Strictly Deterministic CD-Systems of Restarting Automata

H. Messerschmidt and F. Otto

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

Abstract. A CD-system of restarting automata is called *strictly deterministic* if all its component systems are deterministic, and if there is a unique successor system for each component. Here we show that the strictly deterministic CD-systems of restarting automata are strictly more powerful than the corresponding deterministic types of restarting automata, but that they are strictly less powerful than the corresponding deterministic types of nonforgetting restarting automata. In fact, we present an infinite hierarchy of language classes based on the number of components of strictly deterministic CD-systems of restarting automata.

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 [4]. This technique consists in a stepwise 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 [9,12].

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 is moved from left to right 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 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.

Many restricted variants of restarting automata have been studied and put into correspondence to more classical classes of formal languages. For a recent survey see [10] or [11]. Also further extensions of the model have been considered. In particular, in [8] Messerschmidt and Stamer introduced the *nonforgetting restarting automaton*, which, when executing a restart operation, simply changes

E. Csuhaj-Varjú and Z. Ésik (Eds.): FCT 2007, LNCS 4639, pp. 424–434, 2007.

its internal state as with any other operation, instead of resetting it to the initial state. Further, in [7] the authors introduced cooperating distributed systems (CD-systens) of restarting automata and proved that CD-systems of restarting automata working in mode = 1 correspond to nonforgetting restarting automata.

Here we concentrate on CD-systems of restarting automata that are deterministic. It is known that deterministic restarting automata with auxiliary symbols accept exactly the Church-Rosser languages (see, e.g., [10,11]), while nonforgetting deterministic restarting automata are strictly more powerful [6]. However, for CD-systems of restarting automata the notion of determinism can be defined in various different ways. A CD-system $\mathcal{M} := ((M_i, \sigma_i)_{i \in I}, I_0)$ of restarting automata could be called *deterministic* if within each computation of the system $\mathcal{M}$, each configuration has at most a single successor configuration. This is a global view on determinism. On the other hand, we could follow the way determinism is used in CD-grammar systems (see, e.g., [2,3]) and call $\mathcal{M}$ already *deterministic* if all component automata M_i $(i \in I)$ are deterministic. This is a local view on determinism. Here we study a third option, called *strict determinism*, where we require not only that all component automata M_i $(i \in I)$ are deterministic, but also that the successor set σ_i is a singleton for each $i \in I$. This is again a global, but more restricted, view.

We will see that, in analogy to the situation for nondeterministic restarting automata, the globally deterministic CD-systems of restarting automata, when working in mode = 1, correspond exactly to nonforgetting deterministic restarting automata. Further, the expressive power of strictly deterministic CD-systems of restarting automata lies strictly in between that of deterministic restarting automata and that of nonforgetting deterministic restarting automata. In fact, based on the number of component systems of strictly deterministic CD-systems, we will obtain a proper infinite hierarchy of language classes.

This paper is structured as follows. In Section 2 we introduce nonforgetting restarting automata and CD-systems of restarting automata. Then, in Section 3, we define the various types of deterministic CD-systems of restarting automata formally and establish the announced relationship between nonforgetting deterministic restarting automata and globally deterministic CD-systems of restarting automata. In Section 4 we compare the expressive power of strictly deterministic CD-systems to that of globally deterministic CD-systems and to that of deterministic restarting automata. Also we present the announced hierarchy on the number of component systems. The paper concludes with a short discussion pointing out some open problems for future work.

2 Definitions

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 $¢, \$ \notin \Gamma$ 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), where $q' \in Q$, which cause M to shift the read/write window one position to the right and to enter state q'.
- *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 Restart, which cause 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.
- *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 a finite set of 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*.

