$ as usual. Obviously, a shrinking restarting automaton can still be simulated by a nondeterministic single-tape Turing machine in quadratic time and linear space. Observe that similar generalizations have been considered for other types of automata [7], for grammar systems [6], and for string-rewriting systems [3].

The shrinking restarting automaton was introduced in [12], where it was shown that monotone (as well as left-monotone) shrinking restarting automata still characterize the class of context-free languages, and that deterministic shrinking RRWW-automata that are left-monotone are not more expressive than left-monotone deterministic RWW-automata.

Here we study the expressive power of the (nondeterministic) shrinking restarting automaton in general, where we only consider those variants that admit auxiliary symbols in addition to the input alphabet. After restating the basic definitions in Section 2, we establish our first main result in Section 3, which states that the shrinking RWW-automaton is just as expressive as the shrinking RRWW-automaton. As a corollary of our proof, we obtain a reduction by injective morphisms from the language class $\mathcal{L}(\mathsf{RRWW})$ to the language class $\mathcal{L}(\mathsf{RWW})$. This in itself is a major improvement of the reduction from $\mathcal{L}(\mathsf{RRWW})$ to $\mathcal{L}(\mathsf{RWW})$ presented in [11]. It clearly indicates that these classes may be very difficult to separate if they are indeed different.

In Section 4 we investigate further generalizations of the shrinking restarting automaton. First we study the case that the restart operation does not necessarily reset the finite-state control to the initial state, but that this operation is combined with a change of state just like any other operation. Secondly we consider the case that the automaton may perform up to c rewrite operations in each cycle for some constant $c > 1$. We will see in both cases that the expressive power of the shrinking RWW-automaton is not enhanced by these additional capabilities. This confirms our impression that the shrinking restarting automaton is a rather robust model.

Finally, in Section 5 we establish our second main result by giving a characterization of the class of languages accepted by shrinking RWW-automata in terms of the class of finite-change automata as introduced by von Braunmühl and Verbeek [4]. This characterization implies that this class of languages is actually contained in the class of deterministic context-sensitive languages DCSL, thus improving on the best previously known upper bound for $\mathcal{L}(\mathsf{RRWW})$, and that it contains the class Q of quasi-realtime languages [2], which coincides with the complexity class NTIME(lin) (the class of languages that are accepted by nondeterministic multi-tape Turing machines in linear time). In the concluding section we summarize our results and state some open problems.

2 Definitions

Throughout the paper ε will denote the empty word, and $\mathbb{N}_+$ will denote the set of all positive integers.

A (one-way) *restarting automaton*, RRWW-automaton for short, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, $¢, \$ \notin \Gamma$ are symbols that serve as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and

$$\delta : Q \times \mathcal{PC}^{(k)} \to \mathfrak{P}((Q \times (\{\mathsf{MVR}\} \cup \mathcal{PC}^{\leq(k-1)})) \cup \{\mathsf{Restart}, \mathsf{Accept}\})$$

is the *transition relation*. Here $\mathfrak{P}(S)$ denotes the powerset of the set S, $\mathcal{PC}^{(k)}$ is the set of *possible contents* of the read/write window of M, where

$$\mathcal{PC}^{(i)} := (¢ \cdot \Gamma^{i-1}) \cup \Gamma^i \cup (\Gamma^{\leq i-1} \cdot \$) \cup (¢ \cdot \Gamma^{\leq i-2} \cdot \$) \quad (i \geq 0),$$

and

$$\Gamma^{\leq n} := \bigcup_{i=0}^{n} \Gamma^i \quad \text{and} \quad \mathcal{PC}^{\leq(k-1)} := \bigcup_{i=0}^{k-1} \mathcal{PC}^{(i)}.$$

The transition relation contains four different types of transition steps:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$, $u \neq \$$. If M is in state q and sees the string u in its read/write window, then this move-right step causes M to shift the read/write window one position to the right and to enter state q'. However, if the content u of the read/write window is only the symbol $\$$, then no shift to the right is possible.

2. A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u \in \mathcal{PC}^{(k)}$, $u \neq \$$, and $v \in \mathcal{PC}^{\leq(k-1)}$ such that $|v| < |u|$. It causes M to replace the content u of the read/write window by the string v, thereby shortening the tape, and to enter state q'. Further, the read/write window is placed immediately to the right of the string v. However, some additional restrictions apply in that the border markers ¢ and \$ must not disappear from the tape nor that new occurrences of these markers are created. Further, the read/write window must not move across the right border marker \$, that is, if the string u ends in \$, then so does the string v, and after performing the rewrite operation, the read/write window is placed on the \$-symbol.
3. A *restart step* is of the form $\mathsf{Restart} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to place the read/write window over the left end of the tape, so that the first symbol it sees is the left border marker ¢, and to reenter the initial state q_0.
4. An *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to halt and accept.

If $\delta(q, u) = \emptyset$ for some $q \in Q$ and $u \in \mathcal{PC}^{(k)}$, then M necessarily halts, and we say that M *rejects* in this situation.

There is one additional restriction that the transition relation must satisfy. This restriction says that within any computation of M, *rewrite steps and restart steps alternate*, with a rewrite step coming first.

A *configuration* of M can be described by a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \varepsilon$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the head scans the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0¢w\$$, where $w \in \Gamma^*$; if $w \in \Sigma^*$, then $q_0¢w\$$ is an *initial configuration.* Thus, initial configurations are a particular type of restarting configurations.

A phase of a computation of M, called a *cycle*, begins with a restarting configuration, the head moves along the tape performing MVR and Rewrite operations until a Restart operation is performed and thus a new restarting configuration is reached. If no further Restart operation is performed, then the computation necessarily halts after finitely many steps – such a phase is called a *tail.* The above restriction on the transition relation implies that M performs *exactly one* Rewrite operation during each cycle – thus each new phase starts on a shorter word than the previous one, and that it executes at most one Rewrite operation during a tail computation.

An input word $w \in \Sigma^*$ is *accepted* by M, if there is a computation which, starting with the initial configuration $q_0¢w\$$, finishes by executing an Accept instruction. By $L(M)$ we denote the language consisting of all words accepted by M; we say that M *accepts (recognizes) the language* $L(M)$.

In general, an RRWW-automaton is nondeterministic, that is, for some pairs (q, u), there may be more than one applicable transition step. If that is not the case, then the automaton is deterministic. We use the notation det- to denote classes of deterministic restarting automata.

Also some restricted classes of restarting automata have been studied. An RWW-*automaton* is an RRWW-automaton that is required to execute a restart step immediately after performing a rewrite step. An RRW-automaton is an RRWW-automaton which does not use any auxiliary symbols, that is, its tape alphabet coincides with its input alphabet. Finally, an RR-automaton is an RRW-automaton whose rewrite instructions can be viewed as deletions, that is, if $(q', v) \in \delta(q, u)$, then v is a scattered subword of u. Obviously, the restrictions on the rewrite operation can be combined with the restriction on the restart operation, which leads to the RW-automaton and the R-automaton.

Finally, we come to the main topic of this paper, the shrinking restarting automaton. A *shrinking restarting automaton* $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is defined in the same way as a 'standard' restarting automaton with one exception. Namely, it is not required that a rewrite operation reduces the length of the tape. Instead, there must exist a *weight function* $\omega : \Gamma \to \mathbb{N}_+$ such that, for each rewrite step $(q', v) \in \delta(q, u)$ of M, $\omega(u) > \omega(v)$ holds. Thus, it is not necessary that $|u| > |v|$ holds, but rather that the weight decreases from u to v. Here ω is extended to a morphism $\omega : \Gamma^* \to \mathbb{N}$ by taking $\omega(\varepsilon) := 0$ and $\omega(wa) := \omega(w) + \omega(a)$ for all $w \in \Gamma^*$ and $a \in \Gamma$. Obviously, the length function $w \mapsto |w|$ is a particular weight function.

