Lower Bounds for Nonforgetting Restarting Automata and CD-Systems of Restarting Automata

Friedrich Otto

Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

August 25, 2008

Abstract

Some lower bound results are presented that separate the language classes accepted by certain types of nonforgetting restarting automata or CD-systems of restarting automata from each other.

1 Introduction

The *nonforgetting* restarting automaton is a generalization of the restarting automaton that, when executing a restart operation, changes its internal state based on the current state and the actual contents of its read/write window instead of resetting it to the initial state [13]. The expressive power of various monotone and/or deterministic types of nonforgetting restarting automata has been investigated in [10]. Another generalization of the restarting automaton is the *cooperating distributed system* (*CD-system*) of restarting automata [11]. Here a finite system of restarting automata works together in analyzing a given sentence, where they interact based on a given *mode of operation*. As it turned out, CD-systems of restarting automata of some type X working in mode $= 1$ are just as expressive as nonforgetting restarting automata of the same type X. Further, in [12] various types of determinism are introduced for CD-systems of restarting automata called *strict determinism*, *global determinism*, and *local determinism*, and it is shown that globally deterministic CD-systems working in mode $= 1$ correspond to deterministic nonforgetting restarting automata.

Here we derive some lower bound results for some types of nonforgetting restarting automata and for some types of CD-systems of restarting automata. In this way we establish separations between the corresponding language classes, thus providing detailed technical proofs for some of the separation results announced in [9, 10, 12].

2 Definitions

We first describe in short the types of restarting automata we will be dealing with. Then we restate the definition of CD-systems of restarting automata from [11].

An RRWW-*automaton* 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, the symbols $¢, \$ \not\in \Gamma$ 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 δ is the *transition relation* that associates a finite set of *transition steps* to each pair (q, u) consisting of a state $q \in Q$ and a possible contents u of the read/write window. There are four types of transition steps:

– *Move-right steps* of the form (q', MVR) with $q' \in Q$. This step causes M to shift the read/write window one position to the right and to enter state q'. However, the window cannot be shifted across the right border marker \$.

– *Rewrite steps* of the form (q', v), where $q' \in Q$, and v is a string satisfying $|v| < |u|$. This step 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.

– *Restart steps* of the form $\mathsf{Restart}$, which cause M to place the read/write window over the left end of the tape, so that the first symbol it contains is the left border marker ¢, and to reenter the initial state q_0.

– *Accept steps* of the form Accept, which cause M to halt and accept.

If $\delta(q, u) = \emptyset$ for some pair (q, u), then M necessarily halts, and we say that M *rejects* in this situation. There is one additional restriction that the transition relation must satisfy: ignoring move operations, *rewrite steps and restart steps alternate* in any computation of M, with a rewrite step coming first. However, it is more convenient to describe M by so-called *meta-instructions* (see below).

A *configuration* of M is described by a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \varepsilon$ (the empty word) 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.*

In general, an RRWW-automaton is nondeterministic, that is, to some configurations several different instructions may apply. If that is not the case, then the automaton is called *deterministic.* To describe M more transparently, we can use a finite sequence of so-called meta-instructions (see, e.g., [15]).

A *rewriting meta-instruction* for M has the form $(E_1, u \to v, E_2)$, where E_1 and E_2 are regular expressions, and $u, v \in \Gamma^*$ are words satisfying $k \geq |u| > |v|$. To execute a cycle M chooses a meta-instruction of the form $(E_1, u \to v, E_2)$. On trying to execute this meta-instruction M will get stuck (and so reject) starting from the *restarting configuration* $q_0 ¢ w \$$, if w does not admit a factorization of the form $w = w_1 u w_2$ such that $¢ w_1 \in E_1$ and $w_2 \$ \in E_2$. On the other hand, if w does have factorizations of this form, then one such factorization is chosen nondeterministically, and $q_0 ¢ w \$$ is transformed into the restarting configuration $q_0 ¢ w_1 v w_2 \$$. This computation, which is called a *cycle*, is expressed as $w \vdash^c_M w_1 v w_2$. In order to describe the tails of accepting computations we use *accepting meta-instructions* of the form (E_1, Accept), which simply accepts the strings from the regular language E_1.

An input word $w \in \Sigma^*$ is *accepted* by M, if there is a computation of M which starts with the initial configuration $q_0 ¢ w \$$, and which finishes by executing an accepting meta-instruction. By $L(M)$ we denote the language consisting of all words accepted by M.

We are also interested in various restricted types of restarting automata. They are obtained by combining two types of restrictions:

(a) Restrictions on the movement of the read/write window (expressed by the first part of the class name): RR- denotes no restriction, and R- means that each rewrite step is immediately followed by a restart.

(b) Restrictions on the rewrite-instructions (expressed by the second part of the class name): -WW denotes no restriction, -W means that no auxiliary symbols are available (that is, $\Gamma = \Sigma$), and -ε means that no auxiliary symbols are available and that each rewrite step is simply a deletion (that is, if $(q', v) \in \delta(q, u)$ is a rewrite step of M, then v is obtained from u by deleting some symbols).

A *cooperating distributed system of* RRWW-*automata* (or a CD-RRWW-system for short) consists of a finite collection $\mathcal{M} := ((M_i, \sigma_i)_{i \in I}, I_0)$ of RRWW-automata $M_i = (Q_i, \Sigma, \Gamma_i, ¢, \$, q_0^{(i)}, k, \delta_i)$ $(i \in I)$, *successor relations* $\sigma_i \subseteq I$ $(i \in I)$, and a subset $I_0 \subseteq I$ of *initial indices*. Here it is required that $Q_i \cap Q_j = \emptyset$ for all $i, j \in I$, $i \neq j$, that $I_0 \neq \emptyset$, that $\sigma_i \neq \emptyset$ for all $i \in I$, and that $i \notin \sigma_i$ for all $i \in I$. Further, let m be one of the following *modes of operation*, where $j \geq 1$:

$= j$	:	execute exactly j cycles;
t	:	continue until no more cycle can be executed.

The computation of $\mathcal{M}$ in mode $= j$ on an input word w proceeds as follows. First an index $i_0 \in I_0$ is chosen nondeterministically. Then the RRWW-automaton M_{i_0} starts the computation with the initial configuration $q_0^{(i_0)} ¢ w \$$, and executes j cycles. Thereafter an index $i_1 \in \sigma_{i_0}$ is chosen nondeterministically, and M_{i_1} continues the computation by executing j cycles. This continues until, for some $l \geq 0$, the automaton M_{i_l} accepts. Should at some stage the chosen automaton be unable to execute the required number of cycles, then the computation fails.

In mode t the chosen automaton M_{i_l} continues with the computation until it either accepts, in which case $\mathcal{M}$ accepts, or until it can neither execute another cycle nor an accepting tail, in which case an automaton $M_{i_{l+1}}$ with $i_{l+1} \in \sigma_{i_l}$ takes over. Should at some stage the chosen automaton be unable to execute a single cycle or an accepting tail, then the computation of $\mathcal{M}$ fails.

By $L_{\mathrm{m}}(\mathcal{M})$ we denote the language that the CD-RRWW-system $\mathcal{M}$ accepts in mode m. It consists of all words $w \in \Sigma^*$ that are accepted by $\mathcal{M}$ in mode m as described above. If X is any of the above types of restarting automata, then a CD-X-system is a CD-RRWW-system for which all component automata are of type X.

The *nonforgetting restarting automaton* [13] is a generalization of the restarting automaton that is obtained by combining restart transitions with a change of state just like the move-right and rewrite transitions. This allows a nonforgetting restarting automaton M to carry some information from one cycle to the next. We use the notation $(q_1, x) \vdash_M^c (q_2, y)$

to denote a cycle of M that transforms the restarting configuration $q_1¢x\$$ into the restarting configuration $q_2¢y\$$.

Three different notions of determinism have been introduced for CD-systems of restarting automata.

The first notion is motivated by the way in which determinism is used in CD-grammar systems (see, e.g., [2, 3]). A CD-system $\mathcal{M} := ((M_i, \sigma_i)_{i\in I}, I_0)$ of restarting automata is called *locally deterministic* if M_i is a deterministic restarting automaton for each $i \in I$. As the start component is chosen nondeterministically from among all automata M_j with $j \in I_0$, and as in each round the successor component is chosen nondeterministically from among all automata M_j with $j \in \sigma_i$, computations of a locally deterministic CD-system of restarting automata are in general not completely deterministic.

To avoid this remaining nondeterminism we strengthen the above definition. We call a CD-system $\mathcal{M} := ((M_i, \sigma_i)_{i\in I}, I_0)$ *strictly deterministic* if I_0 is a singleton, if M_i is a deterministic restarting automaton, if $|\sigma_i| = 1$ for each $i \in I$, and if the mapping $\sigma : I \to I$ that maps each component to its unique successsor is a bijection.

Remark 2.1 *Let $\mathcal{M} := ((M_i, \sigma_i)_{i\in I}, I_0)$ be a strictly deterministic CD-system of restarting automata. Because of the above restrictions on I_0 and the successor sets σ_i, we can assume without loss of generality that $I = \{1, \dots, n\}$ for some positive integer n, that $I_0 = \{1\}$, that $\sigma_i = \{i + 1\}$ for all $i = 1, \dots, n - 1$, and that $\sigma_n = \{1\}$. This means that each sufficiently long computation of $\mathcal{M}$ consists of a sequence of meta-cycles of the form*

$$w_0 \vdash^c_{M_1} w_1 \vdash^c_{M_2} \cdots \vdash^c_{M_{n-1}} w_{n-1} \vdash^c_{M_n} w_n.$$

Thus, if $\mathcal{M}$ does not have any auxiliary symbols, then with $w_0 \in L_{=1}(\mathcal{M})$ we also have $w_n \in L_{=1}(\mathcal{M})$, that is, $\mathcal{M}$ satisfies a variant of the correctness preserving property for deterministic restarting automata (see, e.g., [4]). In [12] the notion of strict determinism was given without the requirement that the mapping σ is a bijection. In that case a computation may never return to the initial component, which means that the correctness preserving property will in general not be of any help.