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 $|u| > |v|$. 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 $q_0 ¢ w_1 v w_2 \$$. This computation is called a *cycle* of M. It is expressed as $w \vdash_M^c w_1 v w_2$. 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 which, starting with the initial configuration $q_0 ¢ w \$$, consists of a finite sequence of cycles that is followed by an application of 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, 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, $\Gamma = \Sigma$), -ε means that no auxiliary symbols are available and that each rewrite step is simply a deletion (that is, if the rewrite operation $u \to v$ occurs in a meta-instruction of M, then v is obtained from u by deleting some symbols).

A *cooperating distributed system of* RRWW-*automata*, 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 x 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 $q_0^{(i_0)} ¢ x \$$, 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 machine M_{i_l} accepts. Should at some stage the chosen machine M_{i_l} 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 this machine not be able to execute a 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 $x \in \Sigma^*$ 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 following simple example shows that CD-R-systems have much more expressive power than R-automata. As R-automata restart immediately after executing a rewrite operation, rewriting meta-instructions for them are of the form

$(E, u \to v)$, where E is a regular language, and $u \to v$ is a rewrite step of M (see, e.g., [11]).

Example 1. Let $\mathcal{M} := ((M_1, \{2\}), (M_2, \{1\}), \{1\})$ be a CD-R-system, where M_1 is described by the following three meta-instructions:

$$(¢ \cdot \{a, b\}^* \cdot c \cdot \# \cdot \{a, b\}^*, c \cdot \$ \to \$),\ c \in \{a, b\},\ \text{and}\ (¢ \cdot \# \cdot \$, \mathsf{Accept}),$$

and M_2 is given through the meta-instructions:

$$(¢ \cdot \{a, b\}^*, c \cdot \# \to \#),\ c \in \{a, b\}.$$

In mode =1 the machines M_1 and M_2 alternate, with M_1 beginning the computation. Starting with a word of the form $u\#v$, where $u, v \in \{a, b\}^+$, M_1 always deletes the last letter of the second factor, provided it coincides with the last letter of the first factor, and M_2 simply deletes the last letter of the first factor. It follows that $L_{=1}(\mathcal{M})$ coincides with the language $L_{\text{copy}} := \{\, w\#w \mid w \in \{a, b\}^* \,\}$. On the other hand, it is easily seen that L_{copy} is not accepted by any R-automaton.

The *nonforgetting restarting automaton* 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 $q_2¢y\$$.

3 Various Notions of Determinism

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 successor system is chosen nondeterministically from among all systems 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 and if $|\sigma_i| = 1$ for each $i \in I$. Observe that the CD-R-system of Example 1 is strictly deterministic.

However, the restriction of having at most a single possible successor for each component system 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 $\delta_i(q, u) = (\mathsf{Restart}, j)$, where $j \in \sigma_i$, then the component M_j takes over. In this way it is guaranteed that all computations of a globally deterministic CD-system are

deterministic. However, for a component system M_i there can still be several possible successor systems. This is reminiscent of the way in which nonforgetting restarting automata work.

We use 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:

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

Concerning the globally deterministic CD-systems, we have the following results, which correspond to the results for nondeterministic CD-systems established in [7].

Theorem 1. *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 $L_{=1}(\mathcal{M}) = L(M)$ holds.*

For the converse we even have the following stronger result.

Theorem 2. *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 $L(M) = L_{=j}(\mathcal{M})$ holds.*

Thus, we see that globally deterministic CD-systems of restarting automata working in mode $= 1$ are just as powerful as deterministic nonforgetting restarting automata. It remains to study CD-systems that work in mode t.

Theorem 3. *Let $\mathsf{X} \in \{\mathsf{RR}, \mathsf{RRW}, \mathsf{RRWW}\}$, and let $\mathcal{M}$ be a globally deterministic CD-X-system. Then there exists a nonforgetting deterministic X-automaton M such that $L(M) = L_{\mathsf{t}}(\mathcal{M})$ holds.*

It is not clear whether the latter result extends to CD-systems of R(W)(W)-automata. The problem with these types of restarting automata stems from the fact that within a cycle such an automaton will in general not see the complete tape content.