To be precise the weight function should also be defined for the delimiters ¢ and \$, which are not elements of Γ. However, as the restarting automaton is not allowed to remove ¢ or \$ from the tape nor to create new occurrences of these symbols, their weight does not influence the difference $\omega(u) - \omega(v)$ for any rewrite step $(q', v) \in \delta(q, u)$. Therefore, in order to simplify the notation, we do not assign weights to ¢ and \$.

If an automaton M is shrinking with respect to a weight function ω, we say that ω is a weight function *compatible* with M. In order to distinguish the 'standard' variant of the restarting automaton from the shrinking restarting automaton, we sometimes denote the former as *length-reducing* restarting automaton.

Notation. For any class A of automata, $\mathcal{L}(\mathsf{A})$ will denote the class of languages that can be accepted by automata from A. The class of shrinking RRWW-automata is denoted by sRRWW, and similarly the class of shrinking RWW-automata is denoted by sRWW.

Let $\omega : \Gamma \to \mathbb{N}_+$ be a weight function, where Γ is a finite alphabet, and let $\diamondsuit$ and $\triangle$ be two new symbols not contained in Γ. Then $r_\omega : \Gamma^* \to (\Gamma \cup \{\diamondsuit\})^*$ denotes the morphism that is induced by defining $r_\omega(a) := a\diamondsuit^{\omega(a)-1}$ for each $a \in \Gamma$. Thus, for each word $u \in \Gamma^*$, $|r_\omega(u)| = \omega(u)$, that is, the length of the word $r_\omega(u)$ coincides with the weight of u. Further, for $i \in \mathbb{N}_+$, we take r_i to denote the morphism $r_i : (\Gamma \cup \{\diamondsuit\})^* \to (\Gamma \cup \{\diamondsuit, \triangle\})^*$ that is defined by $r_i(a) := a\triangle^{i-1}$ for all $a \in \Gamma \cup \{\diamondsuit\}$. Thus, for each word $u \in (\Gamma \cup \{\diamondsuit\})^*$, $|r_i(u)| = i \cdot |u|$. Observe that r_ω as well as r_i $(i \geq 1)$ are encodings, that is, injective morphisms.

3 sRWW-Automata versus sRRWW-Automata

In this section we will see that the families of languages accepted by sRWW- and sRRWW-automata coincide. Moreover, we present a reduction by morphisms from the language class $\mathcal{L}(\mathsf{RRWW})$ to the language class $\mathcal{L}(\mathsf{RWW})$.

We begin our investigation by establishing a reduction from shrinking to length-reducing restarting automata.

Lemma 1.

(a) *If M is an* sRRWW-*automaton, and if ω is a weight function that is compatible with M, then $r_\omega(L(M)) \in \mathcal{L}(\mathsf{RRWW})$.*
(b) *If M is an* sRWW-*automaton, and if ω is a weight function that is compatible with M, then $r_\omega(L(M)) \in \mathcal{L}(\mathsf{RWW})$.*

Proof. We concentrate on sRRWW-automata, as the proof for sRWW-automata is completely analogous.

Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an sRRWW-automaton accepting the language $L \subseteq \Sigma^*$, and let $\omega : \Gamma \to \mathbb{N}_+$ be a weight function that is compatible with M. We define an RRWW-automaton M' that recognizes the language $r_\omega(L)$ as follows. For each meta-instruction $(E_1, u \to v, E_2)$ of M, M' has a meta-instruction of the form $(r_\omega(E_1), r_\omega(u) \to r_\omega(v), r_\omega(E_2))$, where the regular expression $r_\omega(E_i)$ is obtained from the regular expression E_i $(i = 1, 2)$ by replacing each letter $a \in \Gamma$ by its image $r_\omega(a)$. Similarly, for each meta-instruction of the form (E, Accept) of M, there is a corresponding meta-instruction of the form $(r_\omega(E), \mathsf{Accept})$ for M'. It follows easily that M' accepts the language $r_\omega(L)$, and that it is length-reducing, proving that $r_\omega(L) \in \mathcal{L}(\mathsf{RRWW})$. □

Next we come to the technical main result of this section relating sRRWW-automata to sRWW-automata.

Lemma 2. *Let M be an* sRRWW-*automaton that accepts a language $L \subseteq \Sigma^*$, and let ω be a weight function compatible with M. Then there exists an* sRWW-*automaton M' such that $L(M') = L(M)$, and M' is compatible with a weight function ω' that satisfies the equality $\omega'(a) = 54 \cdot \omega(a)$ for each input letter $a \in \Sigma$.*

We postpone the proof of Lemma 2 to Section 3.1. The next theorem, which is our first main result, is an immediate consequence of that lemma.

Theorem 1. $\mathcal{L}(\mathsf{sRWW}) = \mathcal{L}(\mathsf{sRRWW})$.

In addition, we obtain the following reduction from the language class $\mathcal{L}(\mathsf{RRWW})$ to the class $\mathcal{L}(\mathsf{RWW})$.

Theorem 2. *For each language $L \in \mathcal{L}(\mathsf{RRWW})$, $r_{54}(L) \in \mathcal{L}(\mathsf{RWW})$.*

Proof. Let M be an RRWW-automaton that accepts the language $L \subseteq \Sigma^*$. Then M can be interpreted as a shrinking RRWW-automaton that is compatible with the weight function that associates the weight 1 to each symbol. Thus, by

Lemma 2, there exists an sRWW-automaton M' such that $L(M') = L$, and M' is compatible with a weight function ω' that assigns the weight 54 to each input symbol $a \in \Sigma$. Now Lemma 1 (b) implies that $r_{\omega'}(L) \in \mathcal{L}(\mathsf{RWW})$. As $r_{\omega'}$ maps each symbol $a \in \Sigma$ onto the word $a\Diamond^{53}$, while r_{54} maps a onto the word $a\triangle^{53}$, it is clear that with $r_{\omega'}(L)$ also $r_{54}(L)$ is accepted by some RWW-automaton. □

Thus, Theorem 2 shows that the language class $\mathcal{L}(\mathsf{RRWW})$ reduces to the language class $\mathcal{L}(\mathsf{RWW})$ via injective morphisms. This is a remarkable improvement over the reduction from $\mathcal{L}(\mathsf{RRWW})$ to $\mathcal{L}(\mathsf{RWW})$ presented in [11], which, although being computable in linear-time, is not via morphisms. In fact, the reduction of [11] (see Theorem 3 below) even maps regular languages to non-context-free languages, indicating that it is not well-behaved from a language theoretical point of view.

3.1 Proof of Lemma 2

First, we recall the reduction from $\mathcal{L}(\mathsf{RRWW})$ to $\mathcal{L}(\mathsf{RWW})$ from [11] mentioned above.

Theorem 3. *Let L be a language over Σ, let $a, \triangle, c \notin \Sigma$ be three additional symbols, and let the mapping $\varphi : \Sigma^* \to (\Sigma \cup \{a, \triangle, c\})^*$ be defined by*

$$\varphi(x) := a^{3\cdot|x|} \cdot r_3(x) \cdot c^{3\cdot|x|}$$

for all $x \in \Sigma^*$. If $L \in \mathcal{L}(\mathsf{RRWW})$, then the language $\varphi(L) := \{\, \varphi(x) \mid x \in L \,\}$ belongs to $\mathcal{L}(\mathsf{RWW})$.