The restriction of having only a single possible successor for each component is a rather serious one, as we will see below. Thus, we define a third notion. A CD-system $\mathcal{M} := ((M_i, \sigma_i)_{i\in I}, I_0)$ is called *globally deterministic* if I_0 is a singleton, if M_i is a deterministic restarting automaton for each $i \in I$, and if, for each $i \in I$, each restart operation of M_i is combined with an index from the set σ_i. Thus, when M_i finishes a part of a computation according to the actual mode of operation by executing the restart operation $\delta_i(q, u) = (\mathsf{Restart}, r)$, where $r \in \sigma_i$, then the component M_r takes over. For example, when working in mode $= j$ for some $j > 1$, then the first $j - 1$ applications of restart steps within the computation of a component M_i just restart M_i itself, but at the j-th application of a restart step, the component $r \in \sigma_i$ becomes active, if r is the index associated with this particular restart operation. In this way it is guaranteed that all computations of a globally deterministic CD-system are deterministic. However, for a component M_i there can still be several possible successors. This is reminiscent of the way in which nonforgetting restarting automata work.

We use the prefix det-local to denote locally deterministic CD-systems, the prefix det-global to denote globally deterministic CD-systems, and the prefix det-strict to denote strictly deterministic CD-systems. For each type of restarting automaton $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW},$

$\mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$, it is easily seen that the following inclusions hold:

$$\mathcal{L}(\text{det-X}) \subseteq \mathcal{L}_{=1}(\text{det-strict-CD-X}) \subseteq \mathcal{L}_{=1}(\text{det-global-CD-X}).$$

Concerning globally deterministic CD-systems, the following results have been obtained which correspond to the results for nondeterministic CD-systems established in [11].

Theorem 2.2 [9] *If M is a nonforgetting deterministic restarting automaton of type* X *for some* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*, then there exists a globally deterministic CD-system $\mathcal{M}$ of restarting automata of type* X *such that $L_{=1}(\mathcal{M}) = L(M)$.*

For the converse we even have the following stronger result.

Theorem 2.3 [9] *For each* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*, if $\mathcal{M}$ is a globally deterministic* CD-X*-system, and if j is a positive integer, then there exists a nonforgetting deterministic* X*-automaton M such that $L(M) = L_{=j}(\mathcal{M})$.*

In addition, we have the following inclusion result.

Theorem 2.4 [9] *For each type* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*, and each integer $j \geq 1$, $\mathcal{L}_{=j}(\text{det-global-X}) \subseteq \mathcal{L}_{=1}(\text{det-local-X})$.*

3 A Lower Bound for Locally Deterministic CD-RRW-Systems

Here we separate the locally deterministic CD-RR- and CD-RRW-systems from the nondeterministic CD-RR- and CD-RRW-systems. For that we will use the following example language.

Definition 3.1 *Let $\Sigma_1 := \{a, b, c\}$, let $\varphi_1 : \Sigma_1^* \to \Sigma_0^*$ be the morphism that is defined by $a \mapsto a$, $b \mapsto b$, and $c \mapsto a$, and let $\varphi_2 : \Sigma_1^* \to \Sigma_0^*$ be the morphism that is defined by $a \mapsto a$, $b \mapsto b$, and $c \mapsto b$. Now we define the language L_{cc} as follows:*

$$L_{cc} := \{\, uc\varphi_1(u)c\varphi_2(u) \mid u \in \Sigma_1^* \,\}.$$

Observe that the morphism $(\varphi_1, \varphi_2) : \Sigma_1^* \to (\Sigma_0^* \times \Sigma_0^*)$ is injective. Thus, for each word $u \in \Sigma_1^*$, there exists a unique pair $(v, w) \in \Sigma_0^* \times \Sigma_0^*$ such that $ucvcw \in L_{cc}$, and for each pair $(v, w) \in \Sigma_0^* \times \Sigma_0^*$, there is at most one word $u \in \Sigma_1^*$ such that $ucvcw \in L_{cc}$.

Lemma 3.2 *L_{cc} is accepted by a* CD-RR*-system working in mode $= 1$.*

Proof. Let $\mathcal{M} := ((M_1, \{2\}), (M_2, \{3\}), (M_3, \{1\}), \{1\})$ be the following CD-RR-system with input alphabet Σ_1, where the RR-automata M_1, M_2, and M_3 are given through the following meta-instructions:

$$\begin{array}{rcl}
M_1 & : & (\text{¢} \cdot \Sigma_1^* \cdot ac \cdot \Sigma_0^* \cdot ac \cdot \Sigma_0^*, a \cdot \$ \to \$, \varepsilon), \\
 & & (\text{¢} \cdot \Sigma_1^* \cdot bc \cdot \Sigma_0^* \cdot bc \cdot \Sigma_0^*, b \cdot \$ \to \$, \varepsilon), \\
 & & (\text{¢} \cdot \Sigma_1^* \cdot cc \cdot \Sigma_0^* \cdot ac \cdot \Sigma_0^*, b \cdot \$ \to \$, \varepsilon), \\
 & & (\text{¢} \cdot cc \cdot \$, \mathsf{Accept}), \\
M_2 & : & (\text{¢} \cdot \Sigma_1^+ \cdot c \cdot \Sigma_0^*, xc \to c, \Sigma_0^* \cdot \$) \text{ for all } x \in \Sigma_0, \\
M_3 & : & (\text{¢} \cdot \Sigma_1^*, yc \to c, \Sigma_0^* \cdot c \cdot \Sigma_0^* \cdot \$) \text{ for all } y \in \Sigma_1.
\end{array}$$

Given an input $z \in \Sigma_1^*$, it is obvious that M_1 will immediately reject, if $|z|_c \leq 1$. Thus, assume that w has the form $z = ucvcw$, where $u \in \Sigma_1^*$ and $v, w \in \Sigma_0^*$. Observe that $|u| = |v| = |w|$ must hold if $w = ucvcw$ belongs to the language L_{cc}. If $u = v = w = \varepsilon$, then M_1 accepts immediately. Observe that $w = cc$ belongs to the language L_{cc}. If only one or two of the factors u, v, or w are empty, then M_1 rejects immediately. Otherwise, M_1 compares the last letter of u, say d, to the last letter of v, say e, and the last letter of w, say f. If $\varphi_1(d) = e$ and $\varphi_2(d) = f$, then f is deleted, and M_2 becomes active; otherwise, M_1 halts and rejects. In the latter case $w = ucvcw$ does not belong to L_{cc}, while in the former case M_2 simply deletes the letter e, and then M_3 deletes the letter d. Thus, $\mathcal{M}$ has executed the sequence of cycles $z = ucvcw = u_1dcv_1ecw_1f \vdash_{\mathcal{M}}^{c^3} u_1cv_1cw_1$. Now $z \in L_{cc}$ if and only if $u_1cv_1cw_1 \in L_c$, and hence, it follows inductively that $L_{=1}(\mathcal{M}) = L_{cc}$. □

Observe that the RR-automata M_1 to M_3 in the construction above are nondeterministic, as they must guess the last and the last but one occurrence of the symbol c on the tape while scanning the tape from left to right. In fact, we claim that L_{cc} is not accepted by any locally deterministic CD-RRW-system working in mode $= 1$. To establish this claim we need some preparations. In particular, we will make use of the notion of *Kolmogorov complexity* and its properties [8, 6]. Here we use $K(x)$ to denote the Kolmogorov complexity of a word x over a finite alphabet of cardinality at least two.

A word $x \in \Sigma_1^+$ is called *incompressible* if $K(x) \geq |x|$ holds. It is *c-incompressible* for a constant $c \in \mathbb{N}_+$, if $K(x) \geq |x| - c$ holds. Finally, x is called *random* if $K(x) > |x| - 4 \cdot \log_3 |x|$ holds. Here $\log_3$ denotes the logarithm to base 3.

Lemma 3.3 *For each $c \geq 0$ and each $n \geq 1$, there are more than $(1 - \frac{1}{2 \cdot 3^c}) \cdot 3^n$ words of length n over Σ_1 that are c-incompressible.*

Proof. There are 3^n words of length n over Σ_1. On the other hand, there are only $\sum_{i=0}^{n-c-1} 3^i = \frac{3^{n-c}-1}{2}$ many words of length at most $n-c-1$ over Σ_1. Thus, at most this many words of length n have a description of length at most $n-c-1$. Thus, at least $3^n - \frac{3^{n-c}-1}{2} \geq (1 - \frac{1}{2 \cdot 3^c}) \cdot 3^n$ many words of length n over Σ_1 are c-incompressible. □

Not all factors of incompressible words are themselves incompressible. However, we have the following result saying that sufficiently long suffixes of c-incompressible words are random.

Proposition 3.4 *For each $m \in \mathbb{N}_+$, there exists an integer $n_0 \in \mathbb{N}_+$ such that the following statement holds for all $n \geq n_0$ and all c-incompressible words $u \in \Sigma_1^n$: If u is factored as $u = u_1 u_2 \cdots u_m \hat{u}$ such that $|u_i| = \log_3^2(n)$, $1 \leq i \leq m$, then the suffix $u^{(i)} = u_{i+1} \cdots u_m \hat{u}$ of u is random for each $1 \leq i \leq m$.*

Proof. Let $i \in \{1, \ldots, m\}$, and let $u^{(i)} = u_{i+1} \cdots u_m \hat{u}$, that is, $u = u_1 \cdots u_i u^{(i)}$. Then $|u^{(i)}| = |u| - |u_1 \cdots u_i| = n - i \cdot \log_3^2(n)$. We have to prove that $K(u^{(i)}) > |u^{(i)}| - 4 \cdot \log_3 |u^{(i)}|$ holds.

