Restarting Automata

Notes for a Course at the 3rd International PhD School in Formal Languages and Applications

Friedrich Otto

Fachbereich Mathematik/Informatik, Universität Kassel
34109 Kassel, Germany

otto@theory.informatik.uni-kassel.de

http://www.theory.informatik.uni-kassel.de

Abstract

The restarting automaton was introduced by Jančar et al. in 1995 in order to model the so-called 'analysis by reduction,' which is a technique used in linguistics to analyze sentences of natural languages that have a free word order. By now there are many different models of restarting automata, and their investigation has proved very fruitful in that they offer an opportunity to study the influence of various kinds of resources on their expressive power. Here we introduce and discuss the main variants of these automata, and present the main results that have been obtained until now. In particular, we investigate the relationship of the language classes that are defined through the various types of restarting automata to the classes of the Chomsky hierarchy, and we address some open problems.

Table of Contents

1 Introduction

The *analysis by reduction* is a technique used in linguistics to analyze sentences of natural languages that have a free word order [47, 48]. This analysis consists of a stepwise simplification of a sentence in such a way that the syntactical correctness or incorrectness of the sentence is not affected. After a finite number of steps either a correct simple sentence is obtained, or an error is detected. In the former case the given sentence is accepted as being syntactically correct; if, however, all possible sequences of simplifications yield errors, then the given sentence is not syntactically correct. In this way it is also possible to determine the dependencies between the various parts of the given sentence, and to disambiguate between certain morphological ambiguities contained in the sentence. To illustrate this, we consider the following (admittedly rather artificial) example:

They mean that the means she means are very mean.

To analyze this sentence we read it from left to right until we discover a phrase that can be simplified. For example we can delete the word that or the word very, thus obtaining the following sentences:

(1.) They mean the means she means are very mean.

(2.) They mean that the means she means are mean.

As both these simplifications are correct, we can conclude that the phrases that and very are not dependant on each other, that is, they are *independent*. From (1.) as well as from (2.) we can obtain the following sentence in the next step:

(3.) They mean the means she means are mean.

Next we simplify (3.) in two different ways by deleting the phrase the means and the phrase They mean, respectively:

(4.) They mean she means are mean.

(5.) The means she means are mean.

As (5.) is correct, while (4.) is not, we conclude that the phrase They mean *depends* on the phrase the means. Continuing with (5.) we obtain a simple sentence:

(6.) The means are mean.

This simple sentence is easily recognized as being syntactically correct, and therewith our analysis finishes with success. Thus, we conclude that the original sentence is syntactically correct. In addition we have obtained some information on the dependencies and independencies between certain parts of the sentence. Further, we can also use this analysis to determine that the first occurrence of the word mean in the given sentence is a verb, while the second occurrence of this word is an adjective, in this way solving some disambiguities occurring in the given sentence.

A generative device capable of simulating the analysis by reduction is the *sorted dependency 1-contextual grammar with regular selectors* by Martín-Vide and Gramatovici [12], which combines the *contextual grammars* of Marcus (see, e.g., [9, 27]) with dependency grammars (see, e.g., [8]).

As illustrated by Jančar et al. (cf., e.g., [21]) the restarting automaton was invented to model the analysis by reduction. In fact, many aspects of the work on restarting automata are motivated by the basic tasks of computational linguistics. The notions developed in the study of restarting automata give a rich taxonomy of constraints for various models of analysers and parsers. Already several programs are being used in Czech and German (corpus) linguistics that are based on the idea of restarting automata (cf., e.g., [39, 43]).

As defined in [17] the *restarting automaton* is a nondeterministic machine model that processes strings which are stored in a list (or a 'rubber' tape) with end markers. It has a finite control, and it has a read/write window with a finite look-ahead working on the list of symbols. As such it can be seen as a modification of the *list automaton* presented in [6] and the *forgetting automaton* [15, 16]. However, the restarting automaton can only perform two kinds of operations: *move-right transitions*, which shift the read/write window one position to the right, thereby changing the actual state, and combined *delete/restart transitions*, which delete some symbols from the read/write window, place this window over the left end of the list, and put the automaton back into its initial state. Hence, after performing a delete/restart transition a restarting automaton has no way to remember that it has already performed some steps of a computation. Further, by each application of a delete/restart transition the tape is shortened. In this aspect the restarting automaton is similar to the *contraction automaton* [46]. It follows that restarting automata are linearly space-bounded.