Let M be an sRRWW-automaton with input alphabet Σ that accepts the language $L \subseteq \Sigma^*$, and let ω be a weight function that is compatible with M. From Lemma 1 (a) we see that $r_\omega(L) \in \mathcal{L}(\mathsf{RRWW})$, and Theorem 3 implies that $\varphi(r_\omega(L))$ is accepted by some RWW-automaton M_φ. Further, as φ and r_ω are both injective mappings, it follows that $\varphi(r_\omega(x)) \neq \varphi(r_\omega(y))$ for all $x, y \in \Sigma^*$, $x \neq y$. Thus, we have the following equivalence for all $x \in \Sigma^*$:

$$x \in L \text{ if and only if } \varphi(r_\omega(x)) \in \varphi(r_\omega(L))\,. \qquad (1)$$

In order to prove Lemma 2 we will now construct an sRWW-automaton that, given a word $x \in \Sigma^*$ as input, first transforms x into the word $z := \varphi(r_\omega(x))$, and then simulates the computation of the (length-reducing) RWW-automaton M_φ on the input z. Thus, this sRWW-automaton will accept on input x if and only if z belongs to the language $\varphi(r_\omega(L))$, that is by (1), if and only if x belongs to the language L.

Our method of transforming x into $\varphi(r_\omega(x))$ is nondeterministic. Thus, for a given x, there are several different words z that can be produced, only one of them the intended result $\varphi(r_\omega(x))$. Unfortunately we cannot possibly verify the correctness of the word z produced without destroying it, but we can at least guarantee that $z = \varphi(r_\omega(x))$ if z belongs at all to the set $\varphi(r_\omega(L))$. As the RWW-automaton M_φ accepts only inputs from this set, it follows that this property is sufficient for our purposes.

We first give a high level description of the algorithm that is realized by the intended sRWW-automaton M'. Afterwards we will discuss the details of the implementation.

Let $x \in \Sigma^*$ be the given input. Our algorithm proceeds in three stages, which will be illustrated by an example below:

1. The word x is rewritten deterministically from left to right into $y := r_9(r_\omega(x))$, using a new alphabet A_0 of auxiliary letters. These letters are interpreted as describing two 'tracks,' the first of which now contains the word $y = r_9(r_\omega(x))$, while the second track is empty (we use the symbol $\perp$ to denote 'empty' content).
2. Now the content of the second track is rewritten into $a^p \cdot r_3(r_\omega(x)) \cdot c^p$, where $p := |r_3(r_\omega(x))|$. This is achieved by performing the following steps that all preserve the length of the tape:
   (a) $y' := r_3(x')$ is written as a prefix of the content of the second track for some word x' that is a supersequence of $r_\omega(x)$ (that is, $r_\omega(x)$ is a scattered subsequence of x').
   (b) The second track, containing a word of the form $y' \perp^m$ for some integer m, is rewritten into a word of the form $y'y'' \perp^p$, where $y'' := r_3(x'')$ for a scattered subsequence x'' of x', and $p := m - |y''| > 0$. In order to mark the border between y' and y'', a new subalphabet is used for y''.
   (c) The current content of the second track, $y'y'' \perp^p$, is rewritten deterministically from left to right into $z := a^{|y'|}y''c^p$.
3. The computation of the RWW-automaton M_φ on input z is simulated on the second track of the tape. The automaton M' accepts if and only if this computation of M_φ is accepting.

Example
Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an sRRWW-automaton with $\Sigma = \{b, d\}$, and let ω be a weight function compatible with M such that $\omega(b) = 1$ and $\omega(d) = 2$. Given the input word bd, in Stages 1–2(c) M' can execute the transformations displayed in Figure 1.

Here we have omitted some extra information that is stored in symbols, and that is needed to 'coordinate' the computation (see the implementation details given below). □

Next we describe the realization of the above algorithm by the sRWW-automaton M' in some detail.

Stage 1 can be realized by a sequence of cycles such that, in each of them, M' rewrites the leftmost symbol $a \in \Sigma$ that is still on the tape into the word $r_9(r_\omega(a))$. In order to make this rewrite step weight-reducing (and to distuinguish the symbols from Σ from those of A_0), we encode the word $r_9(r_\omega(a))$ as $r_9(r_\omega(\bar{a}))$,

Input	$b\ d$
Stage 1	$b \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle d \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle \diamondsuit \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle$
	$\perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp$
Stage 2(a)	$b \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle d \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle \diamondsuit \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle$
	$b \triangle \triangle d \triangle \triangle \diamondsuit \triangle \triangle \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp$
Stage 2(b)	$b \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle d \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle \diamondsuit \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle$
	$b \triangle \triangle d \triangle \triangle \diamondsuit \triangle \triangle b \triangle \triangle d \triangle \triangle \diamondsuit \triangle \triangle \perp \perp \perp \perp \perp \perp \perp \perp \perp$
Stage 2(c)	$b \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle d \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle \diamondsuit \triangle \triangle \triangle \triangle \triangle \triangle \triangle \triangle$
	$a\ a\ a\ a\ a\ a\ a\ a\ a\ b \triangle \triangle d \triangle \triangle \diamondsuit \triangle \triangle c\ c\ c\ c\ c\ c\ c\ c\ c$

Fig. 1. Stages 1 to 2(c) of the computation of M' on input bd

where $\bar{a}$ is a marked copy of a from the set $\overline{\Sigma} := \{\bar{a} \mid a \in \Sigma\}$. However, to simplify the notation, we will assume below that the tape content at the end of Stage 1 is simply the word $r_9(r_\omega(x))$.

For Stage 2(a), let $\Lambda := \Sigma \cup \{\diamondsuit, \triangle\}$, where $\diamondsuit, \triangle \notin \Gamma$ are the new symbols used in the definition of the morphisms r_ω and r_i ($i \in \mathbb{N}_+$), respectively. We will rewrite the actual tape content from left to right into appropriate symbols from $\Lambda \times \{\perp\}$. Intuitively this corresponds to the process of adding the second track. Simultaneously, we rewrite the prefix of the content of the second track from left to right into the word $r_3(x')$, where $x' \in (\Sigma \cup \{\diamondsuit\})^*$ is a supersequence of $r_\omega(x)$ chosen nondeterministically. Each rewrite step on the first track transforms 9 symbols, while each rewrite step on the second track replaces 3 symbols. In order to guarantee that x' is a supersequence of $r_\omega(x)$, we use constraints for the meta-instructions of M' which will ensure that the following conditions are met:

- at least one rewrite step is performed on the second track between any two consecutive rewrite steps on the first track,
- a rewrite step $a\triangle^8 \to (a, \perp)(\triangle, \perp)^8$ ($a \in \Sigma \cup \{\diamondsuit\}$) on the first track is enabled only if the symbol a is equal to the rightmost symbol from $\Sigma \cup \{\diamondsuit\}$ on the second track at that moment.