For $r \in \mathbb{N}_+$, let $\mathrm{bin}(r)$ denote the binary representation of r. If $p^{(i)}$ is a (shortest) description for $u^{(i)}$, then $p := 0^{|\mathrm{bin}(|u_1 \cdots u_i|)|} \cdot 1 \cdot \mathrm{bin}(|u_1 \cdots u_i|) \cdot u_1 \cdots u_i \cdot p^{(i)}$ is a description for the word u. Indeed from p a program can first extract the information $|\mathrm{bin}(|u_1 \cdots u_i|)|$, which it can use to determine $\mathrm{bin}(|u_1 \cdots u_i|)$ and then $u_1 \cdots u_i$. Finally, from the remaining suffix of p, which is $p^{(i)}$, it can determine the suffix $u^{(i)}$ of u. Thus, we see that $|p| \geq K(u) \geq n - c$ holds. As

$$\begin{aligned}
|p| &= 2 \cdot |\mathrm{bin}(|u_1 \cdots u_i|)| + 1 + |u_1 \cdots u_i| + |p^{(i)}| \\
&= 2 \cdot \log_2(i \cdot \log_3^2(n)) + 1 + i \cdot \log_3^2(n) + |p^{(i)}| \\
&= 2 \cdot \log_2(i) + 4 \cdot \log_2(\log_3(n)) + 1 + i \cdot \log_3^2(n) + |p^{(i)}|,
\end{aligned}$$

it follows that

$$\begin{array}{rcl} |p^{(i)}| & \geq & n - c - 2 \cdot \log_2(i) - 4 \cdot \log_2(\log_3(n)) - 1 - i \cdot \log_3^2(n) \\ & = & n - (c + 1 + 2 \cdot \log_2(i)) - 4 \cdot \log_2(\log_3(n)) - i \cdot \log_3^2(n). \end{array}$$

On the other hand, we have

$$|u^{(i)}| - 4 \cdot \log_3 |u^{(i)}| \quad = \quad n - i \cdot \log_3^2(n) - 4 \cdot \log_3(n - i \cdot \log_3^2(n)).$$

Since $i \leq m$, where m is a fixed constant, we see that for all sufficiently large values of n,

$$4 \cdot \log_3(n - i \cdot \log_3^2(n)) \quad > \quad (c + 1 + 2 \cdot \log_2(i)) + 4 \cdot \log_2(\log_3(n))$$

holds, which implies that $K(u^{(i)}) = |p^{(i)}| > |u^{(i)}| - 4 \cdot \log_3 |u^{(i)}|$ holds. Thus, $u^{(i)}$ is random.
□

The next proposition, which is taken from [6], is concerned with the behaviour of a deterministic finite-state acceptor on a random word of sufficient length.

Proposition 3.5 *Let A be a deterministic finite-state acceptor with tape alphabet $\Gamma \supseteq \Sigma \neq \emptyset$. Then there exists a constant $n_0 \in \mathbb{N}_+$ such that, for each integer $n > n_0$ and each random word $w \in \Sigma^n$, the following condition is satisfied for each word $v \in \Sigma^+$: Assume that A is in state q when it enters the factor wv on its tape from the left, and that it reaches state q' when its head is located inside the factor v. Then A already encounters state q' while its head is still inside the prefix of w of length $\log_s^2 n$, where $s = |\Sigma|$.*

Assume that $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ is a locally deterministic CD-RRW-system such that $L_{=1}(\mathcal{M}) = L_{cc}$. For each $i \in I$, $M_i = (Q^{(i)}, \Sigma_1, \Sigma_1, ¢, \$, q_0^{(i)}, k, \delta_i)$ is a deterministic RRW-automaton that can also be described through a finite sequence of rewriting meta-instructions $I_j^{(i)} = (E_j^{(i)}, u_j^{(i)} \to v_j^{(i)}, F_j^{(i)})$, $1 \leq j \leq n_i$, and a single accepting meta-instruction $I_0^{(i)} = (E_0^{(i)}, \mathsf{Accept})$.

As each M_i is deterministic, the above meta-instructions are used as follows. Assume that M_i is in the restarting configuration $q_0^{(i)}¢w\$$. Then M_i scans the tape from left to right until it detects the shortest prefix w_1 of w such that $w_1 = w_3 u_j^{(i)}$ and $¢w_3 \in L(E_j^{(i)})$ for some $j \in \{1, \ldots, n_i\}$. It then rewrites $u_j^{(i)}$ into $v_j^{(i)}$ and checks whether the corresponding suffix w_2 of w satisfies the condition that $w_2\$ \in L(F_j^{(i)})$. At the same time it checks whether the original tape content $¢w\$$ belongs to the language $L(E_0^{(i)})$. If the latter holds, then M_i halts and accepts; if $¢w\$ \notin L(E_0^{(i)})$, but $w_2\$ \in L(F_j^{(i)})$, then M_i restarts, which yields the restarting configuration $q_0^{(r)}¢w_3 v_j^{(i)} w_2\$$ for some $r \in \sigma_i$. Finally, if $w_2\$ \in L(F_j^{(i)})$ does not hold, either, then M_i halts and rejects. If no prefix of the above form is found, then M_i halts and rejects as well, unless $¢w\$ \in L(E_0^{(i)})$ holds, in which case M_i halts and accepts.

Thus, we can replace each regular constraint $E_j^{(i)}$ by a regular constraint $\hat{E}_j^{(i)}$ such that

$$\begin{array}{rcl} L(\hat{E}_j^{(i)}) & = & \{ ¢w \mid \; ¢w \in L(E_j^{(i)}), \text{ but for all } x, y \in \Sigma_1^*, \, y \neq \varepsilon : \text{ if } ¢xy = ¢wu_j^{(i)}, \\ & & \quad \text{then } ¢x \notin \bigcup_{r=1}^{n_i} (L(E_r^{(i)}) \cdot u_r^{(i)}) \}, \end{array}$$

and the resulting meta-instructions $\hat{I}_j^{(i)} = (\hat{E}_j^{(i)}, u_j^{(i)} \to v_j^{(i)}, F_j^{(i)})$ $(1 \leq j \leq n_i)$ will also describe M_i together with $I_0^{(i)}$. The advantage of the new constraints is the following:

If $¢w\$ \in \bigcup_{j=1}^{n_i} \left(L(\hat{E}_j^{(i)}) \cdot u_j^{(i)} \cdot \Sigma_1^* \cdot \$ \right)$, then there exist a unique index $j \in \{1, \ldots, n_i\}$ and a unique factorization $w = w_1 u_j^{(i)} w_2$ such that $¢w_1 \in L(\hat{E}_j^{(i)})$ holds. Observe, however, that in general the intersection $L(E_0^{(i)}) \cap \bigcup_{j=1}^{n_i} \left(L(\hat{E}_j^{(i)}) \cdot u_j^{(i)} \cdot \Sigma_1^* \cdot \$ \right)$ will not be empty, that is, some words are accepted by M_i in tail computations that have a prefix from the language $L(\hat{E}_j^{(i)}) \cdot u_j^{(i)}$ for some value of j.

With M_i we can now associate a deterministic finite-state acceptor A_i such that $L(A_i) = \bigcup_{j=1}^{n_i} \left(L(\hat{E}_j^{(i)}) \cdot u_j^{(i)} \right)$. Thus, given an input of the form $¢w\$$, where $w \in \Sigma_1^*$, A_i will determine the shortest prefix $¢w_1$ such that $w_1 = w_3 u_j^{(i)}$ and $¢w_3 \in L(E_j^{(i)})$ for some index $1 \leq j \leq n_i$. Finally, we define a deterministic finite-state acceptor (without initial state) $\mathcal{A}$ as the disjoint union of the finite-state acceptors A_i, $i \in I$.

For the following discussion we distinguish between *inner rewrites* and *suffix rewrites* of $\mathcal{M}$. A rewrite step $u_j^{(i)} \to v_j^{(i)}$ is a *suffix rewrite*, if $u_j^{(i)}$ (and therewith also $v_j^{(i)}$) ends with the right delimiter \$; otherwise it is called an *inner rewrite*. By ρ_{suf} we denote the number of suffix rewrite operations of $\mathcal{M}$.

Proposition 3.6 *For each $m \in \mathbb{N}_+$, there exists an integer $n_0 \in \mathbb{N}_+$ such that the following statement holds for all $n \geq n_0$ and all c-incompressible words $u \in \Sigma_1^n$: If $\mathcal{M}$ starts from an initial configuration corresponding to the input $z = uc\varphi_1(u)c\varphi_2(u)$, then in any resulting computation of $\mathcal{M}$ the first m inner rewrites are executed within the prefix of u of length $m \cdot \log_3^2(n)$.*

Proof. Let $m \in \mathbb{N}_+$ be a constant. For m we obtain a constant $n_0^{(1)} \in \mathbb{N}_+$ from Proposition 3.4, and for the deterministic finite-state acceptor $\mathcal{A}$, we get a constant $n_0^{(2)} \in \mathbb{N}_+$ from Proposision 3.5. Thus, we can choose $n_0 := \max\{n_0^{(1)}, n_0^{(2)}\}$.

Now let $n \geq n_0$, and let $u \in \Sigma_1^n$ be a c-incompressible word. We can factor u as $u = u_1 u_2 \cdots u_m \hat{u}$ such that $|u_i| = \log_3^2(n)$, $1 \leq i \leq m$. Then we know from Proposition 3.4 that the suffix $u^{(i)} = u_{i+1} \cdots u_m \hat{u}$ of u is random for each $1 \leq i \leq m$.

Consider a computation of $\mathcal{M}$ starting from the initial configuration $q_0^{(i_0)} ¢uc\varphi_1(u)c\varphi_2(u)\$$, where $i_0 \in I_0$. If in the first cycle an inner rewrite is executed, then $\mathcal{A}$ must reach a corresponding state from the state that corresponds to the initial state of A_{i_0}. By Proposition 3.5 this happens already inside the prefix u_1. Continuing with the computation of $\mathcal{M}$, the next inner rewrite is either also executed on (the successor of) the prefix u_1, or the same argument can be used to show that it is executed on the factor u_2. Inductively it follows that the first m inner rewrites are executed on the prefix $u_1 \cdots u_m$ of u of length $m \cdot \log_3^2(n)$. □

For the following considerations we denote the constant n_0 from Proposition 3.6 by $n_0(m)$, as it depends on the chosen value of m.