Subsequently Jančar et al. extended their model in various ways. Instead of simply deleting some symbols from the actual contents of the read/write window during a delete/restart transition, a restarting automaton *with rewriting* has combined *rewrite/restart* transitions that replace the contents of the read/write window by a shorter string [18]. Further, the use of auxiliary (that is, non-input) symbols was added to the model in [20], which yields the so-called RWW-*automaton*. Also in [20] the restart transition was separated from the rewrite transition so that, after performing a rewrite step, the automaton can still read the remaining part of the tape before performing a restart transition. This gives the so-called RRWW-*automaton*, which is required to execute exactly one rewrite transition between any two restart transitions. In addition, various notions of *monotonicity* have been discussed for the various types of restarting automata. It turned out that the monotone RWW- and RRWW-automata accept the context-free languages, and that all the various types of monotone deterministic RWW- and RRWW-automata accept the deterministic context-free languages. Finally, *move-left transitions* were added to the restarting automaton, which gave the so-called *two-way restarting automaton* (RLWW-automaton) [42]. This automaton can first scan its tape completely, and then move left to apply a rewrite transition to some factor of the tape contents.

Here we give an introduction to the restarting automaton and its many variants. We will compare the expressive power of the various types of restarting automata to each other and to the classes of the Chomsky hierarchy. Of particular interest will be the class GCSL of *growing context-sensitive languages*, which was considered in detail by Dahlhaus and Warmuth, who proved that the membership problems for these languages are solvable in polynomial time [7]. A detailed discussion of this class, which has many nice closure properties, can be found in [4] (see also [28]). We will see that even the most restricted type of restarting automaton is still sufficiently powerful to accept all deterministic context-free languages. On the other hand, the unrestricted RLWW-automaton is more expressive than the class GCSL. However, it is still open whether there is any context-sensitive language that cannot be accepted by any RLWW-automaton.

The paper is structured as follows. In the next section the basic variants of the restarting automaton are defined. The definitions given here differ slightly from the ones found in the literature (cf., e.g., [21, 38]), but they are easily seen to be equivalent to them. In Section 3 the monotone restarting automata are considered, and the relationships between the various language classes thus obtained and the (deterministic) context-free languages are described. Then we turn to the various types of deterministic restarting automata (Section 4), and we discuss their relationship to the Church-Rosser languages [29, 32]. In Section 5 the weakly monotone restarting automata are introduced, and it is shown that they characterize the growing context-sensitive languages. Then in Section 6 the language classes defined by the unrestricted types of nondeterministic restarting automata are compared to each other and to the classes of the Chomsky hierarchy, and in Section 7 we discuss in short the notion of *left-monotonicity* for restarting automata. This notion, which at first glance appears to be symmetric to the notion of monotonicity, shows a completely different behaviour for the deterministic types of restarting automata. In the final section further variants of the restarting automaton are discussed in short, and some open problems are presented.

Due to the introductory nature of this presentation many aspects of the restarting automaton are only touched upon. In addition we give almost no proofs, but for all results presented the appropriate references are cited. Furthermore, we illustrate the concepts presented through detailed examples.

Throughout the paper we will use the following notation. All alphabets considered are finite. For an alphabet Σ, Σ^* denotes the set of all strings over Σ including the empty string ε, and Σ^+ is the set of all non-empty strings. For a string w, $|w|$ denotes its *length*, and w^n is defined by $w^0 := \varepsilon$ and $w^{n+1} := w^n w$, where uv simply denotes the concatenation of the strings u and v. Further, w^R will denote the *mirror image* of w. For a language L, we will use the notation L^R to denote the language $L^R := \{ w^R \mid w \in L \}$. By DCFL, CFL, and CSL we denote the classes of deterministic context-free languages, context-free languages, and context-sensitive languages, respectively. Finally, for an automaton M, the language accepted by M is denoted as $L(M)$, and for a class A of automata, $\mathcal{L}(\mathsf{A})$ denotes the class of languages that are accepted by the automata from that class.

2 Definitions and Examples

In defining the restarting automaton and its main variants we will not follow the historical development outlined above, but we will first present the most general model, the RLWW-automaton, and then describe the other variants as restrictions thereof.

As indicated above a restarting automaton is a nondeterministic machine model that has a finite control and a read/write window that works on a list of symbols delimited by end markers:

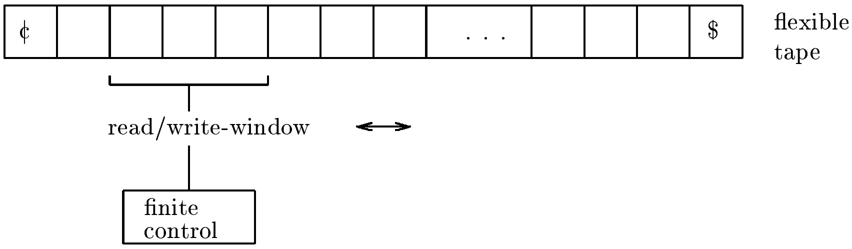

Figure 1. Schematic representation of a restarting automaton

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

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

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

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

and

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

The transition relation describes five different types of transition steps:

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

(2.) A *move-left step* is of the form $(q', \mathsf{MVL}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$, $u \notin ¢ \cdot \Gamma^*$. It causes M to shift the read/write window one position to the left and to enter state q'. This, however, is only possible if the window is not already at the left end of the tape.