This 'coordination' of the rewrite steps on the first track with the rewrite steps on the second track will be achieved through some additional information that is stored in symbols as a third coordinate. So let $\hat{\Upsilon} := \Lambda \times \{\perp\} \times \{0, 1\}$ and $\bar{\Upsilon} := \Lambda \times \Lambda \times \{0, 1\}$. Further, for $i \in \{0, 1\}$, let $\hat{\Upsilon}_i := \Lambda \times \{\perp\} \times \{i\}$. The following meta-instructions describe Stage 2(a) in full detail, where $i, j \in \{0, 1\}$:

- the meta-instructions starting Stage 2(a), performing the first rewrite step on both tracks simultaneously:

$(\text{¢}, a\triangle^8 \to (a,a,0)(\triangle,\triangle,0)^2(\triangle,\perp,0)^6)$ for all $a \in \Sigma$,

- the meta-instructions for performing rewrite steps on the first track:
  $(\text{¢}\bar{\Upsilon}^* \cdot (a,b,i)(\triangle,\triangle,i)(\triangle,\triangle,i) \cdot \hat{\Upsilon}^* \cdot \hat{\Upsilon}_{1-i}, b\triangle^8 \to$
  $(b,\perp,i)(\triangle,\perp,i)^8)$ for all $a \in \Lambda$ and $b \in \Sigma \cup \{\Diamond\}$,
- the meta-instructions for performing rewrite steps on the second track:
  $(\text{¢}\bar{\Upsilon}^* \cdot (\triangle,\triangle,i), (a,\perp,j)(\triangle,\perp,j)(\triangle,\perp,j) \to$
  $(a,b,1-i)(\triangle,\triangle,1-i)(\triangle,\triangle,1-i))$ for all $a \in \Lambda$ and $b \in \Sigma \cup \{\Diamond\}$.

Observe that the value of the third coordinate alternates in consecutive rewrite steps on each track. Further, a rewrite step on the first track is only possible in case that the rightmost rewrite step on the second track was done for the same symbol from $\Sigma \cup \{\Diamond\}$. Note that rewrite steps on the second track are only possible to the left of the most recent rewrite step on the first track. Moreover, the third coordinate of the rightmost symbol rewritten on the second track must agree with the third coordinate of the current rewrite step on the first track. These constraints guarantee that the aforementioned conditions are satisfied. In addition, they ensure that the word x', the encoding $r_3(x')$ of which is written on the second track, is a supersequence of $r_\omega(x)$.

In Stage 2(b) we apply the techniques developed in [13] for realizing the intended copying. As the ideas behind them are similar to those used in Stage 2(a), we skip the details.

Stage 2(c) is a straightforward deterministic computation. The rewriting is simply done from left to right, replacing 3 symbols per cycle. Finally, Stage 3 is just the simulation of the automaton M_φ on the content of the second track.

As each of these stages starts by rewriting the leftmost symbols on the tape into symbols that do not occur in previous stages, it is ensured that the stages are applied in the appropriate order. Finally, the fact that $L(M') = L$ follows from the following proposition.

Proposition 1. *The automaton M' accepts $x \in \Sigma^*$ if and only if $x \in L$.*

Proof. From the above description of M' we see immediately that M' has an accepting computation for each word $x \in L$. On the other hand, if an input $x \in \Sigma^*$ is accepted by M', then all of the following conditions are met:

- $|y| = 9 \cdot |r_\omega(x)|$;
- $|y'| \geq 3 \cdot |r_\omega(x)|$, as $y' = r_3(x')$ for a supersequence of x' of $r_\omega(x)$;
- $|y''| \leq |y'|$, as $y'' = r_3(x'')$ for a scattered subsequence x'' of x';
- $|y'| + |y''| + p = |y| = 9 \cdot |r_\omega(x)|$, as none of the steps in Stage 2 changes the length of the tape content.

Further, M' accepts in Stage 3 if and only if M_φ accepts on input $a^{|y'|} \cdot y'' \cdot c^p$, which in turn happens if and only if $|y'| = |y''| = p$ and $y'' = r_3(r_\omega(\hat{x}))$ for some

$\hat{x} \in L$. As r_3 is injective, it follows that $x'' = r_\omega(\hat{x})$. Now x'' is a subsequence of x', and x' is a supersequence of $r_\omega(x)$ such that

$$3 \cdot |x'| = |y'| = |y''| = 3 \cdot |x''| = p = 3 \cdot |r_\omega(x)|,$$

which implies that $x'' = x' = r_\omega(x)$ holds. As r_ω is injective, this yields $\hat{x} = x$, which in turn shows that $x \in L$. Thus, $L(M') = L$. □

It remains to show that M' is weight-reducing with respect to a weight function ω' satisfying $\omega'(a) = 54 \cdot \omega(a)$ for each letter $a \in \Sigma$. For defining ω' we consider the various stages of M' in turn:

– Stage 1: A symbol $a \in \Sigma$ of weight $54 \cdot \omega(a)$ is replaced by $9 \cdot \omega(a)$ symbols. We choose weight 5 for each of these symbols. In this way the overall weight is reduced from $54 \cdot \omega(a)$ to $45 \cdot \omega(a)$ by each of these rewrite steps.
– Stage 2(a): During this phase each symbol on the tape is rewritten at most twice, while no symbols are inserted or removed from the tape. Each time we rewrite a symbol, we replace it by a symbol from a new subalphabet. Thus, if we assign weight 4 and 3 to these new symbols, respectively, then each of these rewrite steps is weight-reducing.
– Stage 2(b): During this phase each symbol on the tape is rewritten at most once. Again we use a new subalphabet for the rewrite steps, the letters of which are assigned the weight 2. Again each rewrite step is then weight-reducing.
– Stage 2(c): During this phase each symbol on the tape is rewritten exactly once, again using a new subalphabet. By assigning weight 1 to the elements of this subalphabet, we ensure that these rewrite steps are weight-reducing, too.
– Stage 3: Here the RWW-automaton M_φ is being simulated, which only has length-reducing rewrite steps. As it only works with letters of weight 1, its computation is also weight-reducing.

This completes the proof of Lemma 2.

4 Further Generalizations of Restarting Automata

Here we consider two further generalizations of the restarting automaton. One of the essential restrictions of the standard model (and also of the shrinking model) is the fact that whenever a restarting automaton executes a restart operation, then its finite-state control is reset to the initial state. Hence, in its finite-state control the automaton cannot store any information from one cycle to the next, that is, all such information must be stored on the tape. But here the second essential restriction comes it, which states that a restarting automaton must perform exactly one rewrite step in each cycle. In particular, this implies that, in each cycle, only one short factor of the tape content can be changed, and therewith used for storing information for future cycles.

Here we generalize from both these restrictions. First we consider restarting automata for which the restart operation is combined with a state transition just as all the other operations. This yields the so-called *non-forgetting* restarting automaton. Then we turn to the restarting automaton with *multiple rewrites* which performs c rewrite transitions in each cycle for some constant $c \in \mathbb{N}_+$.

4.1 Non-Forgetting sRRWW-Automata

We say that a (length-reducing or shrinking) restarting automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is *non-forgetting* if all its restart transitions are of the form $(q', \mathsf{Restart}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$. This restart transition causes M to place its read/write window over the left end of the tape and to enter state q'. In particular, this allows M to carry some information from one cycle to the next.

Non-forgetting restarting automata have been introduced by Messerschmidt and Stamer in [16], where they show that non-forgetting deterministic R-automata already accept languages that are not even growing context-sensitive. For example, there exists an automaton of this form for the copy language $L_{\text{copy}} := \{ w\#w \mid w \in \{a, b\}^* \}$, which is not growing context-sensitive [5, 14]. As $\mathcal{L}(\mathsf{det\text{-}R})$ is a proper subclass of the class CRL of Church-Rosser languages, which in turn is a proper subclass of GCSL, this shows that at least for restarting automata without auxiliary symbols the non-forgetting variant is much more expressive than the standard variant.