Proposition 3.7 *There exists a constant $\gamma \in \mathbb{N}$ such that the following statement holds for all $m \in \mathbb{N}_+$: If $u \in \Sigma_1^n$ is a c-incompressible word of length $n \geq n_0(m)$, then before the first inner rewrite and in between any two of the first m inner rewrites that $\mathcal{M}$ executes in an accepting computation starting from an initial configuration corresponding to the input $uc\varphi_1(u)c\varphi_2(u)$, $\mathcal{M}$ executes at most γ suffix rewrites.*

Proof. For all $i \in I$ and all $1 \leq j \leq n_i$, if the rewrite step $u_j^{(i)} \to v_j^{(i)}$ is an inner rewrite, then let $B_j^{(i)}$ be a finite-state acceptor for the language $L(F_j^{(i)})$, and if $u_j^{(i)} \to v_j^{(i)}$ is a suffix rewrite, then let $B_j^{(i)}$ be a finite-state acceptor for the language $L(\hat{E}_j^{(i)})$. Assume that $B_j^{(i)}$ has $b_j^{(i)} \geq 1$ internal states, and let $\beta := \prod_{i\in I} \prod_{j=1}^{n_i} b_j^{(i)}$.

A suffix rewrite $x \to y$ has the form $x = x_1\$$ and $y = y_1\$$ for some words $x_1 \in \Sigma_1^{k-1}$ and $y_1 \in \Sigma_1^*$ of length $|y_1| \leq k-2$. The next suffix rewrite $x' \to y'$ that follows after executing the suffix rewrite $x \to y$ necessarily satisfies the condition that $x' = \hat{x}y_1\$$ for some word $\hat{x} \in \Sigma_1^{k-1-|y_1|}$. In particular, a sequence of ℓ suffix rewrites replaces a suffix of length at most $\ell \cdot (k-1)$ of $\varphi_2(u)$ by the right-hand side y of the last rewrite in that sequence.

Now assume that $m' < m$ is minimal such that in the accepting computation considered $\mathcal{M}$ executes a sequence of suffix rewrites of length $\alpha > \rho_{suf} \cdot \beta$ between the m'-th and the $m'+1$-st inner rewrite step. Hence, at least one of the suffix rewrites of $\mathcal{M}$, say $x = x_1\$ \to y_1\$ = y$, is used at least $\beta + 1$ times. Let u' be the successor of the word u that is produced from u by the first m' inner rewrites, and let w' be the successor of the word $\varphi_2(u)$ that is produced by the suffix rewrites that are executed before and between the first m' inner rewrites, that is, $\varphi_2(u) = \hat{w}\tilde{w}$ and $w' = \hat{w}y'$, where $\tilde{w}$ is rewritten into y' by these suffix rewrites. Then the computation considered contains a subcomputation of the following form:

$$\begin{array}{rcl} ¢u'c\,\varphi_1(u)cw'\$ & = & ¢u'c\,\varphi_1(u)c\,w_0w_1\cdots w_\beta w_{\beta+1}y'\$ \\ & \vdash_{\mathcal{M}}^{s_0} & ¢u'c\,\varphi_1(u)c\,w_0w_1\cdots w_\beta y_1\$ \\ & \vdash_{\mathcal{M}}^{s_1} & ¢u'c\,\varphi_1(u)c\,w_0w_1\cdots w_{\beta-1}y_1\$ \\ & \vdash_{\mathcal{M}}^{s_2} & \cdots \\ & \vdash_{\mathcal{M}}^{s_{\beta-1}} & ¢u'c\,\varphi_1(u)c\,w_0w_1y_1\$ \\ & \vdash_{\mathcal{M}}^{s_\beta} & ¢u'c\,\varphi_1(u)c\,w_0y_1\$. \end{array}$$

In each of the cycles preceding this subcomputation the factor $w_0w_1\cdots w_\beta$ is scanned by the active component of $\mathcal{M}$, which means that one of the finite-state acceptors $B_j^{(i)}$ is used to read this factor. Consider the positions that correspond to the first letter of $w_1, w_2, \ldots, w_\beta$ and $w_{\beta+1}$. As these are $\beta+1$ many positions, we see from the definition of β that there are indices $1 \leq r < s \leq \beta+1$ such that each of the $B_j^{(i)}$ used is in the same state when reading the first letter of w_r and the first letter of w_s.

Now consider the input

$$\hat{u} := uc\,\varphi_1(u)cw_0w_1\cdots w_r(w_{r+1}\cdots w_s)^2w_{s+1}\cdots w_\beta w_{\beta+1}\tilde{w}$$

that is obtained from $uc\,\varphi_1(u)c\,\varphi_2(u)$ by repeating the factor $w_{r+1}\cdots w_s$. Then $\hat{u} \notin L_{cc}$, as the third syllable is too long. However, given $\hat{u}$ as input, $\mathcal{M}$ can execute the following computation:

$$\begin{array}{ll} \multicolumn{2}{l}{¢uc\,\varphi_1(u)cw_0w_1\cdots w_r(w_{r+1}\cdots w_s)^2w_{s+1}\cdots w_\beta w_{\beta+1}\tilde{w}\$} \\ \vdash_{\mathcal{M}}^{*} & ¢u'c\,\varphi_1(u)cw_0w_1\cdots w_r(w_{r+1}\cdots w_s)^2w_{s+1}\cdots w_\beta w_{\beta+1}y'\$ \\ \vdash_{\mathcal{M}}^{s_0} & ¢u'c\,\varphi_1(u)cw_0w_1\cdots w_r(w_{r+1}\cdots w_s)^2w_{s+1}\cdots w_\beta y_1\$ \\ \vdash_{\mathcal{M}}^{*} & ¢u'c\,\varphi_1(u)cw_0w_1\cdots w_rw_{r+1}\cdots w_sw_{r+1}\cdots w_sy_1\$ \\ \vdash_{\mathcal{M}}^{*} & ¢u'c\,\varphi_1(u)cw_0w_1\cdots w_rw_{r+1}\cdots w_sy_1\$ \\ \vdash_{\mathcal{M}}^{*} & ¢u'c\,\varphi_1(u)cw_0y_1\$. \end{array}$$

This, however, contradicts the error preserving property for $\mathcal{M}$. Thus, before the first inner rewrite and between any two of the first m inner rewrites $\mathcal{M}$ must not execute more than $\gamma := \rho_{suf} \cdot \beta$ suffix rewrites. □

Thus, in the initial part of the above computation up to the application of the m-th inner rewrite step, a suffix w_2 of $\varphi_2(u)$ of length at most $\gamma \cdot m \cdot (k-1)$ is replaced by a word of the form $w_3 y_1$, where $y = y_1\$$ is the righthand side of a suffix rewrite rule, and w_3 is the prefix of w_2 that is not touched by these suffix rewrites.

Using the same pumping argument as in the proof above, it can be shown that $\mathcal{M}$ cannot accept in a tail computation any word that can be reached from an initial configuration on input $uc\,\varphi_1(u)c\,\varphi_2(u)$ by executing less than m inner rewrite steps, if u is a c-incompressible word of length $n \geq n_0(m)$. Thus, we have the following result.

Lemma 3.8 *Let $m \geq 1$, and let $u \in \Sigma_1^n$ be a c-incompressible word of length $n \geq n_0(m)$. Then each accepting computation of $\mathcal{M}$ on input $uc\,\varphi_1(u)c\,\varphi_2(u)$ contains at least m applications of inner rewrite steps.*

Next we will see that many c-incompressible words u of length n are rewritten into the same word u' by the first m inner rewrites of $\mathcal{M}$.

Proposition 3.9 *Let $c \geq 1$ and $m \geq 1$ be constants, and let $n \geq n_0(m)$. Then there exists a word $u' \in \Sigma_1^*$ that has the following properties:*

- *$n - m \cdot k \leq |u'| \leq n - m$;*
- *there are at least $(1 - \frac{1}{2 \cdot 3^c}) \cdot 3^{m-1}$ different c-incompressible words u of length n such that, for each of these words, $\mathcal{M}$ has an accepting computation on input $uc\,\varphi_1(u)c\,\varphi_2(u)$ that rewrites the prefix u into the word u' through the first m inner rewrites of this computation;*
- *u' has the form $u' = u_1'\hat{u}$, where $|\hat{u}| = n - m \cdot \log_3^2(n)$, and each of the c-incompressible words u above has the same suffix $\hat{u}$.*

Proof. According to Lemma 3.3 there are more than $(1 - \frac{1}{2 \cdot 3^c}) \cdot 3^n$ words of length n over Σ_1 that are c-incompressible. On the other hand, by executing m rewrite operations inside the prefix of length $m \cdot \log_3^2(n)$ of a c-incompressible word u of length n, $\mathcal{M}$ reduces the word u to a word $u' \in \Sigma_1^*$ of length n' such that $n - m \cdot k \leq n' \leq n - m$. As there are only $\sum_{i=n-m\cdot k}^{n-m} 3^i \leq \sum_{i=0}^{n-m} 3^i \leq 3^{n-m+1}$ words of this length, we see that there exists a word u' such that at least $(1 - \frac{1}{2 \cdot 3^c}) \cdot \frac{3^n}{3^{n-m+1}} = (1 - \frac{1}{2 \cdot 3^c}) \cdot 3^{m-1}$ different c-incompressible words u of length n are reduced to this particular word u'. As for each of these c-incompressible words of length n the rewrite operations are executed inside the prefix of length $m \cdot \log_3^2(n)$, we see that they all have the same suffix $\hat{u}$ of length $n - m \cdot \log_3^2(n)$, and that $u' = u_1'\hat{u}$ holds for some word $u_1' \in \Sigma_1^*$. □

Let γ be the constant from Proposition 3.7, let $m \in \mathbb{N}_+$ be a constant, and let $n \geq n_0(m)$ such that $n \geq m \cdot \log_3^2(n) + \gamma \cdot m \cdot (k-1)$. Finally, let $u \in \Sigma_1^n$, and let $v := \varphi_1(u)$ and $w := \varphi_2(u)$. Then u can be factored as $u = u_1\hat{u}$ such that $|u_1| = m \cdot \log_3^2(n)$ and $|\hat{u}| \geq \gamma \cdot m \cdot (k-1)$, and w can be factored as $w = w_1 w_2$ such that $|w_2| = \gamma \cdot m \cdot (k-1)$ and $|w_1| \geq m \cdot \log_3^2(n)$. Obviously, the word $z = ucvcw$ belongs to the language L_{cc}, and accordingly $\mathcal{M}$ has accepting computations that start from an initial configuration corresponding to this input.