4 Strictly Deterministic CD-Systems

Here we study the expressive power of strictly deterministic CD-systems of restarting automata. As seen in Example 1 the copy language L_{copy} is accepted by a strictly deterministic CD-R-system with two components. This language is not growing context-sensitive [1,5]. As deterministic RRWW-automata only accept Church-Rosser languages, which are a proper subclass of the growing context-sensitive languages, this yields the following separation result.

Proposition 1. *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$,

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

Let $L_{\text{copy}^m} := \{\, w(\#w)^{m-1} \mid w \in \Sigma_0^+ \,\}$ be the m-fold copy language. Analogously to Example 1 it can be shown that this language is accepted by a strictly deterministic CD-R-system with m components that works in mode $= 1$. The next example deals with a generalization of these languages.

Example 2. Let $L_{\text{copy}^*} := \{\, w(\#w)^n \mid w \in (\Sigma_0^2)^+,\ n \geq 1 \,\}$ be the iterated copy language, where $\Sigma_0 := \{a, b\}$. This language is accepted by a strictly deterministic CD-RWW-system $\mathcal{M} = ((M_1, \{2\}), (M_2, \{1\}), \{1\})$ with input alphabet $\Sigma := \Sigma_0 \cup \{\#\}$ and tape alphabet $\Gamma := \Sigma \cup \Gamma_0$, where $\Gamma_0 := \{A_{a,a}, A_{a,b}, A_{b,a}, A_{b,b}\}$. The RWW-automata M_1 and M_2 are given through the following meta-instructions, where $c, d, e, f \in \Sigma_0$:

$$\begin{aligned}
M_1 :\quad & (\mathbb{¢} \cdot (\Sigma_0^2)^* \cdot cd \cdot \# \cdot (\Sigma_0^2)^*, cd \cdot \# \to A_{c,d} \cdot \#),\\
& (\mathbb{¢} \cdot (\Sigma_0^2)^* \cdot cd \cdot \# \cdot (\Sigma_0^2)^*, cd \cdot \$ \to A_{c,d} \cdot \$),\\
& (\mathbb{¢} \cdot (\Sigma_0^2)^* \cdot cd \cdot \# \cdot (\Sigma_0^2)^*, cdA_{e,f} \to A_{c,d}A_{e,f}),\\
& (\mathbb{¢} \cdot \Gamma_0^* \cdot A_{c,d} \cdot \# \cdot (\Sigma_0^2)^*, cd \cdot \# \to A_{c,d} \cdot \#),\\
& (\mathbb{¢} \cdot \Gamma_0^* \cdot A_{c,d} \cdot \# \cdot (\Sigma_0^2)^*, cd \cdot \$ \to A_{c,d} \cdot \$),\\
& (\mathbb{¢} \cdot \Gamma_0^* \cdot A_{c,d} \cdot \# \cdot (\Sigma_0^2)^*, cdA_{e,f} \to A_{c,d}A_{e,f}),\\
& (\mathbb{¢} \cdot \Gamma_0^+ \cdot \$, \mathsf{Accept}),\\
M_2 :\quad & (\mathbb{¢} \cdot (\Sigma_0^2)^+, cd \cdot \# \to \#),\ \ (\mathbb{¢}, cd \cdot \# \to \varepsilon),\\
& (\mathbb{¢} \cdot \Gamma_0^+, A_{c,d} \cdot \# \to \#),\quad (\mathbb{¢}, A_{c,d} \cdot \# \to \varepsilon).
\end{aligned}$$

In mode $= 1$, the two components M_1 and M_2 are used alternatingly, with M_1 starting the computation. Let $x := w_1\#w_2\#\ldots\#w_m$ be the given input, where $w_1, w_2, \ldots, w_m \in (\Sigma_0^2)^+$ and $m \geq 2$. First w_1 is compared to w_2 by processing these strings from right to left, two letters in each round. During this process w_1 is erased, while w_2 is encoded using the letters from Γ_0. Next the encoded version of w_2 is used to compare w_2 to w_3, again from right to left. This time the encoded version of w_2 is erased, while w_3 is encoded. This continues until all syllables w_i have been considered. It follows that $L_{=1}(\mathcal{M}) = L_{\text{copy}^*}$ holds.