While it is still open whether the same is true for (length-reducing) restarting automata with auxiliary symbols, we will show below that for shrinking RRWW-automata this generalization does not increase the expressive power.

Theorem 4. *For each non-forgetting* sRRWW-*automaton, there exists an* sRWW-*automaton that accepts the same language.*

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a non-forgetting sRRWW-automaton accepting a language $L \subseteq \Sigma^*$. We construct an sRRWW-automaton M' that simulates the computations of M in a cycle-by-cycle fashion. As $\mathcal{L}(\mathsf{sRRWW}) = \mathcal{L}(\mathsf{sRWW})$ by Theorem 1, this proves the intended result.

To simplify the discussion we assume without loss of generality that each rewrite transition of M has the form $u \to y$ for some word $y \notin \{\varepsilon, ¢, \$\}$. If a cycle of a computation of M ends by executing the restart instruction $(q', \mathsf{Restart}) \in \delta(q, u)$, then the state q' is called the *end-state* of that cycle.

The simulation of each cycle of M will require up to 3 cycles of M'. The idea of the simulation consists in encoding information about the end-state of a cycle of a computation of M in which the rewrite transition $u \to v$ is executed in an encoded version of the word v. To this end the auxiliary alphabet must be extended appropriately. However, there is a technical problem, as after simulating several cycles of M, the tape content of M' may contain encodings of several end-states of M, among which it must distinguish the one that corresponds to the cycle of M just prior to the one that is currently being simulated. To overcome this problem we will remove the extra information that is redundant, just keeping the encodings of the end-states of the two most recent cycles of the computation of M that is being simulated.

To this end, let $\Gamma' := \Gamma \cup (\Gamma \times Q \times \{0,1\})$ be the tape alphabet of M'. A symbol from $\Gamma_0 := \Gamma \times Q \times \{0\}$ will be interpreted as the encoding of the end-state of the current cycle of M, while a symbol from $\Gamma_1 := \Gamma \times Q \times \{1\}$ will be interpreted as the encoding of the end-state of the previous cycle of M.

Depending on the number of symbols from $\Gamma_0 \cup \Gamma_1$ on its tape, M' will execute the following types of cycles:

1. If the tape content w belongs to Σ^*, then M' is just in the process of simulating the first cycle of a computation of M on input w. Assume that during this cycle, M performs the rewrite transition $u \to vay$, where $a \in \Gamma \setminus \{¢, \$\}$, and that q is the end-state of this cycle. Then M' simulates this rewrite step by executing the rewrite transition $u \to v(a,q,1)y$.
2. If the tape content is of the form $wa'z$ for some words $w, z \in \Gamma^*$ and $a' = (a,q,1) \in \Gamma_1$, then M' is allowed to simulate the next cycle of the computation of M, assuming that the previous cycle of M ended with the tape content waz and end-state q.

   Assume that in this cycle, M performs the rewrite transition $x \to vby$, where $b \in \Gamma \setminus \{¢, \$\}$, and that $\bar{q}$ is the end-state of this cycle. If, in the corresponding rewrite configuration (simulating the rewrite transition $x \to vby$ of M), the window of M' does not contain the symbol a', then M' simulates the current rewrite step by $x \to v(b,\bar{q},0)y$, and if the window of M' contains the symbol a', then M' executes the rewrite step $x_0(a,q,1)x_1 \to v(b,\bar{q},1)y$ for the appropriate factorization $x = x_0ax_1$ of x.
3. If the tape content is of the form $wa'y$ for a symbol $a' = (a,q,1) \in \Gamma_1$ and some words w, y satisfying $wy \in \Gamma^* \cdot \Gamma_0 \cdot \Gamma^*$, then M' simply performs a rewrite step that replaces the symbol a' by the symbol a. In this way the information about the end-state of the last but one cycle of M is deleted, before the simulation of the next cycle of M starts.
4. If the tape content is of the form $wa'y$ for a symbol $a' = (a,q,0) \in \Gamma_0$ and some words $w, y \in \Gamma^*$, then M' replaces the symbol a' by the symbol $(a,q,1)$. In this way the information about the last cycle of M stored on the tape of M' is modified to be interpreted as information on the last but one cycle of M. Thus, the simulation of the next cycle of M can be started.

Finally, assume that M accepts the tape content $u \in \Gamma^*$ in a tail computation following a cycle with end-state q. Then M' will accept all words of the form $u_0(a,q,1)u_1$ in tail computations, where $u = u_0au_1$.

As M' is nondeterministic, it can guess the end-state of the previous cycle of M and which of the above cases occurs in each cycle. If at some point it realizes that its guess is incorrect, it halts and rejects. Otherwise, it correctly simulates the computation of M until it reaches a halting configuration. Thus, M' can simulate each accepting computation of M, and it is easily seen that each accepting computation of M' corresponds to an accepting computation of M. In particular, this shows that $L(M') = L$.

It remains to show that M' is shrinking. Let ω be a weight function that is compatible with M, and let $\omega' : \Gamma \to \mathbb{N}_+$ be defined by taking $\omega'(a) := 3 \cdot \omega(a)$ for each $a \in \Gamma$. Then with respect to the weight function ω', each rewrite step of M reduces the weight of the tape content by at least 3. Now we extend the function ω' to elements of $\Gamma_0 \cup \Gamma_1$ by taking $\omega'((a, q, 0)) := \omega'(a) + 2$ and $\omega'((a, q, 1)) := \omega'(a) + 1$ for each $a \in \Gamma$ and $q \in Q$. Then the sRRWW-automaton M' is compatible with this weight function ω'. Indeed, if M' applies the simulation of a rewrite transition of the automaton M (see cases 1 and 2 above), then the overall weight of the tape content is reduced with respect to ω', as each rewrite step of M decreases the overall weight of the tape content by at least 3, while the symbol from $\Gamma_0 \cup \Gamma_1$ that is introduced by the corresponding step of M' increases the overall weight of the resulting tape content by at most 2. In the other cases, either a symbol $(a, q, 0) \in \Gamma_0$ is replaced by $(a, q, 1) \in \Gamma_1$ or a symbol $(a, q, 1) \in \Gamma_1$ is replaced by $a \in \Gamma$. Each of these rewrite steps reduces the overall weight by 1. This completes the proof of Theorem 4. □

4.2 sRRWW-Automata with Multiple Rewrites

A (shrinking) restarting automaton M with *multiple rewrites* is a (shrinking) restarting automaton that performs up to c rewrite operations in each cycle for some constant $c \in \mathbb{N}_+$. Observe that by definition each rewrite operation of a (length-reducing) restarting automaton reduces the length of the tape content, and that each rewrite operation of a shrinking restarting automaton reduces the overall weight of the tape content with respect to a compatible weight function.

It is easily seen that there exists a deterministic R-automaton with multiple rewrites (actually two rewrites per cycle suffice) that accepts the language L_{copy}. Hence, for this type of restarting automaton the generalization to multiple rewrites per cycle increases the expressive power considerably.

It is not known whether this is also true for (length-reducing) restarting automata with auxiliary symbols, but we can at least show that for shrinking restarting automata with auxiliary symbols the generalization to multiple rewrites does not increase the expressive power.

Theorem 5. *For each* sRRWW-*automaton with multiple rewrites, there exists an* sRWW-*automaton that accepts the same language.*

Proof. From Theorem 1 we know that the class of languages $\mathcal{L}(\mathsf{sRWW})$ coincides with the class of languages that are recognized by sRRWW-automata. Thus, it suffices to give a simulation of an sRRWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ with $c > 1$ rewrites per cycle by an sRRWW-automaton M'. Without loss of generality we can assume that each rewrite step of M is of the form $u \to y$ for some word $y \notin \{\varepsilon, ¢, \$\}$.