None of these computations can just consist of an accepting tail, and also none of them can just consist of suffix rewrites (cf. the proof of Proposition 3.7). Indeed if u is c-incompressible, then each of these computations contains at least m applications of inner rewrite steps (Lemma 3.8). We now consider the initial phase of such a computation that ends with the m-th inner rewrite. If u is c-incompressible, then by Proposition 3.6 the m inner rewrites of this initial phase are executed within the prefix u_1 of u. By Proposition 3.7 at most γ suffix rewrites are executed before the first inner rewrite and between any two of these m inner rewrites. Thus, these suffix rewrites only affect the suffix w_2 of w. Hence, the initial phase of the computation considered transforms the initial tape content $ucvcw = u_1\hat{u}cvcw_1w_2$ into a word of the form $u_1'\hat{u}cvcw_1y$, that is, the inner factor $\hat{u}cvcw_1$ is not changed at all during this initial phase. Thus, for each cycle of the partial computation considered, the behaviour of $\mathcal{M}$ on this factor can be expressed by one of the finite-state acceptors $B_j^{(i)}$ ($i \in I$, $1 \leq j \leq n_i$) introduced in the proof of Proposition 3.7.

Let $B_j^{(i)}$ be one of these finite-state acceptors with finite set of states $Q(B_j^{(i)})$ and transition function $\delta(B_j^{(i)})$. Without loss of generality we can assume that $B_j^{(i)}$ is deterministic and complete, that is, for each $q \in Q(B_j^{(i)})$ and each word $x \in \Sigma_1^*$, $\delta(B_j^{(i)})(q,x)$ is a unique and well-defined element of $Q(B_j^{(i)})$. Hence, the word $\hat{u}cvcw_1$ induces a unique mapping $\psi_u(B_j^{(i)}) : Q(B_j^{(i)}) \to Q(B_j^{(i)})$ by taking $\psi_u(B_j^{(i)})(q) := \delta(B_j^{(i)})(q, \hat{u}cvcw_1)$ for all $q \in Q(B_j^{(i)})$. As $B_j^{(i)}$ has $b_j^{(i)}$ many states, we see that there are $(b_j^{(i)})^{b_j^{(i)}}$ such mappings.

This consideration applies to all the finite-state acceptors $B_j^{(i)}$. Thus, with the c-incompressible word u, we can associate a mapping

$$\psi_u : \prod_{i\in I, 1\leq j\leq n_i} Q(B_j^{(i)}) \quad\to \prod_{i\in I, 1\leq j\leq n_i} Q(B_j^{(i)})$$

by taking the product of all mappings $\psi_u(B_j^{(i)})$. Then there are

$$\eta := \prod_{i\in I, 1\leq j\leq n_i} (b_j^{(i)})^{b_j^{(i)}}$$

such mappings. Observe that η is a constant that only depends on the CD-RRW-system $\mathcal{M}$.

Now we are prepared to prove the following lower bound result.

Theorem 3.10 *The language L_{cc} is not accepted by any locally deterministic* CD-RRW-*system working in mode* $= 1$.

Proof. Assume that $\mathcal{M}$ is a locally deterministic CD-RRW-system such that $L_{=1}(\mathcal{M}) = L_{cc}$. Let γ be the corresponding constant from Proposition 3.7, let η be the corresponding constant from the considerations above, let ι be the number of components of $\mathcal{M}$, let k denote the size of the read/write windows of the components of $\mathcal{M}$, and let ρ_{suf} be the number of different suffix rewrite rules of $\mathcal{M}$.

We choose a constant $m \in \mathbb{N}_+$ such that

$$3^{m-2} > \eta \cdot \iota \cdot \gamma \cdot (k-1) \cdot \rho_{suf} \cdot m$$

holds, and let $n \geq n_0(m)$ such that $n \geq m \cdot \log_3^2(n) + \gamma \cdot m \cdot (k-1)$. Then by Proposition 3.9 there exists a word $u' \in \Sigma_1^*$ that has the following properties:

- $n - m \cdot k \leq |u'| \leq n - m$;
- there are at least $(1 - \frac{1}{2\cdot 3^c}) \cdot 3^{m-1}$ different c-incompressible words u of length n such that, for each of these words, $\mathcal{M}$ has an accepting computation on input $uc\varphi_1(u)c\varphi_2(u)$ that rewrites the prefix u into the word u' through the first m inner rewrites of this computation;

- u' has the form $u' = u'_1\hat{u}$, where $|\hat{u}| = n - m \cdot \log_3^2(n)$, and each of the c-incompressible words u above has the same suffix $\hat{u}$.

Then each of these c-incompressible words u can be factored as $u = u_1\hat{u}$ such that $|u_1| = m \cdot \log_3^2(n)$ and $|\hat{u}| \geq \gamma \cdot m \cdot (k-1)$, and $w = \varphi_2(u)$ can be factored as $w = w_1w_2$ such that $|w_2| = \gamma \cdot m \cdot (k-1)$ and $|w_1| \geq m \cdot \log_3^2(n)$. Obviously, the word $z = uc\,\varphi_1(u)cw$ belongs to the language L_{cc}, and accordingly $\mathcal{M}$ has accepting computations that start from an initial configuration corresponding to this input.

As above we consider the initial phase of such a computation that ends with the m-th inner rewrite. If u is c-incompressible, by Proposition 3.6 the m inner rewrites of this initial phase are executed within the prefix u_1 of u. By Proposition 3.7 at most γ suffix rewrites are executed before the first inner rewrite and between any two of these m inner rewrites. Thus, these suffix rewrites only affect the suffix w_2 of w. Hence, the initial phase of the computation considered transforms the initial tape content $uc\,\varphi_1(u)cw = u_1\hat{u}c\,\varphi_1(u)cw_1w_2$ into a word of the form $u'_1\hat{u}c\,\varphi_1(u)cw_1y$, that is, the inner factor $\hat{u}c\,\varphi_1(u)cw_1$ is not changed at all during this initial phase. Thus, as pointed out above this inner factor induces a unique mapping ψ_u on the product of the sets of states of the finite-state acceptors $B_j^{(i)}$ $(i \in I,\ 1 \leq j \leq n_i)$.

There are at least $(1 - \frac{1}{2\cdot 3^c}) \cdot 3^{m-1}$ different c-irreducible words $u \in \Sigma_1^n$ with the above properties. As

$$(1 - \frac{1}{2 \cdot 3^c}) \cdot 3^{m-1} \geq \frac{1}{2} \cdot 3^{m-1} > 3^{m-2} > \eta \cdot \iota \cdot \gamma \cdot (k-1) \cdot \rho_{suf} \cdot m,$$

as there are at most η different mappings ψ_u, as there are at most $\gamma \cdot m \cdot (k-1) \cdot \rho_{suf}$ many different suffixes y that can be produced from the same word w_2 by using at most $\gamma \cdot m$ suffix rewrite steps, and as there are only ι components of $\mathcal{M}$, there are (at least) two different c-incompressible words $u, U \in \Sigma_1^n$, a suffix y derivable from w_2, and an index i_0 of a component of $\mathcal{M}$ such that the following conditions are all satisfied simultaneously:

- $u = u_1\hat{u}$ and $U = U_1\hat{u}$, where $|u_1| = |U_1| = m \cdot \log_3^2(n)$,
- $w = \varphi_2(u) = w_1w_2$ and $W = \varphi_2(U) = W_1w_2$, where $|w_2| = \gamma \cdot m \cdot (k-1)$,
- the mapping ψ_u and ψ_U are identical,
- there is an accepting computation of $\mathcal{M}$ on input $uc\,\varphi_1(u)c\,\varphi_2(u)$ such that through the first m inner rewrites, the prefix u_1 of u is rewritten into the word u', through the suffix rewrites up to the m-th inner rewrite the suffix w_2 of $\varphi_2(u)$ is rewritten into the word y, and after this part of the computation component i_0 becomes active,
- there is an accepting computation of $\mathcal{M}$ on input $Uc\,\varphi_1(U)c\,\varphi_2(U)$ such that through the first m inner rewrites, the prefix U_1 of U is rewritten into the word u', through the suffix rewrites up to the m-th inner rewrite the suffix w_2 of $\varphi_2(U)$ is rewritten into the word y, and after this part of the computation component i_0 becomes active.

Now consider the input $z_{mix} := Uc\,\varphi_1(u)c\,\varphi_2(u)$. As $u \neq U$, and as the mapping $(\varphi_1, \varphi_2) : \Sigma_1^* \to (\Sigma_0^* \times \Sigma_0^*)$ is injective, we see that $z_{mix} \notin L_{cc}$ holds. Thus, there must not be an accepting computation of $\mathcal{M}$ on input z_{mix}.