For accepting the language L_{copy^*} without using auxiliary symbols we have a CD-system of restarting automata that is globally deterministic.

Lemma 1. *The language* L_{copy^*} *is accepted by a globally deterministic* CD-R*-system working in mode* $= 1$.

Proof. Let $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ be the CD-R-system that is specified by $I := \{0, 1, 2, 3, 4, 5, 6\}$, $I_0 := \{0\}$, $\sigma(0) := \{5\}$, $\sigma(1) := \{2, 6\}$, $\sigma(2) := \{1, 6\}$, $\sigma(3) := \{4, 5\}$, $\sigma(4) := \{3, 5\}$, $\sigma(5) := \{1\}$, $\sigma(6) := \{3\}$, and M_0 to M_6 are given through the following meta-instructions, where $c, d \in \Sigma_0$:

$$\begin{aligned}
M_0 :\ & (\mathbb{¢} \cdot ((\Sigma_0^2)^+ \cdot \Sigma_0 \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^+ \cdot \Sigma_0, c \cdot \$ \to \$, \mathsf{Restart}(5)),\\
& (\mathbb{¢} \cdot (u \cdot \#)^+ \cdot u \cdot \$, \mathsf{Accept}) \text{ for all } u \in \Sigma_0^2,
\end{aligned}$$

$$
\begin{aligned}
M_1 &: (\mathbb{¢} \cdot ((\Sigma_0^2)^+ \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^+, c \cdot \$ \to \$, \mathsf{Restart}(6)),\\
&\quad (\mathbb{¢} \cdot ((\Sigma_0^2)^+ \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^+ \cdot c, d \cdot \# \to \#, \mathsf{Restart}(2)),\\
M_2 &: (\mathbb{¢} \cdot ((\Sigma_0^2)^+ \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^+, c \cdot \$ \to \$, \mathsf{Restart}(6)),\\
&\quad (\mathbb{¢} \cdot ((\Sigma_0^2)^+ \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^+ \cdot c, d \cdot \# \to \#, \mathsf{Restart}(1)),\\
M_3 &: (\mathbb{¢} \cdot ((\Sigma_0^2)^+ \cdot \Sigma_0 \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^+ \cdot \Sigma_0, c \cdot \$ \to \$, \mathsf{Restart}(5)),\\
&\quad (\mathbb{¢} \cdot ((\Sigma_0^2)^* \cdot \Sigma_0 \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^* \cdot \Sigma_0 \cdot c, d \cdot \# \to \#, \mathsf{Restart}(4)),\\
&\quad (\mathbb{¢} \cdot (u \cdot \#)^+ \cdot \$, \mathsf{Accept}) \text{ for all } u \in \Sigma_0^2,\\
M_4 &: (\mathbb{¢} \cdot ((\Sigma_0^2)^+ \cdot \Sigma_0 \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^+ \cdot \Sigma_0, c \cdot \$ \to \$, \mathsf{Restart}(5)),\\
&\quad (\mathbb{¢} \cdot ((\Sigma_0^2)^* \cdot \Sigma_0 \cdot c \cdot \#)^+ \cdot (\Sigma_0^2)^* \cdot \Sigma_0 \cdot c, d \cdot \# \to \#, \mathsf{Restart}(3)),\\
&\quad (\mathbb{¢} \cdot (u \cdot \#)^+ \cdot \$, \mathsf{Accept}) \text{ for all } u \in \Sigma_0^2,\\
M_5 &: (\mathbb{¢} \cdot \Sigma_0^+, c \cdot \# \to \#, \mathsf{Restart}(1)),\\
M_6 &: (\mathbb{¢} \cdot \Sigma_0^+, c \cdot \# \to \#, \mathsf{Restart}(3)).
\end{aligned}
$$