First we assume that M performs exactly c rewrite steps in each cycle. The case that some cycles contain less than c rewrite steps will be discussed later. The automaton M' simulates each cycle of M by $2c$ cycles. At the beginning and at the end of such a simulation the tape content of M' will be identical to the corresponding tape content of M, while M' will use some new auxiliary symbols during the simulation process.

During the first c cycles of such a simulation M' simulates the c rewrite steps of M from left to right, one at a time. However, each symbol $a \in \Gamma$ that is written by M during the i-th rewrite step is encoded by M' by the symbol (a, q, i), where q is the state of M after performing this rewrite step. In the next c cycles M' replaces the factors from $(\Gamma \times Q \times \{1, \ldots, c\})^+$ by the appropriate factors from Γ^+, again from left to right, one at a time.

Let $\Gamma_i := \Gamma \times Q \times \{i\}$ $(1 \le i \le c)$. Thus, each time the tape content contains symbols from $\Gamma_1, \ldots, \Gamma_i$ for some $i < c$, the automaton M' should simulate the $(i + 1)$-st rewrite step of the current cycle of M. On the other hand, each time the tape content contains symbols from $\Gamma_i, \ldots, \Gamma_c$ for some $i > 0$, M' has to replace a factor from Γ_i^+ by the appropriate factor from Γ^+.

Finally assume that during the current cycle, M only performs c' rewrite steps for some $c' < c$. Then M' uses essentially the technique described above for the case c'. However, during each of the first c' rewrite steps, M' encodes the additional information in the symbols written that it is currently simulating a cycle of M that contains only c' rewrite steps. Of course, M' does not know this fact at the beginning of the simulation, so that it has to guess the value of c', and in case the guess is not correct, it will realize this and reject.

One can easily verify that M' accepts the same language as M. Further, using a technique similar to the one used in the proof of Theorem 4, one can prove that M' is shrinking. □

Combining Theorems 4 and 5 we obtain the following result.

Corollary 1. *The class* $\mathcal{L}$(sRWW) *of languages recognized by* sRWW-*automata coincides with the class of languages that are recognized by non-forgetting* sRRWW-*automata with multiple rewrites.*

5 Restarting Automata versus Finite-Change Automata

In this section we derive a characterization of the language class $\mathcal{L}$(sRRWW) in terms of a restricted type of Turing machines considered by von Braunmühl and Verbeek [4]. Based on this characterization we will obtain some new relationships between the language classes accepted by (shrinking) RRWW-automata and some classical time and space complexity classes.

In [4] von Braunmühl and Verbeek introduced a model of the Turing machine that they called *finite-change automaton.* A finite-change automaton is a non-deterministic single-tape Turing machine A that is parameterized by a function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(n) \ge n$ for all $n \in \mathbb{N}$ and a constant $k \in \mathbb{N}_+$. Given an input of length n (as the initial inscription of its tape), A must not visit more than $f(n)$ cells, and it must not change the content of any cell more than k times during any accepting computation on the given input. By $k\mathsf{C}(f)$ we denote the class of finite-change automata meeting these restrictions, and by $\mathsf{FC}(f)$ we denote the union

$$\mathsf{FC}(f) := \bigcup_{k>0} k\mathsf{C}(f).$$

For the special case of the identity function $f(n) = n$, we denote the corresponding classes of finite-change automata by $k\mathsf{C}$ and FC, respectively.

In order to enable the finite-change automaton to recognize the left end and the right end of the given input, we assume here that the initial tape content for a given input $x \in \Sigma^*$ is of the form ¢x\$, where ¢ and \$ are special markers not contained in Σ. Hence, the length of the initial tape inscription is $n := |x| + 2$. Another option would be to use particularly marked symbols for the first and the last letter of the input word. That these approaches are equivalent follows from the following technical result of [4].

Lemma 3. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function satisfying $f(n) \geq n$ for each $n \in \mathbb{N}$. Then $\mathcal{L}(\mathsf{FC}(f(n))) = \mathcal{L}(\mathsf{FC}(c \cdot f(n)))$ for each constant $c \geq 1$.*

The following proposition expresses an easy observation.

Proposition 2. $\mathcal{L}(\mathsf{RR}) \subseteq \mathcal{L}(1\mathsf{C})$.

Proof. Let M be an RR-automaton, that is, in each rewrite step M only deletes some symbols from the current content of its read/write window. A one-change automaton A can simulate the computations of M cycle by cycle as follows. If there are only input symbols on the tape between the end markers ¢ and \$, then A simulates the first cycle of a computation of M, starting at the ¢-symbol. After finding those positions on the tape that contain the symbols that M deletes in the first cycle of the current computation, A simply replaces each of them by the special 'erase symbol' E. Thereafter A moves its head back to the ¢-symbol and starts the simulation of the next cycle of M. If there are occurrences of the symbol E on its tape, A just ignores all occurrences of this special symbol, determining those positions on the tape that contain the input symbols that M deletes in the current cycle. Again A replaces each of these symbols by the symbol E. It follows that $\mathcal{L}(\mathsf{RR}) \subseteq \mathcal{L}(1\mathsf{C})$. □

Using a more sophisticated simulation we obtain the following result.

Lemma 4. $\mathcal{L}(\mathsf{sRRWW}) \subseteq \mathcal{L}(\mathsf{FC})$.

Proof. Let M be an sRRWW-automaton accepting a language $L \subseteq \Sigma^*$. By Lemma 1 (a), there exist an injective morphism r_ω and a (length-reducing) RRWW-automaton M' with input alphabet $\Sigma \cup \{\diamondsuit\}$ such that, for each word $x \in \Sigma^*$, $x \in L$ if and only if $r_\omega(x) \in L(M')$. According to [13], Theorems 1 and 2, there exist an injective morphism φ and an RR-automaton M'' such that, for all words $x' \in (\Sigma \cup \{\diamondsuit\})^*$, $x' \in L(M')$ if and only if $\varphi(x') \in L(M'')$. Thus, we see that, for all words $x \in \Sigma^*$, $x \in L$ if and only if $\varphi'(x) \in L(M'')$, where φ' is the morphism obtained by the composition of r_ω and φ.

Based of the observations above, we construct a finite-change automaton A that proceeds as follows:

1. First the image $\varphi'(x)$ of the input word x is written onto the part of the tape immediately to the right of the given input.
2. Then the computation of the RR-automaton M'' on $\varphi'(x)$ is simulated.

Obviously, A accepts the language L. Further, this simulation requires at most space $(c+1) \cdot n$, where $c := \max\{\, |\varphi'(a)| \mid a \in \Sigma \,\}$. During the first part of the computation the content of each cell is changed at most once. During the second part of the computation the content of each cell is again changed at most once (see Proposition 2).

Hence, A is a two-change automaton for the language L that uses only linear space. From Lemma 3 it follows that $L = L(A) \in \mathcal{L}(\mathsf{FC})$. □

Thus, each shrinking RRWW-automaton can be simulated by a finite-change automaton. Interestingly also the converse of the inclusion above holds.

Lemma 5. $\mathcal{L}(\mathsf{FC}) \subseteq \mathcal{L}(\mathsf{sRRWW})$.