There is an accepting computation of $\mathcal{M}$ on input $Uc\,\varphi_1(U)c\,\varphi_2(U)$ that begins with an initial segment $Uc\,\varphi_1(U)c\,\varphi_2(U) \vdash_{\mathcal{M}}^{c^*} u'\hat{u}c\,\varphi_1(U)cW_1y$. Also there is an accepting computation of $\mathcal{M}$ on input $uc\,\varphi_1(u)c\,\varphi_2(u)$ that begins with an initial segment $uc\,\varphi_1(u)c\,\varphi_2(u) \vdash_{\mathcal{M}}^{c^*}$

$u'\hat{u}c\,\varphi_1(u)cw_1y$. In both cases component i_0 is the next that becomes active. As the mappings ψ_u and ψ_U that are induced by the factors $\hat{u}c\,\varphi_1(u)cw_1$ and $\hat{u}c\,\varphi_1(U)cW_1$, respectively, coincide, it follows that in the former of these accepting computations we can replace the factor $\varphi_1(U)cW_1$ by $\varphi_1(u)cw_1$, which yields the following computation:

$$Uc\,\varphi_1(u)c\,\varphi_2(u) = U_1\hat{u}c\,\varphi_1(u)cw_1w_2 \vdash^{c^*}_{\mathcal{M}} u'\hat{u}c\,\varphi_1(u)cw_1y.$$

Also in this case component i_0 is the next that becomes active. Hence, with the word $uc\,\varphi_1(u)c\,\varphi_2(u)$, $\mathcal{M}$ will also accept the word z_{mix}. This, however, contradicts our assumption that $L_{=1}(\mathcal{M}) = L_{cc}$ holds. It follows that L_{cc} is not accepted by any locally deterministic CD-RRW-system that is working in mode $=1$. □

Together with Lemma 3.2 this yields the following separation results.

Corollary 3.11 [9] *For all types* $\mathsf{X} \in \{\mathsf{RR}, \mathsf{RRW}\}$,

$$\mathcal{L}_{=1}(\text{det-local-CD-X}) \subset \mathcal{L}_{=1}(\text{CD-X}) = \mathcal{L}(\text{nf-X}).$$

4 A Lower Bound for Deterministic Monotone nf-RRW-Automata

Each computation of a restarting automaton proceeds in cycles, where each cycle contains exactly one application of a rewrite operation. Thus, each cycle C contains a unique configuration $\alpha q\beta$ in which a rewrite instruction is applied. The number $|\beta|$ is called the *right distance* of C, denoted by $D_r(C)$. We say that a *sequence of cycles* $S = (C_1, C_2, \cdots, C_n)$ is *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$. Now a computation of a restarting automaton M is called *monotone* if the corresponding sequence of cycles is monotone, and a restarting automaton M is *monotone* if all its computations that start with an initial configuration are monotone. Here we apply the lower bound technique developed in the previous section to separate the deterministic monotone nf-RRW-automata from the deterministic monotone nf-RRWW-automata.

Definition 4.1 *As before let* $\Sigma_0 = \{a, b\}$ *and* $\Sigma_1 = \{a, b, c\}$, *and let* $\varphi : \Sigma_1^* \to \Sigma_0^*$ *be the morphism that is induced by* $a \mapsto aa$, $b \mapsto bb$, *and* $c \mapsto ab$. *Then* $\varphi : \Sigma_1^* \to \Sigma_0^*$ *is an encoding.*

Now let L_p *be the following language:*

$$L_p := \{\, wc(\varphi(w))^R \mid w \in \Sigma_1^*,\ |w| \geq 2,\ \textit{and } |w| \equiv 0 \textit{ mod } 2 \,\}.$$

Using auxiliary letters the language L_p is easily accepted by a deterministic, monotone, and nonforgetting restarting automaton.

Lemma 4.2 L_p *is accepted by a deterministic monotone* nf-RRWW-*automaton.*

Proof. Let M be the nonforgetting RRWW-automaton with input alphabet Σ_1 and tape alphabet $\Gamma := \Sigma_1 \cup \Delta$, where $\Delta := \{\, [x, y] \mid x, y \in \Sigma_1 \,\}$, that is given by the following sequence of meta-instructions:

(1)	$(q_0, ¢ \cdot \Delta^*, xy \to [x, y], \Sigma_0^* \cdot c \cdot \Sigma_1^* \cdot c \cdot \Sigma_0^* \cdot \$, q_0)$	for all $x, y \in \Sigma_1$,
(2)	$(q_0, ¢ \cdot \Delta^*, xy \to [x, y], (\Sigma_0^2)^* \cdot c \cdot \Sigma_0^* \cdot \$, q_1)$	for all $x, y \in \Sigma_1$,
(3)	$(q_1, ¢ \cdot \Delta^* \cdot \Sigma_0^*, x \cdot c \cdot \varphi(x)^R \to c, \Sigma_0^* \cdot \$, q_1)$	for all $x \in \Sigma_0$,
(4)	$(q_1, ¢ \cdot \Delta^*, [x, y] \cdot c \cdot (\varphi(xy))^R \to c, \Sigma_0^* \cdot \$, q_1)$	for all $x, y \in \Sigma_1$,
(5)	$(q_1, ¢ \cdot c \cdot \$, \mathsf{Accept})$.	

Given an input $z \in \Sigma_1^*$, M works as follows. If $|z| < 3$, then M rejects immediately, as neither meta-instruction (1) nor (2) is applicable to the corresponding initial tape contents $¢ \cdot z \cdot \$$. Thus, assume that $z = xyz_1$, where $x, y \in \Sigma_1$ and $z_1 \in \Sigma_1^+$. Again, if z_1 does not contain an occurrence of the symbol c, then again it follows that M will reject immediately. Hence, we can assume that $z_1 = z_2cz_3$, where $z_2 \in \Sigma_1^*$ and $z_3 \in \Sigma_0^*$. Then M rewrites its tape contents $¢z\$ = ¢xyz_2cz_3\$$ into $¢[x, y]z_2cz_3\$$, and it restarts in the restarting state q_1, if $|z_2|_c = 0$ and $|z_2| \equiv 0 \bmod 2$, and it restarts in the restarting state q_0, if $|z_2|_c \geq 1$. Continuing with the latter case, assume that $z_2 = z_4fg\beta$, where $z_4 \in (\Sigma_1^2)^*$, $f, g \in \Sigma_1$ such that $f = c$ or $g = c$, and $\beta \in \Sigma_0^*$. Then, after a finite number of cycles the prefix of z_4fg is encoded by a string $\alpha \in \Delta^+$, and then M switches to the restarting state q_1, provided $|\beta| \equiv 0 \bmod 2$ holds. If $|\beta| \equiv 1 \bmod 2$ holds, or if $f = c$ and $g\beta = \varepsilon$, which means that $fg\beta cz_3 = ccz_3$ is rewritten into $[c, c]z_3$, then M cannot complete the current cycle and rejects. Otherwise, starting from the restarting state q_1, M compares the prefix $[x, y]\alpha\beta$ to the suffix z_3, and it accepts if and only if $z_3 = (\varphi(xyz_4fg\beta))^R$ holds. Thus, we see that $L(M) = L_p$.

Finally, it is easily seen from the definition of M that M is deterministic, and it follows from the above description of the way in which M works that it is also monotone. Thus, M is a deterministic monotone nf-RRWW-automaton for the language L_p. □

Now the main part of the work in this section consists in establishing the following lower bound result.

Theorem 4.3 *The language L_p is not accepted by any deterministic monotone nonforgetting* RRW-*automaton.*

Proof. Assume that $M = (Q, \Sigma_1, \Sigma_1, ¢, \$, q_0, k, \delta)$ is a deterministic monotone nonforgetting RRW-automaton such that $L(M) = L_p$. We will now derive a contradiction through a sequence of claims.

As M is monotone, we see that each (accepting) computation of M consists of two parts: The first part consists of a number of cycles in which inner rewrite steps are performed, and the second part consists only of cycles in which suffix rewrite steps are performed. Concerning the first part we have the following result.

Claim 1. For each $m \in \mathbb{N}_+$, there exists an integer $n_0 \in \mathbb{N}_+$ such that the following statement holds for all even $n \geq n_0$ and all c-incompressible words $u \in \Sigma_1^n$: If M starts from an initial configuration corresponding to the input $z = uc(\varphi(u))^R$, then in the resulting computation of M the first m inner rewrites are executed within the prefix of u of length $m \cdot \log_3^2(n)$.

Proof. This is proved in exactly the same way as Proposition 3.6. □

As before we will denote the constant n_0 that corresponds to a given value of m by $n_0(m)$. The next claim shows that the first part of the computation of M on input $uc(\varphi(u))^R$ consists of more than m cycles.

Claim 2. Let $m \geq 1$, and let $u \in \Sigma_1^n$ be a c-incompressible word of even length $n \geq n_0(m)$. Then the accepting computation of M on input $uc(\varphi(u))^R$ contains more than m applications of inner rewrite steps.

Proof. Assume that the accepting computation of M on input $uc(\varphi(u))^R$ consists of only $r \leq m$ cycles in which inner rewrite steps are executed followed by a sequence of cycles that only involve applications of suffix rewrite steps. Then it follows from Claim 1 above that in the first part of the computation considered, $uc(\varphi(u))^R$ is rewritten into a word of the form $u'u_2c(\varphi(u))^R$, where $u = u_1u_2$, $|u_1| = r \cdot \log_3^2 n$, and u' is obtained from u_1 through the execution of the above-mentioned r inner rewrite steps. In the second part of the computation considered a sequence of cycles is executed that only involves suffix rewrite steps, and that leads to a restarting configuration from which M accepts. By using the pumping argument from the proof of Proposition 3.7 this can be shown to yield a contradiction. Thus, in the computation considered M must execute more than m inner rewrite steps. $\square$

After performing an inner rewrite step inside the prefix of length $m \cdot \log_3^2 n$ of a c-incompressible word of even length $n \geq n_0(m)$, M scans the corresponding suffix $u_2 c(\varphi(u))^R$ of the tape contents, and we can assume that at the right delimiter it either accepts, executes a restart operation $\mathsf{Restart}(q)$ for some $q \in Q$, or rejects. Thus, with the word $u_2 c(\varphi(u))^R$ we can associate a mapping

$$\psi_u : Q \to \{\mathsf{Accept}, \mathsf{Reject}\} \cup \{\, \mathsf{Restart}(q) \mid q \in Q \,\}.$$

Obviously, there are only $\eta := |Q|^{|Q|+2}$ such mappings. Accordingly, we choose a constant $m \in \mathbb{N}_+$ such that $3^{m-2} > \eta \cdot |Q|$, and let $n \geq n_0(m)$ be an even integer.