(3.) A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u \in \mathcal{PC}^{(k)}$, $u \neq \$$, and $v \in \mathcal{PC}^{\leq(k-1)}$ such that $|v| < |u|$. It causes M to replace the contents u of the read/write window by the string v, and to enter state q'. Further, the read/write window is placed immediately to the right of the string v. However, some additional restrictions apply in that the border markers $¢$ and $\$$ must not disappear from the tape nor that new occurrences of these markers are created. Further, the read/write window must not move across the right border marker $\$$, that is, if the string u ends in $\$$, then so does the string v, and after performing the rewrite operation, the read/write window is placed on the $\$$-symbol.

(4.) A *restart step* is of the form $\mathsf{Restart} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to move its read/write window to the left end of the tape, so that the first symbol it sees is the left border marker $¢$, and to re-enter the initial state q_0.

(5.) An *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to halt and accept.

If $\delta(q, u) = \emptyset$ for some $q \in Q$ and $u \in \mathcal{PC}^{(k)}$, then M necessarily halts, and we say that M *rejects* in this situation. Further, the letters in $\Gamma \smallsetminus \Sigma$ are called *auxiliary symbols*.

A *configuration* of M is a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \varepsilon$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here $q \in Q$ represents the current state, $\alpha\beta$ is the current contents of the tape, and it is understood that the read/write window contains 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*. Further, we use Accept to denote the *accepting configurations*, which are those configurations that M reaches by executing an Accept instruction. A configuration of the form $\alpha q \beta$ such that $\delta(q, \beta_1) = \emptyset$, where β_1 is the current contents of the read/write window, is a *rejecting configuration*. A *halting configuration* is either an accepting or a rejecting configuration.

In general, the automaton M is *nondeterministic*, that is, there can be two or more instructions with the same left-hand side (q, u), and thus, there can be more than one computation for an input word. If this is not the case, the automaton is *deterministic*. We will use the prefix det- to denote the deterministic classes of restarting automata.

We observe that any finite computation of a two-way restarting automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the head moves along the tape performing MVR, MVL, and $\mathsf{Rewrite}$ operations until a $\mathsf{Restart}$ operation is performed and thus a new restarting configuration is reached. If no further $\mathsf{Restart}$ operation is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. We require that M performs *exactly one* $\mathsf{Rewrite}$ operation during any cycle – thus each new phase starts on a shorter word than the previous one. During a tail at most one $\mathsf{Rewrite}$ operation may be executed. By $\vdash^c_M$ we denote the execution of a complete cycle, and $\vdash^{c^*}_M$ is the reflexive transitive closure of this relation. It can be seen as the *rewrite relation* that is realized by M on the set of restarting configurations.

An input $w \in \Sigma^*$ is accepted by M, if there exists a computation of M which starts with the initial configuration $q_0¢w\$$, and which finally ends with executing an accept step.

Given an input of length n, an RLWW-automaton can execute at most n cycles. Thus, we have the following result, where NP and P denote the well-known complexity classes, and DCSL denotes the class of *deterministic context-sensitive languages*, that is, the space complexity class $\mathsf{DSPACE}(n)$.

Proposition 2.1 (a) $\mathcal{L}(\mathsf{RLWW}) \subseteq \mathrm{NP} \cap \mathsf{CSL}$.
(b) $\mathcal{L}(\mathsf{det\text{-}RLWW}) \subseteq \mathrm{P} \cap \mathsf{DCSL}$.

Before we continue with more definitions, we present a simple example.

Example 2.2 Let $M := (Q, \Sigma, \Sigma, ¢, \$, q_0, 3, \delta)$ be the deterministic RLWW-automaton that is defined by taking $Q := \{q_0, q_c, q_d, q_r\}$, $\Sigma := \{a, b, c, d\}$, and δ as given by the following table:

$$\begin{array}{llcl}
(1) & (q_0, x) & = & (q_0, \mathsf{MVR}) \quad \text{for all } x \in \{aaa, aab, abb, abc, bbb, bbc, bbd\}\ ,
\end{array}$$

$$\begin{array}{llcllll}
(2) & (q_0, ¢c\,\$) & = & \mathsf{Accept}, & (9) & (q_c, abc) & = & (q_r, c), \\
(3) & (q_0, ¢d\,\$) & = & \mathsf{Accept}, & (10) & (q_c, bbc) & = & (q_c, \mathsf{MVL}), \\
(4) & (q_0, ¢ab) & = & (q_0, \mathsf{MVR}), & (11) & (q_c, bbb) & = & (q_c, \mathsf{MVL}), \\
(5) & (q_0, ¢aa) & = & (q_0, \mathsf{MVR}), & (12) & (q_c, abb) & = & (q_r, b), \\
(6) & (q_0, bc\,\$) & = & (q_c, \mathsf{MVL}), & (13) & (q_d, bbd) & = & (q_d, \mathsf{MVL}), \\
(7) & (q_0, bd\,\$) & = & (q_d, \mathsf{MVL}), & (14) & (q_d, bbb) & = & (q_d, \mathsf{MVL}), \\
(8) & (q_r, -) & = & \mathsf{Restart}, & (15) & (q_d, abb) & = & (q_r, \varepsilon).
\end{array}$$