Proof. Let $A \in \mathsf{FC}$ be a finite-change automaton which changes each tape cell at most j times in the course of an accepting computation, where $j \in \mathbb{N}_+$ is a constant. Notice that in an accepting computation A can only visit the cells containing the original input word and the cells located directly to the left and to the right of the input word. Recall that we assume that the input is given in the form ¢x\$, where ¢ and \$ are special markers not contained in the input alphabet.

By a simple modification we can convert A into a finite-change automaton A' such that $L(A) = L(A')$, and A' meets the following restrictions during accepting computations:

– A' only visits the cells containing the input ¢x\$;
– A' changes each cell at most j times;
– each step of A' which changes the content of a cell leaves the head of A' stationary, that is, rewrite operations and head movements are performed by different steps;
– the working alphabet Γ of A' consists of $j+1$ disjoint subalphabets $\Gamma_0, \ldots, \Gamma_j$, where Γ_0 is the input alphabet, and the i-th change of the content of a cell produces an element of Γ_i for all $i = 1, \ldots, j$.

Now we present a simulation of the finite-change automaton A' by an sRRWW-automaton M.

If Q is the set of states of A', then we take $\Lambda := \Gamma \cup (\Gamma \times Q \times \{0,1\})$ to be the tape alphabet of M. Recall that $\Gamma_0 \subset \Gamma$ is the input alphabet of A', and therewith of M. The automaton M works according to the following description, employing ideas similar to those used in the simulation presented in the proof of Theorem 4:

1. If there are no auxiliary symbols on the tape, M simulates that part of a computation of A' that starts with an initial configuration and that ends when A' changes the content of a tape cell for the first time, or that ends with A' accepting or rejecting without changing the content of any tape cell. In the latter case M also accepts or rejects, respectively, in a tail computation. In the former case assume that A' changes the content of a tape cell by executing the instruction $\delta(q, a) = (q', b)$, that is, A' replaces an occurrence of the symbol $a \in \Gamma_0$ by an occurrence of the symbol $b \in \Gamma_1$ and changes its state from q to q'. M determines this tape cell, and then M rewrites the symbol a into the symbol $(b, q', 0)$. There is a slight problem due to the fact that M is a one-way device, while A' can move its head both to the left and to the right. However, in each step M can simply guess the corresponding crossing sequence of A' and verify the correctness of its guess, using the standard method of simulating a two-way finite-state acceptor by a one-way finite-state acceptor.
2. If the tape content is of the form $wa'z$ for some words $w, z \in \Gamma^*$ and a symbol $a' = (a, q, 0) \in \Gamma \times Q \times \{0\}$, then M simulates that part of a computation of A' on waz that starts in state q at the position of the symbol a' and that ends when A' changes the content of the next tape cell, or that ends with A' accepting or rejecting, if no more tape cells are changed. Note that during this part of its computation A' behaves exactly like a nondeterministic two-way finite-state acceptor. As M can only move its window from left to right, it cannot simulate this part of the computation of A' in a step by step fashion. However, M can nondeterministically guess the crossing sequences of A' on all consecutive positions of the tape, which will describe this part of the computation of A' completely. In this way M is able to determine the position and the type of the next change operation applied by A'. Assume that this operation replaces an occurrence of the symbol b by an occurrence of the symbol c, and that A' enters state q' after executing this operation. Then M replaces the content of the tape cell corresponding to the symbol b by the symbol $b' = (c, q', 1)$. Should the positions of the symbols b and a' coincide, which means in particular that $b = a$, then M replaces the symbol a' by b'.
3. If the tape content is of the form $wa'y$ for a symbol $a' = (a, q, 0) \in \Gamma \times Q \times \{0\}$ and some words w, y satisfying $wy \in \Gamma^* \cdot (\Gamma \times Q \times \{1\}) \cdot \Gamma^*$, then M simply rewrites the symbol a' into the symbol a.
4. If the tape content is of the form $wa'z$ for some words $w, z \in \Gamma^*$ and a symbol $a' = (a, q, 1) \in \Gamma \times Q \times \{1\}$, then M replaces the symbol a' by the symbol $(a, q, 0) \in \Gamma \times Q \times \{0\}$.

Note that the third coordinate of the symbols from $\Gamma \times Q \times \{0, 1\}$ is used to distinguish between the positions of the last and the last but one change operation of A' in the current computation. One can easily verify that each accepting computation of M corresponds to an accepting computation of A'. Further, for each accepting computation of A', there exists an accepting computation of M. Thus, we conclude that $L(M) = L(A') = L(A)$ holds.

Finally it remains to show that M is shrinking. For doing so we define a weight function ω by taking $\omega(a) := 3(j+1-s)$ and $\omega((a,q,i)) := 3(j+1-s)+i+1$ for each $a \in \Gamma_s$, $s = 0,\ldots,j$, $i \in \{0,1\}$, and $q \in Q$. Note that each rewrite operation of M satisfies one of the following conditions:

- it replaces an element of Γ_s by an element of $\Gamma_{s+1} \times Q \times \{0,1\}$ for some $s < j$ (in steps 1, 2);
- it replaces an element of $\Gamma_s \times Q \times \{1\}$ by the corresponding element of $\Gamma_s \times Q \times \{0\}$ for some $s \leq j$ (in step 4);
- it replaces an element of $\Gamma_s \times Q \times \{0\}$ by an element of $\Gamma_{s+1} \times Q \times \{1\}$ for some $s < j$ (in step 2);
- it replaces an element of $\Gamma_s \times Q \times \{0\}$ by the corresponding element of Γ_s for some $s \leq j$ (in step 3).

Each of these rewrite steps is weight-reducing with respect to ω. Hence, M is indeed an sRRWW-automaton for the language $L(A)$. □

As a consequence of Lemma 4 and Lemma 5, we obtain the following characterization.

Theorem 6. $\mathcal{L}(\mathsf{FC}) = \mathcal{L}(\mathsf{sRRWW})$.

This characterization allows us to establish some new relationships between (shrinking) restarting automata on the one hand and some more classical language classes on the other hand. Let CSL and DCSL denote the class of context-sensitive languages and the class of deterministic context-sensitive languages, respectively, and let $\mathsf{CSL}_{\mathsf{lin}}$ be the class of all *linear-time bounded context-sensitive languages*. A language L belongs to this class, if there exists a context-sensitive grammar $G = (N, T, S, P)$ that generates L and a constant $c \geq 1$ such that, for all $n \in \mathbb{N}_+$, each word of length n in L has a derivation in G of length at most $c \cdot n$ [1]. Finally, by Q we denote the class of *quasi-realtime languages*. This is the class of languages that are accepted by nondeterministic multi-tape Turing machines in realtime, that is, $\mathsf{Q} = \mathsf{NTIME}(n)$.

Proposition 3. [1, 2, 5] $\mathsf{GCSL} \subsetneq \mathsf{CSL}_{\mathsf{lin}} \subsetneq \mathsf{Q} = \bigcup_{c \geq 1} \mathsf{NTIME}(c \cdot n)$.

From Theorems 2 and 3 of [4] the inclusions $\mathsf{Q} \subseteq \mathcal{L}(\mathsf{FC}) \subseteq \mathsf{DCSL}$ follow. Together with the above characterization of the class $\mathcal{L}(\mathsf{sRRWW})$ this yields our second main result.

Corollary 2. $\mathsf{Q} \subseteq \mathcal{L}(\mathsf{sRRWW}) \subseteq \mathsf{DCSL}$.