Claim 3. There exists a word $u' \in \Sigma_1^*$ that has the following properties:

- $n - m \cdot k \leq |u'| \leq n - m$;
- there are at least $(1 - \frac{1}{2 \cdot 3^c}) \cdot 3^{m-1}$ different c-incompressible words u of length n such that, for each of these words, M has an accepting computation on input $uc(\varphi(u))^R$ that rewrites the prefix u into the word u' through the first m inner rewrites of this computation;
- u' has the form $u' = u_1' \hat{u}$, where $|\hat{u}| = n - m \cdot \log_3^2(n)$, and each of the c-incompressible words u above has the same suffix $\hat{u}$.

Proof. This is proved in exactly the same way as Proposition 3.9. $\square$

There are at least $(1 - \frac{1}{2 \cdot 3^c}) \cdot 3^{m-1}$ different c-irreducible words $u \in \Sigma_1^n$ with the above properties. As

$$(1 - \frac{1}{2 \cdot 3^c}) \cdot 3^{m-1} \geq \frac{1}{2} \cdot 3^{m-1} > 3^{m-2} > \eta \cdot |Q|,$$

as there are at most η different mappings ψ_u, and as there are at most $|Q|$ different restarting states of M, there are (at least) two different c-incompressible words $u, U \in \Sigma_1^n$, and a restarting state $q \in Q$ such that the following conditions are all satisfied simultaneously:

- $u = u_1 \hat{u}$ and $U = U_1 \hat{u}$, where $|u_1| = |U_1| = m \cdot \log_3^2(n)$,
- the mapping ψ_u and ψ_U are identical,
- through the first m inner rewrite steps in the accepting computation of M on input $uc(\varphi(u))^R$, the prefix u_1 of u is rewritten into the word u', and after these m cycles M restarts in the restarting state q,
- through the first m inner rewrite steps in the accepting computation of M on input $Uc(\varphi(U))^R$, the prefix U_1 of U is rewritten into the word u', and after these m cycles M restarts in the restarting state q.

Now consider the input $z_{mix} := Uc(\varphi(u))^R$. As $u \neq U$, and as the mapping $\varphi : \Sigma_1^* \to \Sigma_0^*$ is injective, we see that $z_{mix} \notin L_p$ holds. Thus, there must not be an accepting computation of M on input z_{mix}.

There is an accepting computation of M on input $Uc(\varphi(U))^R$ that begins with an initial segment $Uc(\varphi(U))^R \vdash_M^{c^*} u'\hat{u}c(\varphi(U))^R$. Also there is an accepting computation of M on input $uc(\varphi(u))^R$ that begins with an initial segment $uc(\varphi(u))^R \vdash_M^{c^*} u'\hat{u}c(\varphi(u))^R$. In both cases q is the restarting state with which M starts the next cycle. As the mappings ψ_u and ψ_U that are induced by the factors $\hat{u}c(\varphi(u))^R$ and $\hat{u}c(\varphi(U))^R$, respectively, coincide, it follows that in the former of these accepting computations we can replace the factor $(\varphi(U))^R$ by $(\varphi(u))^R$, which yields the following computation:

$$Uc(\varphi(u))^R = U_1\hat{u}c(\varphi(u))^R \vdash_M^{c^*} u'\hat{u}c(\varphi(u))^R.$$

Also in this case q is the restarting state with which M starts the next cycle. Hence, with the word $uc(\varphi(u))^R$, M will also accept the word z_{mix}. This, however, contradicts our assumption that $L(M) = L_p$ holds. It follows that L_p is not accepted by any deterministic monotone nonforgetting RRW-automaton. □

Together with Lemma 4.2 this yields the following separation results.

Corollary 4.4 [10] $\mathcal{L}(\text{det-mon-nf-RRW}) \subset \mathcal{L}(\text{det-mon-nf-RRWW})$.

5 A Lower Bound for Deterministic Monotone nf-RR-Automata

In the proof of Lemma 4.2 we have presented a deterministic monotone nf-RRWW-automaton M with tape alphabet $\Gamma = \Sigma_1 \cup \Delta$ for the language L_p. Here we consider the characteristic language $L_p^\Gamma := L_\mathrm{C}(M)$ of M. We claim that this language is accepted by a deterministic monotone nf-RRW-automaton, but that it is not accepted by any deterministic monotone nf-RR-automaton.

Lemma 5.1 *L_p^Γ is accepted by a deterministic monotone* nf-RRW-*automaton.*

Proof. Let M_C be the deterministic nf-RRW-automaton that is obtained from the aforementioned nf-RRWW-automaton M by declaring all tape symbols to input symbols. Then it is immediate that $L(M_\mathrm{C}) = L_\mathrm{C}(M) = L_p^\Gamma$ holds. It remains to show that M_C is monotone, that is, for each restarting configuration of M_C, the computation of M_C starting from that configuration is monotone. But this follows easily from the definition of the nf-RRWW-automaton M, as its meta-instructions ensure that a cycle can be completed only if the current tape content is of the form $¢Uxcycz\$$ or $¢Uycz\$$, where $U \in \Delta^*$, $x \in \Sigma_1^*$, and $y, z \in \Sigma_0^*$. In the former case the current rewrite replaces the prefix of x of length 2 by a symbol from Δ, and in the latter case it deletes some symbols surrounding the c-symbol. □

It remains to establish the announced lower bound.

Theorem 5.2 *The language L_p^Γ is not accepted by any deterministic monotone nonforgetting* RR-*automaton.*

Proof. Assume to the contrary that $M' = (Q, \Gamma, \Gamma, ¢, \$, q_0, k, \delta)$ is a deterministic monotone nf-RR-automaton such that $L(M') = L_p^\Gamma$ holds. Define a deterministic nf-RR-automaton M_r as a restricted variant of M' by taking $M_r := (Q, \Sigma_1, \Sigma_1, ¢, \$, q_0, k, \delta_r)$. Here δ_r is simply the restriction of δ to words that only contain letters from the subalphabet Σ_1 of Γ, that is, $\delta_r(q,u) := \delta(q,u)$ for all $q \in Q$ and all words $u \in ¢\cdot\Sigma_1^{k-1} \cup \Sigma_1^k \cup \Sigma_1^{\leq k-1}\cdot\$ \cup ¢\cdot\Sigma_1^{\leq k-2}\cdot\$$. As M' is an RR-automaton, all its rewrite operations are just deletions. Thus, for each input $w \in \Sigma_1^*$, the computation of M_r on input w is identical to the computation of M' on input w. Hence, we see that M_r is monotone, and that $L(M_r) = L(M') \cap \Sigma_1^* = L_p^\Gamma \cap \Sigma_1^* = L_{\mathrm{C}}(M) \cap \Sigma_1^* = L_p$. This, however, contradicts Theorem 4.3, which states that the language L_p is not accepted by any deterministic monotone nf-RRW-automaton. This shows that the language L_p^Γ is indeed not accepted by any deterministic monotone nf-RR-automaton. □

Together with Lemma 5.1 this yields the following separation results.

Corollary 5.3 [10] $\mathcal{L}(\text{det-mon-nf-RR}) \subset \mathcal{L}(\text{det-mon-nf-RRW})$.

6 A Lower Bound for Monotone nf-RW-Automata

Next we apply our lower bound technique to separate the monotone nf-RW-automata from the monotone nf-RWW-automata by showing that the context-free language L_{pal} (see below) is not accepted by any monotone nf-RW-automaton.

Definition 6.1 *The language L_{pal} is defined as follows, where $\Sigma_0 = \{a, b\}$:*

$$L_{pal} := \{ ww^R \mid w \in \Sigma_0^* \},$$

that is, L_{pal} is the language of palindromes of even length over Σ_0.

Obviously, L_{pal} is context-free, and therewith it is accepted by a monotone nf-RWW-automaton, since

$$\mathsf{CFL} = \mathcal{L}(\text{mon-R(R)WW}) = \mathcal{L}(\text{mon-nf-R(R)WW})$$

according to [5] and [9, 10]. On the other hand, the following lower bound result shows that monotone nf-RW-automata are less expressive.

Theorem 6.2 *The language L_{pal} is not accepted by any monotone nonforgetting* RW-*automaton.*

Proof. Assume that $M = (Q, \Sigma_0, \Sigma_0, \text{¢}, \$, q_0, k, \delta)$ is a monotone nonforgetting RW-automaton such that $L(M) = L_{pal}$. We will now derive a contradiction through a sequence of claims.

As M is monotone, we see that each (accepting) computation of M consists of two parts: The first part consists of a number of cycles in which inner rewrite steps are performed, and the second part consists only of cycles in which suffix rewrite steps are performed. Using the arguments from the proof of Proposition 3.7 it can easily be shown that the second part of any accepting computation consists of a sequence of cycles the length of which is bounded by a constant depending on M. Concerning the first part we have the following result.

Claim 1. For each $m \in \mathbb{N}_+$, there exists an integer $n_0 \in \mathbb{N}_+$ such that the following statement holds for all $n \geq n_0$ and all c-incompressible words $u \in \Sigma_0^n$: If M starts from an initial configuration corresponding to the input $z = uu^R$, then in an accepting computation of M, M executes more than m inner rewrite steps, and the first m of these rewrites are executed within the prefix of u of length $m \cdot \log_2^2(n)$.

Proof. For each $u \in \Sigma_0^*$, an accepting computation of M on input uu^R consists of a sequence of cycles in which inner rewrites are performed, followed by a sequence of bounded length in which suffix rewrites are performed. If the first sequence could be bounded in length by a fixed constant m, then for words u of sufficient length we could use pumping to fool M into accepting incorrect words. Thus, for each $m \in \mathbb{N}_+$, there exists an integer $n_0^{(0)}$ such that each accepting computation of M on input uu^R ($|u| \geq n_0^{(0)}$) begins with more than m inner rewrites.