Obviously, M accepts the strings c and d immediately. So let $w \in \Sigma^+ \smallsetminus \{c, d\}$. Starting from the initial configuration $q_0¢w\$$, M will get stuck (and therewith reject) while scanning w from left to right unless w is of the form $w = a^m b^n c$ or $w = a^m b^n d$ for some positive integers m and n. In these cases the configuration $¢a^m b^{n-1} q_0 bc\,\$$ or $¢a^m b^{n-1} q_0 bd\,\$$ is reached by performing the transition steps (4) or (5) and (1) repeatedly, and then either the state q_c (6) or the state q_d is entered (7). Now M performs MVL-steps until the read/write window gets back to the boundary between the syllables a^m and b^n. If the actual state is q_c, that is, if w ends in c, then a factor ab is deleted (12); if the actual state is q_d, that is, if w ends in d, then a factor abb is deleted (15). In both case M enters the state q_r and restarts (8). Thus, we see that M accepts the following language

$$L_1 := \{\, a^n b^n c \mid n \geq 0 \,\} \cup \{\, a^n b^{2n} d \mid n \geq 0 \,\},$$

which is a well-known example of a context-free language that is not deterministic context-free.

Next we restate some basic facts about computations of restarting automata. These facts will be used repeatedly in the following, mostly without mentioning them explicitly.

Proposition 2.3 (Error Preserving Property).
Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an RLWW*-automaton, and let u, v be words over its* input *alphabet Σ. If $q_0¢u\$ \vdash_M^{c^*} q_0¢v\$$ holds and $u \notin L(M)$, then $v \notin L(M)$, either.*

Proposition 2.4 (Correctness Preserving Property).
Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an RLWW*-automaton, and let u, v be words over its* input *alphabet Σ. If $q_0¢u\$ \vdash_M^{c^*} q_0¢v\$$ is an initial segment of an* accepting *computation of M, then $v \in L(M)$.*

Also the following fact from [42] will be used repeatedly.

Proposition 2.5 (Pumping Lemma).
For any RLWW*-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, there exists a constant p such that the following holds. Assume that $q_0¢uvw\$ \vdash_M^c q_0¢uv'w\$$, where $u = u_1u_2u_3$ and $|u_2| = p$. Then there exists a factorization $u_2 = z_1z_2z_3$ such that z_2 is non-empty, and*

$$q_0¢u_1z_1(z_2)^iz_3u_3vw\$ \vdash_M^c q_0¢u_1z_1(z_2)^iz_3u_3v'w\$$$

holds for all $i \geq 0$, that is, z_2 is a 'pumping factor' in the above cycle. Similarly, such a pumping factor can be found in any factor of length p of w. Such a pumping factor can also be found in any factor of length p of a word accepted in a tail computation.

Each cycle of each computation of an RLWW-automaton M consists of three phases: first M scans its tape performing MVR- and MVL-instructions, then it executes a Rewrite step, and finally it scans its tape again performing MVR- and MVL-steps. Hence, in the first and the last phase of each cycle M behaves like a nondeterministic two-way finite-state acceptor (2NFA). In analogy to the proof that the language accepted by a 2NFA is regular (see, e.g, [14]), the following result can be established. Here a restarting automaton is called an RRWW-*automaton* if it does not use any MVL-transitions. Thus, in each cycle an RRWW-automaton can scan its tape only once from left to right.

Theorem 2.6 [42] *Let $M_L = (Q_L, \Sigma, \Gamma, ¢, \$, q_0, k, \delta_L)$ be an* RLWW*-automaton. Then there exists an* RRWW*-automaton $M_R = (Q_R, \Sigma, \Gamma, ¢, \$, q_0, k, \delta_R)$ such that, for all $u, v \in \Gamma^*$,*

$$q_0¢u\$ \vdash_{M_L}^c q_0¢v\$ \quad \text{if and only if} \quad q_0¢u\$ \vdash_{M_R}^c q_0¢v\$,$$

and the languages $L(M_L)$ and $L(M_R)$ coincide.

Thus, as far as nondeterministic restarting automata are concerned, the MVL-instruction is not needed. However, this does not hold for the deterministic restarting automata in general, as we will see later. For the RRWW-automata we have the following normalization result, which is easily proved by using standard techniques from automata theory.

Lemma 2.7 *Each* RRWW*-automaton M is equivalent to an* RRWW*-automaton M' that satisfies the following additional restriction:*

(*) *M' makes an accept or a restart step only when it sees the right border marker \$ in its read/write window.*

This lemma means that in each cycle of each computation and also during the tail of each accepting computation the read/write window moves all the way to the right before a restart is made, respectively, before the machine halts and accepts.