Hence, we obtain the relationships depicted in Figure 2, where wmon-RRWW denotes the class of weakly monotone RRWW-automata [11]. This improves on the previously known results, as it was open whether the language classes $\mathcal{L}(\mathsf{RRWW})$ and DCSL are at all comparable under inclusion.

$$\begin{tikzcd}
 & \mathsf{CSL} & \\
 & \mathsf{DCSL} \arrow[u, "?"'] & \\
\mathcal{L}(\mathsf{FC}) \arrow[ur, "?"] & = & \mathcal{L}(\mathsf{sR(R)WW}) \arrow[ul, "?"'] \\
\mathsf{Q} \arrow[u, "?"'] & & \mathcal{L}(\mathsf{RRWW}) \arrow[u, "?"'] \\
\mathsf{CSL}_{\mathsf{lin}} \arrow[u] & & \mathcal{L}(\mathsf{RWW}) \arrow[u, "?"'] \\
\mathsf{GCSL} \arrow[u] & = & \mathcal{L}(\mathsf{wmon\text{-}RRWW}) \arrow[u]
\end{tikzcd}$$

Fig. 2. An unmarked arrow indicates that the inclusion is proper, while a question mark indicates that it is an open problem whether the corresponding inclusion is proper. For those classes that are not connected via directed paths in the diagram it is open whether any inclusions hold.

6 Concluding Remarks

We have investigated the expressive power of the shrinking restarting automaton, which is a rather straightforward generalization of the restarting automaton. We have seen that this generalization yields a very robust model of computation, without increasing the asymptotic upper time and space bounds for the membership problem in comparison to the standard model. Indeed, the model with combined restart and rewrite operations (the shrinking RWW-automaton) is as powerful as the general shrinking RRWW-automaton, and even the additional capabilities of changing the internal state in a restart transition (as opposed to resetting the state to the initial state) and to perform multiple rewrite operations in each cycle do not increase the expressive power of this model.

Finally, we obtained a characterization of the shrinking RRWW-automaton in terms of a certain type of nondeterministic single-tape Turing machines, the so-called finite-change automaton. This characterization implies that the set of languages recognized by shrinking restarting automata is included in the language class DCSL, and that it includes all languages recognized in linear time by multi-tape nondeterministic Turing machines.

However, it remains open whether the shrinking RRWW-automaton is at all more powerful than the standard RRWW-automaton. Further, although we have obtained a new simplified reduction from the language class $\mathcal{L}(\mathsf{RRWW})$ to the language class $\mathcal{L}(\mathsf{RWW})$, we still have no clue whether or not these two classes coincide. Finally, it is not known whether any of the inclusions $\mathsf{Q} \subseteq \mathcal{L}(\mathsf{sRRWW})$ and $\mathcal{L}(\mathsf{sRRWW}) \subseteq \mathsf{DCSL}$ is proper.

Here we have only studied shrinking restarting automata with auxiliary letters. An obvious direction for future research is the study of the expressive power and the properties of shrinking restarting automata without auxiliary symbols.

Acknowledgements. The authors want to thank F. Mráz and M. Plátek for many fruitful discussions regarding restarting automata and for pointing them to the work of von Braunmühl and Verbeek.

References

1. R.V. Book, Time-bounded grammars and their languages. *Journal of Computer and System Sciences*, 5 (1971) 397–429.
2. R.V. Book and S. Greibach, Quasi-realtime languages. *Mathematical Systems Theory*, 4 (1970) 97–111.
3. R.V. Book and F. Otto, *String-Rewriting Systems.* Springer, New York, 1993.
4. B. von Braunmühl and R. Verbeek, Finite-change automata. In K. Weihrauch (ed.), *4th GI Conference, Proc*, *Lecture Notes in Computer Science* 67, Springer, Berlin, 1979, 91–100.
5. G. Buntrock, *Wachsende kontext-sensitive Sprachen.* Habilitationsschrift, Fakultät für Mathematik und Informatik, Universität Würzburg, 1996.
6. G. Buntrock and K. Loryś, On growing context-sensitive languages. In W. Kuich (ed.), *Automata, Languages and Programming, Proc. ICALP'92*, *Lecture Notes in Computer Science* 623, Springer, Berlin, 77–88.
7. G. Buntrock and F. Otto, Growing context-sensitive languages and Church-Rosser languages. *Information and Computation*, 141 (1998) 1–36.
8. E. Dahlhaus and M. Warmuth, Membership for growing context-sensitive grammars is polynomial. *Journal of Computer and System Sciences*, 33 (1986) 456–472.
9. P. Jančar, F. Mráz, M. Plátek and J. Vogel, Restarting automata. In H. Reichel (ed.), *FCT'95, Proc.*, *Lecture Notes in Computer Science* 965, Springer, Berlin, 1995, 283–292.
10. P. Jančar, F. Mráz, M. Plátek and J. Vogel, On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics*, 4 (1999) 287–311.
11. T. Jurdziński, K. Loryś, G. Niemann and F. Otto, Some results on RRW- and RRWW-automata and their relationship to the class of growing context-sensitive languages. *Mathematische Schriften Kassel* 14/01, Universität Kassel, 2001. Also: *Journal of Automata, Languages and Combinatorics*, to appear.
12. T. Jurdziński and F. Otto, On left-monotone restarting automata. *Mathematische Schriften Kassel* 17/03, Universität Kassel, 2003.
13. T. Jurdziński, F. Otto, F. Mráz and M. Plátek, On the complexity of 2-monotone restarting automata. *Mathematische Schriften Kassel* 4/04, Universität Kassel, 2004. An extended abstract appeared in C.S. Calude, E. Calude and M.J. Dinneen (eds.), *Developments in Language Theory, Proc. DLT 2004*, *Lecture Notes in Computer Science* 3340, Springer, Berlin, 2004, 237–248.
14. C. Lautemann, One pushdown and a small tape. In K. Wagner (ed.), *Dirk Siefkes zum 50. Geburtstag*, Technische Universität Berlin and Universität Augsburg, 1988, 42–47.
15. R. McNaughton, P. Narendran and F. Otto, Church-Rosser Thue systems and formal languages. *Journal of the Association for Computing Machinery*, 35 (1988) 324–344.

16. H. Messerschmidt and H. Stamer, Restart-Automaten mit mehreren Restart-Zuständen. In H. Bordihn (ed.), *Workshop "Formale Methoden in der Linguistik" und 14. Theorietag "Automaten und Formale Sprachen", Proc.*, Institut für Informatik, Universität Potsdam, 2004, 111–116.
17. 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, Proc. DLT 1999*, World Scientific, Singapore, 2000, 103–114.
18. G. Niemann and F. Otto, Further results on restarting automata. In M. Ito and T. Imaoka (eds.), *Words, Languages and Combinatorics III, Proc.*, World Scientific, Singapore, 2003, 352–369.
19. K. Oliva, K. Květoň and R. Ondruška, The computational complexity of rule-based part-of-speech tagging. In V. Matousek and P. Mautner (eds.), *TSD 2003, Proc.*, *Lecture Notes in Computer Science* 2807, Springer, Berlin, 2003, 82–89.
20. F. Otto, Restarting automata and their relations to the Chomsky hierarchy. In Z. Esik and Z. Fülöp (eds.), *Developments in Language Theory, Proc. DLT'2003*, *Lecture Notes in Computer Science* 2710, Springer, Berlin, 2003, 55-74.
21. M. Plátek, M. Lopatková and K. Oliva, Restarting automata: Motivations and applications. In M. Holzer (ed.), *Workshop "Petrinets" und 13. Theorietag "Automaten und Formale Sprachen"*, Institut für Informatik, Technische Universität München, Garching, 2003, 90–96.