It remains to prove that the first m inner rewrites are executed within the prefix of length $m \cdot \log_2^2(n)$ for each c-incompressible word of sufficient length. Here we cannot simply refer to the proof of Proposition 3.6, as M is nondeterministic. In fact, we must use the fact that M is monotone.

With each (restarting) state q of M, we can associate a deterministic finite-state acceptor A_q such that, given a word $w \in \Sigma_0^*$ as input, A_q will determine the shortest prefix of the form $w_1 u_j$ of w such that $¢w_1 \in L(E_j)$ for a meta-instruction of M of the form $I_j = (q, E_j, u_j \to v_j, q')$. By $\mathcal{A}$ we denote the disjoint union of these deterministic finite-state acceptors.

Now let $m \in \mathbb{N}_+$ be given. For m we obtain a constant $n_0^{(1)} \in \mathbb{N}_+$ from Proposition 3.4 (or rather its analog for Σ_0), and for the deterministic finite-state acceptor $\mathcal{A}$, we get a constant $n_0^{(2)} \in \mathbb{N}_+$ from Proposision 3.5. We now take n_0 as $n_0 := \max\{n_0^{(0)}, n_0^{(1)}, n_0^{(2)}\}$ (cf. the proof of Proposition 3.6).

Now let $n \geq n_0$, and let $u \in \Sigma_0^n$ be a c-incompressible word. We can factor u as $u = u_1 u_2 \cdots u_m \hat{u}$ such that $|u_i| = \log_2^2(n)$, $1 \leq i \leq m$. Then we know from Proposition 3.4 that the suffix $u^{(i)} = u_{i+1} \cdots u_m \hat{u}$ of u is random for each $1 \leq i \leq m$.

Consider a computation of M starting from the initial configuration $q_0 ¢ uu^R \$$. Assume that in the first cycle $(q_0, uu^R) \vdash_M^c (q_1, x)$, M executes an inner rewrite, but not inside the prefix u_1. Now consider the second cycle that begins with the restarting configuration $q_1 ¢ x \$ = q_1 ¢ u_1 x' \$$. If in the second cycle M also executes an inner rewrite, then $\mathcal{A}$ must reach a corresponding state starting from the state that corresponds to the initial state of A_{q_1}. By Proposition 3.5 this happens already inside the prefix u_1. Thus, in the second cycle M could execute an inner rewrite inside the prefix u_1, which implies that the sequence consisting of these two cycles is not monotone. This contradicts our assumption that M is monotone. Hence, the inner rewrite in the first cycle is executed inside the prefix u_1. Arguing inductively it follows that, for each $i = 1, \ldots, m$, the i-th inner rewrite is executed inside the prefix $u_1 \ldots u_i$. □

Claim 2. Let $m \geq 1$, and let $n \geq n_0$. Then there exists a word $u' \in \Sigma_0^*$ that has the following properties:

- $n - m \cdot k \leq |u'| \leq n - m$;
- there are at least $(1 - \frac{1}{2^{c+1}}) \cdot 2^{m-1}$ different c-incompressible words u of length n such that, for each of these words, M has an accepting computation on input uu^R that rewrites the prefix u into the word u' through the first m inner rewrites of this computation;
- u' has the form $u' = u_1' \hat{u}$, where $|\hat{u}| = n - m \cdot \log_2^2(n)$, and each of the c-incompressible words u above has the same suffix $\hat{u}$.

Proof. There are at least $(1 - \frac{1}{2^{c+1}}) \cdot 2^n$ words of length n in Σ_0^* that are c-incompressible. On the other hand, by executing m inner rewrite steps on the prefix of length $m \cdot \log_2^2(n)$ of a c-incompressible word $u \in \Sigma_0^n$, u is rewritten into a word u' such that $n - m \cdot k \leq |u'| \leq n - m$. As there are only $\sum_{i=n-m\cdot k}^{n-m} 2^i \leq \sum_{i=0}^{n-m} 2^i \leq 2^{n-m+1}$ words satisfying this length restriction, we see that there exists a word u' such that at least $(1 - \frac{1}{2^{c+1}}) \cdot \frac{2^n}{2^{n-m+1}} = (1 - \frac{1}{2^{c+1}}) \cdot 2^{m-1}$ different c-incompressible words u of length n are reduced to this particular word u'. Further,

u' has the form $u' = u'_1\hat{u}$, where $\hat{u}$ is the common suffix of length $n - m \cdot \log_2^2(n)$ of all the c-incompressible words u that are rewritten into this word u'. □

We now choose a constant $m \in \mathbb{N}_+$ such that $2^{m-2} > 2 \cdot \iota$, where ι be the number of (restarting) states of M, and take $n \geq n_0$ such that $n \geq m \cdot \log_2^2(n)$. Then by Claim 2 there exists a word $u' \in \Sigma_0^*$ of length $n - m \cdot k \leq |u'| \leq n - m$ such that there are at least $(1 - \frac{1}{2^{c+1}}) \cdot 2^{m-1} > 2 \cdot \iota$ different c-incompressible words u of length n such that, for each of these words u, M has an accepting computation on input uu^R that rewrites the prefix u into the word u' through the first m inner rewrites of this computation, and u' has the form $u' = u'_1\hat{u}$, where $|\hat{u}| = n - m \cdot \log_2^2(n)$, and each of the c-incompressible words u above has the same suffix $\hat{u}$. Obviously, the word $z = uu^R$ belongs to the language L_{pal}, and accordingly M has accepting computations that start from an initial configuration corresponding to this input.

We consider the initial phase of such a computation that ends with the m-th inner rewrite. By Claim 1 these m inner rewrites are all executed within the prefix u_1 of u, transforming this prefix into the word u'_1. As $(1 - \frac{1}{2^{c+1}}) \cdot 2^{m-1} > 2 \cdot \iota$, there are (at least) two different c-incompressible words $u, U \in \Sigma_0^n$, and a (restarting) state p of M such that the following conditions are all satisfied simultaneously:

- $u = u_1\hat{u}$ and $U = U_1\hat{u}$, where $|u_1| = |U_1| = m \cdot \log_2^2(n)$,
- there is an accepting computation of M on input uu^R such that through the first m inner rewrites, the prefix u_1 of u is rewritten into the word u'_1, and after this part of the computation M restarts in state p,
- there is an accepting computation of $\mathcal{M}$ on input UU^R such that through the first m inner rewrites, the prefix U_1 of U is rewritten into the word u'_1, and after this part of the computation M restarts in state p.

Now consider the input $z_{mix} := Uu^R$. As $u \neq U$, we see that $z_{mix} \notin L_{pal}$ holds. Thus, there must not be an accepting computation of M on input z_{mix}.

There is an accepting computation of M on input UU^R that begins with an initial segment $UU^R \vdash_M^{c^*} u'_1\hat{u}U^R$. Also there is an accepting computation of M on input uu^R that begins with an initial segment $uu^R \vdash_M^{c^*} u'_1\hat{u}u^R$. In both cases M restarts in the same state p. It follows that in the former of these accepting computations we can replace the factor U^R by u^R, which yields the computation:

$$Uu^R = U_1\hat{u}u^R \vdash_M^{c^*} u'_1\hat{u}u^R.$$

Also in this case M restarts in state p. Hence, with the word uu^R, M will also accept the word z_{mix}. This, however, contradicts our assumption that $L_{=1}(M) = L_{pal}$ holds. It follows that L_{pal} is not accepted by any monotone nonforgetting RW-automaton. □

This yields the following separation result.

Corollary 6.3 [10] $\mathcal{L}(\text{mon-nf-RW}) \subset \mathcal{L}(\text{mon-nf-RWW}) = \text{CFL}$.

References

[1] G. Buntrock. *Wachsende kontext-sensitive Sprachen.* Habilitationsschrift, Fakultät für Mathematik und Informatik, Universität Würzburg, 1996.

[2] E. Csuhaj-Varju, J. Dassow, J. Kelemen, and G. Păun. *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[3] J. Dassow, G. Păun, and G. Rozenberg. Grammar systems. In: G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Vol. 2, Springer, Berlin, 1997, 155–213.

[4] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In: H. Reichel (ed.), *FCT 1995, Proc.*, *Lect. Notes Comput. Sci. 965*, Springer, Berlin, 1995, 283–292.

[5] 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.

[6] T. Jurdziński and F. Otto. Restarting automata with restricted utilization of auxiliary symbols. *Theoretical Computer Science* 363 (2006) 162–181.

[7] 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.

[8] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, 1997.

[9] H. Messerschmidt. *CD-Systems of Restarting Automata.* Doctoral dissertation, Fachbereich Elektrotechnik/Informatik, Universität Kassel, 2008.

[10] H. Messerschmidt and F. Otto. On nonforgetting restarting automata that are deterministic and/or monotone. In: D. Grigoriev, J. Harrison, and E.A. Hirsch (eds.), *CSR 2006, Proc.*, *Lect. Notes Comput. Sci. 3967*, Springer, Berlin, 2006, 247–258.

[11] H. Messerschmidt and F. Otto. Cooperating distributed systems of restarting automata. *Intern. J. Found. Comput. Sci.* 18 (2007) 1333–1342.

[12] H. Messerschmidt and F. Otto. Strictly deterministic CD-systems of restarting automata. In: E. Csuhaj-Varjú and Z. Ésik (eds.), *FCT 2007, Proc.*, *Lect. Notes Comput. Sci. 4639*, Springer, Berlin, 2007, 424–434.

[13] 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.

[14] K. Oliva, P. Květoň, and R. Ondruška. The computational complexity of rule-based part-of-speech tagging. In: V. Matousek and P. Mautner (eds.), *TSD 2003, Proc.*, *Lect. Notes Comput. Sci. 2807*, Springer, Berlin, 2003, 82–89.

[15] F. Otto. Restarting automata. In: Z. Ésik, C. Martin-Vide, and V. Mitrana (eds.), *Recent Advances in Formal Languages and Applications*, Studies in Computational Intelligence Vol. 25, Springer, Berlin, 2006, 269–303.

[16] 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.