Based on this fact each cycle (and also the tail) of a computation of an RRWW-automaton consists of three phases. Accordingly, the transition relation of an RRWW-automaton can be described through a sequence of so-called *meta-instructions* [37] of the form $(R_1, u \to v, R_2)$, where R_1 and R_2 are regular languages, called the *regular constraints* of this instruction, and u

and v are strings such that $|u| > |v|$. The rule $u \to v$ stands for a Rewrite step of the RRWW-automaton M considered. On trying to execute this meta-instruction M will get stuck (and so reject) starting from the configuration $q_0 ¢ w \$$, if w does not admit a factorization of the form $w = w_1 u w_2$ such that $¢ w_1 \in R_1$ and $w_2 \$ \in R_2$. On the other hand, if w does have a factorization of this form, then one such factorization is chosen nondeterministically, and $q_0 ¢ w \$$ is transformed into $q_0 ¢ w_1 v w_2 \$$. In order to describe the tails of accepting computations we use meta-instructions of the form $(¢ \cdot R \cdot \$, \mathsf{Accept})$, where the strings from the regular language R are accepted by M in tail computations. We illustrate this concept by describing an RRWW-automaton for the language L_1 of Example 2.2.

Example 2.8 Let M_1 be the RRWW-automaton with input alphabet $\{a, b, c, d\}$ and without auxiliary symbols that is described by the following sequence of meta-instructions:

$$\begin{array}{llllll} (1) & (¢ \cdot a^*, & ab \to \varepsilon, & b^* \cdot c\,\$), & (3) & (¢ c\,\$, \quad \mathsf{Accept}), \\ (2) & (¢ \cdot a^*, & abb \to \varepsilon, & b^* \cdot d\,\$), & (4) & (¢ d\,\$, \quad \mathsf{Accept}). \end{array}$$

It is easily seen that $L(M_1) = L_1$ holds.

This way of describing an RRWW-automaton corresponds to the characterization of the class $\mathcal{L}(\mathsf{RRWW})$ by certain infinite prefix-rewriting systems as given in [38], Corollary 6.4.

Finally, we introduce some restricted types of restarting automata. A restarting automaton is called an RWW-*automaton* if it makes a restart immediately after performing a rewrite operation. In particular, this means that it cannot perform a rewrite step during the tail of a computation.

A cycle of a computation of an RWW-automaton M consists of two phases only. Accordingly, the transition relation of an RWW-automaton can be described by a finite sequence of *meta-instructions* of the form $(R, u \to v)$, where R is a regular language, and u and v are strings such that $|u| > |v|$, and meta-instructions of the form $(¢ \cdot R \cdot \$, \mathsf{Accept})$ for describing tail computations. This description corresponds to the characterization of the class $\mathcal{L}(\mathsf{RWW})$ by certain infinite prefix-rewriting systems as given in [38], Corollary 6.2.

Example 2.9 The following sequence of meta-instructions defines an RWW-automaton M_2 with input alphabet $\Sigma := \{a, b, c, d\}$ and tape alphabet $\Gamma := \Sigma \cup \{C, D\}$ for the language L_1:

$$\begin{array}{lllll} (1) & (¢ \cdot a^*, & ab \to C), & (5) & (¢ Cc\,\$, \quad \mathsf{Accept}), \\ (2) & (¢ \cdot a^*, & abb \to D), & (6) & (¢ Dd\,\$, \quad \mathsf{Accept}), \\ (3) & (¢ \cdot a^*, & aCb \to C), & (7) & (¢ c\,\$, \quad \mathsf{Accept}), \\ (4) & (¢ \cdot a^*, & aDbb \to D), & (8) & (¢ d\,\$, \quad \mathsf{Accept}). \end{array}$$

An RLWW-automaton is called an RLW-*automaton* if its tape alphabet Γ coincides with its input alphabet Σ, that is, if no auxiliary symbols are available. It is an RL-*automaton* if it is an RLW-automaton for which the right-hand side v of each Rewrite step $(q', v) \in \delta(q, u)$ is a scattered subword of the left-hand side u. Analogously, we obtain the RRW- and RR-automata from the RRWW-automaton and the RW- and R-automata from the RWW-automaton. The restarting automaton described in Example 2.2 is a deterministic RL-automaton, and the automaton M_1 of Example 2.8 is an RR-automaton. Further, for RLW- and RL-automata a result analogous to Theorem 2.6 holds.

Figure 2 summarizes the obvious inclusions between the language classes defined by the various types of restarting automata, the inclusions of Proposition 2.1, and the equalities of Theorem 2.6. In the following we are concerned with two main topics:

(a) Which of the inclusions in Figure 2 are proper?

(b) Where do the language classes defined by the various types of restarting automata lie in relation to other well-known language classes?

To answer these questions we will derive additional characterizations for some of these classes, and we will present some closure properties. The next result explains the role that the auxiliary symbols play in restarting automata. It is a slight generalization of a result that has been presented in [38] for RRWW- and for RWW-automata.

Theorem 2.10 *A language L is accepted by a (deterministic)* RLWW*-automaton if and only if there exist a (deterministic)* RLW*-automaton M_1 and a regular language R such that $L = L(M_1) \cap R$.*

This characterization yields the following closure property.