Clearly M_0 to M_6 are deterministic R-automata. Given an input of the form $w\#w\#\cdots\#w$, where $|w| = 2m > 2$, M_0 verifies that all syllables are of even length, and that they all end in the same letter, say c. This letter c is deleted from the last syllable, and M_5 is called, which simply deletes the last letter (that is, c) from the first syllable. Now M_1 is called, which in cooperation with M_2, removes the last letter from all the other syllables. Finally the tape content $w_1\#w_1\#\cdots\#w_1$ is reached, where $w = w_1c$. In this situation M_1 (or M_2) notices that all syllables are of odd length, and that they all end with the same letter, say d, which it then removes from the last syllable. Now using M_6, M_3, and M_4 this letter is removed from all other syllables. This process continues until either an error is detected, in which case $\mathcal{M}$ rejects, or until a tape content of the form $u\#u\#\cdots\#u$ is reached for a word $u \in \Sigma_0^2$, in which case $\mathcal{M}$ accepts. Thus, we see that $\mathcal{M}$ accepts the language L_{copy^*} working in mode $= 1$. □

Contrasting the positive results above we have the following result.

Theorem 4. *The language L_{copy^*} is not accepted by any strictly deterministic* CD-RRW*-system that is working in mode $= 1$.*

Proof. Let $\mathcal{M} = ((M_i, \sigma_i)_{i\in I}, I_0)$ be a strictly deterministic CD-RRW-system that accepts the language L_{copy^*} in mode $= 1$. We can assume that $I = \{0, 1, \ldots, m\}$, that $I_0 = \{0\}$, that $\sigma_i = \{i+1\}$ for all $i = 0, 1, \ldots, m-1$, and that $\sigma_m = \{s\}$ for some $s \in I$. Thus, each computation of $\mathcal{M}$ has the following structure:

$$w_0 \vdash_{\mathcal{M}}^{c^s} w_s \vdash_{M_s}^{c} w_{s+1} \vdash_{\mathcal{M}}^{c^{m-s-1}} w_m \vdash_{M_m}^{c} w_{m+1} \vdash_{M_s}^{c} w_{m+2} \vdash_{\mathcal{M}}^{c^{m-s-1}} \cdots,$$

that is, it is composed of a head $w_0 \vdash_{\mathcal{M}}^{c^s} w_s$ that consists of s cycles and of a sequence of meta-cycles of the form $w_s \vdash_{M_s}^{c} w_{s+1} \vdash_{\mathcal{M}}^{c^{m-s-1}} w_m \vdash_{M_m}^{c} w_{m+1}$ that consist of $m - s + 1$ cycles each.

Let $x := w\#w(\#w)^n$ be an input word with $w \in (\Sigma_0^2)^*$, where $|w|$ and the exponent n are sufficiently large. Then $x \in L_{\text{copy}^*}$, and hence, the computation of $\mathcal{M}$ that begins with the restarting configuration $q_0^{(0)}\mathbb{¢}x\$$ is accepting. We

will now analyze this computation. The factors w of x and their descendants in this computation will be denoted as *syllables*. To simplify the discussion we use indices to distinguish between different syllables.

$\mathcal{M}$ must compare each syllable w_i to all the other syllables. As $|w_i| = |w|$ is large, it can compare w_i to w_j for some $j \neq i$ only piecewise. However, during this process it needs to distinguish the parts that have already been compared from those parts that have not. This can only be achieved by rewriting w_i and w_j accordingly. Since no auxiliary symbols are available, this must be done by rewrite operations that either delete those parts of w_i and w_j that have already been compared, or that use the symbol # to mark the active positions within w_i and w_j. Hence, it takes at least $|w|/k$ many rewrite operations on w_i and the same number of rewrite operations on w_j to complete the comparison, where k is the maximal size of the read/write window of a component system of $\mathcal{M}$. As each rewrite operation is length-reducing, we see that after w_i and w_j have been compared completely, the remaining descendants of w_i and of w_j are of length at most $(1 - 1/k) \cdot |w|$, that is, information on w_i and on w_j has been lost during this process. It follows that $\mathcal{M}$ actually needs to compare w_i to all other syllables simultaneously.