Corollary 2.11 *The language classes $\mathcal{L}$(RLWW), $\mathcal{L}$(RRWW), and $\mathcal{L}$(RWW) and their deterministic counterparts are closed under the operation of intersection with regular languages.*

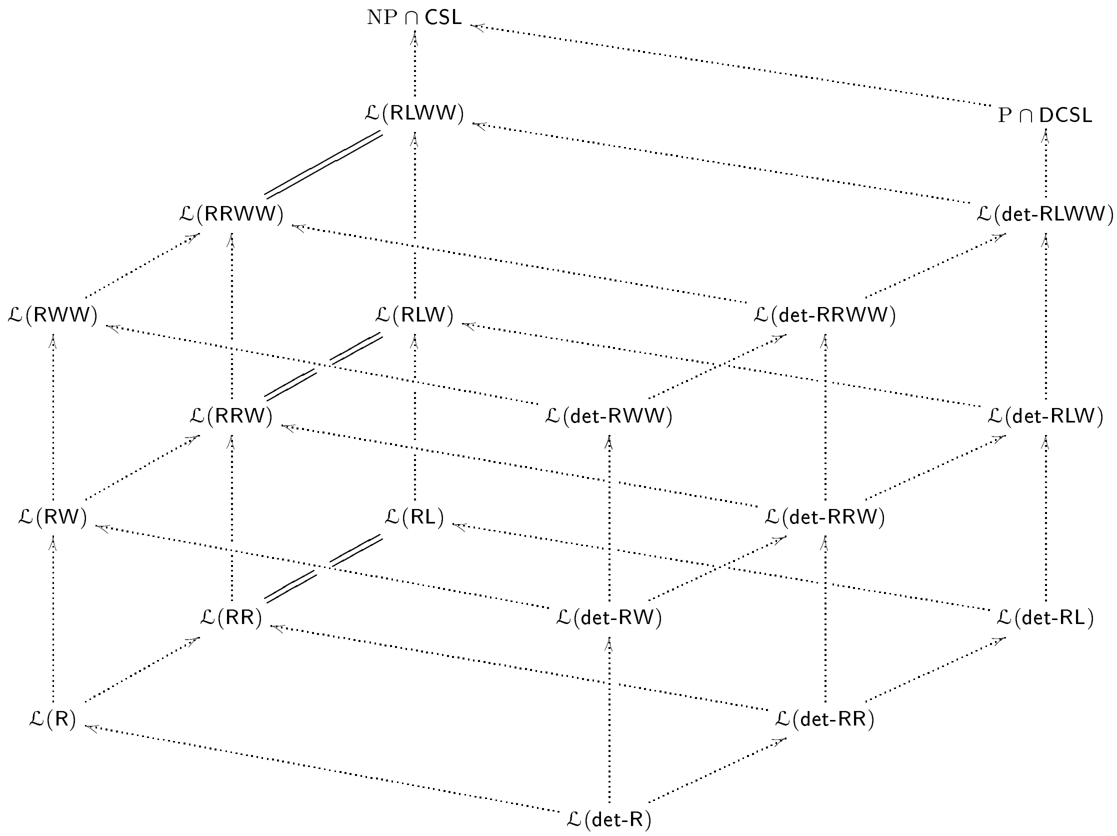

Figure 2. Inclusions between the language classes defined by the basic types of restarting automata. The equalities, expressed by =, follow from Theorem 2.6, while $\cdot \cdots\cdots\!\!> \cdot$ denotes a (not necessarily proper) inclusion.

Theorem 2.10 has another important consequence. Let $M = (Q, \Sigma, \Gamma, \text{¢}, \$, q_0, k, \delta)$ be an RLWW-automaton. The corresponding RLW-automaton M_1 and the regular language R of Theorem 2.10 are obtained by taking $M_1 := (Q, \Gamma, \Gamma, \text{¢}, \$, q_0, k, \delta)$ and $R := \Sigma^*$, that is, M_1 is obtained from M by simply declaring all auxiliary letters of M to input symbols. The identity mapping on Σ^* is obviously a reduction from the language $L(M)$ to the language $L(M_1)$. Thus, we have the following consequence.

Corollary 2.12 *The language class $\mathcal{L}$(RLWW) is reducible in linear time and constant space to the language class $\mathcal{L}$(RLW).*

An analogous result holds for the classes $\mathcal{L}$(RRWW) and $\mathcal{L}$(RWW) and the corresponding deterministic classes.

Next we turn to a reduction that replaces arbitrary rewrite operations by delete operations, that is, we reduce the class $\mathcal{L}$(RLW) to the class $\mathcal{L}$(RL).

Let $\Gamma_1 = \{a_1, \ldots, a_m\}$ be a finite alphabet, let k a positive integer, and let $\Gamma_2 := \{0, 1, c, d\}$. We define an encoding $\varphi_{k,m} : \Gamma_1 \to \Gamma_2^+$ as follows:

$$a_i \mapsto c1^{m+1-i}0^i(cd1^{m+1}0^{m+1})^k \quad (1 \le i \le m).$$

Then, for all $1 \leq i \leq m$,

$$|\varphi_{k,m}(a_i)| = (m+1) \cdot (2k+1) + 2k + 1 = (m+2) \cdot (2k+1).$$

The encoding $\varphi_{k,m}$ is naturally extended to strings by taking

$$\varphi_{k,m}(x_1 x_2 \ldots x_n) := \varphi_{k,m}(x_1) \ldots \varphi_{k,m}(x_n) \text{ for all } x_1, \ldots, x_n \in \Gamma_1,\ n \geq 0.$$

Observe that $\varphi_{k,m} : \Gamma_1^* \to \Gamma_2^*$ is indeed an encoding, that is, it is an injective mapping. It has the following important property.

Lemma 2.13 [25]
For all $u \in \Gamma_1^k$ and $v \in \Gamma_1^*$, if $|v| < k$, then $\varphi_{k,m}(v)$ is a scattered subword of $\varphi_{k,m}(u)$.

Based on this kind of encoding we obtain our second reduction.

Theorem 2.14 *If a language L is accepted by a (deterministic)* RLW*-automaton M with tape alphabet Γ_1 of size m and read/write window of size k, then there exists a (deterministic)* RL*-automaton M' which accepts the language $\varphi_{k,m}(L) \subseteq \Gamma_2^*$.*

Proof. Let $M = (Q, \Gamma_1, \Gamma_1, ¢, \$, q_0, k, \delta)$ be an RLW-automaton, $|\Gamma_1| = m$, and let $L = L(M)$. Let p denote the number $p := (m+2) \cdot (2k+1)$. Then we can construct an RL-automaton M' with tape alphabet Γ_2 and read/write window of size $k \cdot p$ as follows. Whenever the distance of the read/write window of M' from the left end of its tape is a multiple of p, then M' checks whether the read/write window contains a string that is the image $\varphi_{k,m}(u)$ for some $u \in \Gamma_1^*$. In the negative it rejects immediately, in the affirmative it performs an action that simulates the actual transition of M. It follows that M' accepts the language $\varphi_{k,m}(L)$, and that M' is deterministic, if M is. $\Box$

Obviously each encoding of the form $\varphi_{k,m}$ can be computed in linear time and constant space. Thus, we have the following result.

Corollary 2.15 *The language class $\mathcal{L}$(RLW) is reducible in linear time and constant space to the language class $\mathcal{L}$(RL).*

Again, an analogous result holds for the classes $\mathcal{L}$(RRW) and $\mathcal{L}$(RW) and the corresponding deterministic classes.

We close this section with another example. Let $L_{\text{copy}} := \{\, w\#w \mid w \in \{a,b\}^* \,\}$. This language is obviously context-sensitive, but it is not growing context-sensitive [5, 26].

Example 2.16 Let $M := (Q, \Sigma, \Gamma, ¢, \$, q_0, 3, \delta)$ be the RRWW-automaton on $\Sigma := \{a, b, \#\}$ and $\Gamma := \Sigma \cup \{\, [c,d] \mid c,d \in \{a,b\} \,\}$ that is given through the following sequence of meta-instructions, where $c, d \in \{a,b\}$:

(1) $(¢ \cdot \{a,b\}^*, cd\# \to [c,d]\#, \{a,b\}^* \cdot \$)$,	(4) $(¢ \cdot \{a,b\}^* \cdot \# \cdot \{a,b\}^*, [c,d]\$ \to \$, \varepsilon)$,
(2) $(¢ \cdot \{a,b\}^* \cdot [c,d]\# \cdot \{a,b\}^*, cd\$ \to [c,d]\$, \varepsilon)$,	(5) $(¢c\#c\$, \mathsf{Accept})$,
(3) $(¢ \cdot \{a,b\}^*, [c,d]\# \to \#, \{a,b\}^* \cdot [c,d]\$)$,	(6) $(¢\#\$, \mathsf{Accept})$.

Then it is easily verified that $L(M) = L_{\text{copy}}$ holds.

This example shows that the language class $\mathcal{L}$(RRWW) contains languages that are not growing context-sensitive. But we can do much better than that.