Now assume that, for some j, no rewrite operation of the j-th meta-cycle is performed on the first two syllables of x. Then from that point on, no rewrite operation will be performed on the first syllable for many more meta-cycles. Indeed, as all component systems of $\mathcal{M}$ are deterministic, a change has to be propagated all the way from the third syllable back to the first syllable by a sequence of rewrites before another rewrite operation can affect w_1. This, however, means that at least $|w_2|/k$ many rewrite operations are applied to the second syllable, while no rewrite operation is applied to the first syllable. As observed above this destroys information on w_2 such that it is not possible anymore to verify whether or not w_1 and w_2 were identical.

It follows that at least one rewrite operation is applied to the first two syllables in each meta-cycle. As each rewrite operation is length-reducing, this implies that after at most $2 \cdot |w| + 1$ many meta-cycles the first two syllables have been completely erased. However, altogether these meta-cycles only execute $(2 \cdot |w| + 1) \cdot (m - s + 1)$ many rewrite operations, that is, only some of the syllables of x have been compared to w_1 and to w_2 during this process, provided that $n > (2 \cdot |w| + 1) \cdot (m - s + 1)$. It follows that $L_{=1}(\mathcal{M}) \neq L_{\mathrm{copy}^*}$. □

Corollary 1. *For all types* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}\}$*,*

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

Using similar techniques as in the proof of Theorem 4 the following result can be shown.

Theorem 5. L_{copy^m} *is not accepted by any strictly deterministic* CD-RRW-*system with less than* m *components working in mode* $= 1$*.*

By $\mathcal{L}_{=1}(\mathsf{det\text{-}strict\text{-}CD\text{-}X}(m))$ we denote the class of languages that are accepted by strictly deterministic CD-systems of restarting automata of type X that have

m components and that work in mode $= 1$. As the m-fold copy language L_{copy^m} is accepted by a strictly deterministic CD-R-system with m components that is working in mode $= 1$, the above result yields the following proper inclusion.

Corollary 2. *For all types* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}\}$ *and all* $m \geq 1$,

$$\mathcal{L}_{=1}(\text{det-strict-CD-}\mathsf{X}(m)) \subset \mathcal{L}_{=1}(\text{det-strict-CD-}\mathsf{X}(m+1)).$$

Thus, for each type $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}\}$, we have an infinite hierarchy between the class of languages accepted by deterministic restarting automata of type X and the class of languages accepted by nonforgetting deterministic restarting automata of that type. However, strictly deterministic CD-systems working in mode t are more expressive.

Proposition 2. *The language* L_{copy^*} *is accepted by a strictly deterministic* CD-RW*-system working in mode* t.

Proof. L_{copy^*} is accepted by the CD-RW-system $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ that is specified by $I := \{0, 1, 2\}$, $I_0 := \{0\}$, $\sigma(0) := \{1\}$, $\sigma(1) := \{2\}$, $\sigma(2) := \{0\}$. Here M_0, M_1, and M_2 are given through the following meta-instructions, where $\Sigma_0 := \{a, b\}$ and $c, d, e \in \Sigma_0$:

$$\begin{aligned}
M_0 \ &: (\mathrm{¢} \cdot ((\Sigma_0^2)^+ \cdot cd \cdot \#)^+ \cdot (\Sigma_0^2)^+, cd \cdot \$ \to \#\$),\\
&\quad\ (\mathrm{¢} \cdot cd \cdot (\# \cdot cd)^+ \cdot \$, \mathsf{Accept}),\\
M_1 \ &: (\mathrm{¢} \cdot (\Sigma_0^2)^+ \cdot (\#\# \cdot (\Sigma_0^2)^+)^*, cd \cdot \# \cdot e \to \#\# \cdot e),\\
M_2 \ &: (\mathrm{¢} \cdot (\Sigma_0^2)^+ \cdot (\# \cdot (\Sigma_0^2)^+)^*, \#\# \to \#),\\
&\quad\ (\mathrm{¢} \cdot (\Sigma_0^2)^+ \cdot (\# \cdot (\Sigma_0^2)^+)^+, \#\$ \to \$).
\end{aligned}$$

□

5 Concluding Remarks

We have seen that for restarting automata without auxiliary symbols the strictly deterministic CD-systems yield an infinite hierarchy that lies strictly in between the deterministic restarting automata and the nonforgetting deterministic restarting automata. However, the following related questions remain open:

1. Does this result extend to restarting automata with auxiliary symbols?
2. Are the locally deterministic CD-systems of restarting automata strictly more expressive than the globally deterministic CD-systems of restarting automata of the same type?
3. A nondeterministic CD-system of restarting automata is called *strict* if there is only a single initial system (that is, $|I_0| = 1$), and if the set of successors is a singleton for each component. It is easily seen that strict CD-systems are more expressive than restarting automata. However, is there a proper hierarchy of strict CD-systems, based on the number of component systems, that lies in between the (nondeterministic) restarting automata and the nonforgetting restarting automata?

References

1. Buntrock, G.: Wachsende kontext-sensitive Sprachen. Habilitationsschrift, Fakultät für Mathematik und Informatik, Universität Würzburg (1996)
2. Csuhaj-Varju, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems. A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, London (1994)
3. Dassow, J., Păun, G., Rozenberg, G.: Grammar systems. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 2, pp. 155–213. Springer, Heidelberg (1997)
4. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Restarting automata. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 965, pp. 283–292. Springer, Heidelberg (1995)
5. Lautemann, C.: One pushdown and a small tape. In: Wagner, K. (ed.) Dirk Siefkes zum 50. Geburtstag, Technische Universität Berlin and Universität Augsburg, pp. 42–47 (1988)
6. Messerschmidt, H., Otto, F.: On nonforgetting restarting automata that are deterministic and/or monotone. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 247–258. Springer, Heidelberg (2006)
7. Messerschmidt, H., Otto, F.: Cooperating distributed systems of restarting automata. Intern. J. Found. Comput. Sci. (to appear)
8. Messerschmidt, H., Stamer, H.: Restart-Automaten mit mehreren Restart-Zuständen. In: Bordihn, H. (ed.) Workshop Formale Methoden in der Linguistik und 14. Theorietag Automaten und Formale Sprachen, Proc., pp. 111–116. Institut für Informatik, Universität Potsdam (2004)
9. Oliva, K., Květoň, P., Ondruška, R.: The computational complexity of rule-based part-of-speech tagging. In: Matoušek, V., Mautner, P. (eds.) TSD 2003. LNCS (LNAI), vol. 2807, pp. 82–84, Springer, Heidelberg (2003)
10. Otto, F.: Restarting automata and their relations to the Chomsky hierarchy. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 55–74. Springer, Heidelberg (2003)
11. Otto, F.: Restarting automata. In: Ésik, Z., Martin-Vide, C., Mitrana, V. (eds.) Recent Advances in Formal Languages and Applications. Studies in Computational Intelligence, vol. 25, pp. 269–303. Springer, Heidelberg (2006)
12. Plátek, M., Lopatková, M., Oliva, K.: Restarting automata: Motivations and applications. In: Holzer, M. (ed.) Workshop Petrinets und 13. Theorietag Automaten und Formale Sprachen, Institut für Informatik, Technische Universität München, Garching, pp. 90–96 (2003)