Example 2.17 Let $M' := (Q, \Sigma, \Gamma, ¢, \$, q_0, 4, \delta)$ be the RWW-automaton on $\Sigma := \{a, b, \#\}$ and $\Gamma := \Sigma \cup \{\#'\} \cup \Gamma'$, where $\Gamma' := \{\, [c,d,S] \mid c,d \in \{a,b\}, S \in \{0,1\} \,\}$, that is given through the following sequence of meta-instructions. Here $c,d,e,f \in \{a,b\}$, $S, S' \in \{0,1\}$, and $\bar{S} := 1 - S$:

(1) $(¢, cd \to [c,d,0])$,
(2) $(¢ \cdot \Gamma'^* \cdot [e,f,S], cd \to [c,d,\bar{S}])$,
(3) $(¢[c,d,0] \cdot \{a,b\}^* \cdot \#, cd \to [c,d,0])$,
(4) $(¢ \cdot \Gamma'^* \cdot [c,d,S] \cdot \{a,b\}^* \cdot \# \cdot \Gamma'^* \cdot [e,f,\bar{S}], cd \to [c,d,S])$,
(5) $(¢ \cdot \Gamma'^*, [c,d,S] \cdot X \cdot [e,f,S'] \to \#')$ for all $X \in \{\#, \#'\}$,
(6) $(¢ \cdot \Gamma'^*, [c,d,S] \cdot g \cdot X \cdot [e,f,S'] \to g\#')$ for all $g \in \{a,b\}, X \in \{\#, \#'\}$,
(7) $(¢X\$, \mathsf{Accept})$ for all $X \in \{\#, \#'\}$,
(8) $(¢gXg\$, \mathsf{Accept})$ for all $g \in \{a,b\}, X \in \{\#, \#'\}$.

Given the string $abaab\#abaab$ as input, M' can execute the following computation:

$$\begin{array}{rll}
q_0 ¢\, \underline{ab}aab\#abaab\$ & \vdash^c_{(1)} & q_0 ¢\, [a,b,0]aab\#\underline{ab}aab\$ \\
& \vdash^c_{(3)} & q_0 ¢\, [a,b,0]\underline{aa}b\#[a,b,0]aab\$ \\
& \vdash^c_{(2)} & q_0 ¢\, [a,b,0][a,a,1]b\#[a,b,0]\underline{aa}b\$ \\
& \vdash^c_{(4)} & q_0 ¢\, [a,b,0]\underline{[a,a,1]b\#[a,b,0]}[a,a,1]b\$ \\
& \vdash^c_{(6)} & q_0 ¢\, \underline{[a,b,0]b\#'[a,a,1]}b\$ \\
& \vdash^c_{(6)} & q_0 ¢\, b\#'b\$ \\
& \vdash_{(8)} & \mathsf{Accept}.
\end{array}$$

We see that the meta-instructions (1) and (2) compress the first $\{a,b\}$-syllable $a_1a_2a_3a_4a_5a_6\ldots$ of a given input into a string of the form $[a_1,a_2,0][a_3,a_4,1][a_5,a_6,0]\ldots$. The meta-instructions (3) and (4) compress the second $\{a,b\}$-syllable in the same way. The alternating values in the third components of the Γ'-symbols guarantee that between any two applications of meta-instructions (3) or (4) at least one application of meta-instruction (2) takes place. It follows that the meta-instructions (1) to (4) can compress the first two $\{a,b\}$-syllables completely only if the second syllable is a scattered subword of the first syllable. Now the meta-instructions (5) and (6) delete the compressed syllables in a synchronous manner, and M' accepts if and only if the two syllables have the same length. Together these observations imply that M' accepts an input w if and only if w has the form $w = u\#u$ for some string $u \in \{a,b\}^*$, that is, we see that $L(M') = L_{\text{copy}}$.

By the corresponding analogue to Theorem 2.10, there exists an RW-automaton M_1' and a regular language R such that $L(M_1') \cap R = L(M') = L_{\text{copy}}$. As $L_{\text{copy}} \notin \mathsf{GCSL}$, and as the class GCSL is closed under the operation of intersection with regular languages, it follows that $L(M_1') \notin \mathsf{GCSL}$. Finally, as 12 is the cardinality of the tape alphabet of M_1', it follows from Theorem 2.14 that $\varphi_{4,12}(L(M_1'))$ is accepted by some R-automaton. As GCSL is closed under inverse morphisms, this implies that the language $\varphi_{4,12}(L(M_1'))$ is not growing context-sensitive, either. Thus, we obtain the following non-inclusion result.

Corollary 2.18 $\mathcal{L}(\mathsf{R}) \not\subseteq \mathsf{GCSL}$.

Hence, already the R-automaton has a fairly large expressive power.

3 Monotone Restarting Automata

In [18] the notion of monotonicity is introduced for restarting automata. Here we consider a slightly more general notion, which is taken from [22]. Let M be an RLWW-automaton. Each computation of M can be described by a sequence of cycles $C_1, C_2 \ldots, C_n$, where C_n is the last cycle, which is followed by the tail of the computation. Each cycle C_i of this computation contains a unique configuration of the form $¢xquy\$$ such that q is a state and $(q',v) \in \delta(q,u)$ is the Rewrite step that is applied during this cycle. By $D_r(C_i)$ we denote the *right distance* $|y\$|$ of this cycle, and $D_l(C_i) := |¢x|$ is the *left distance* of this cycle. The sequence of cycles $C_1, C_2, \ldots, C_n$ is called *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$ holds. A computation of M is called *monotone* if the corresponding sequence of cycles is monotone. Observe that the tail of the computation is not taken into account here. Finally, the RLWW-automaton M is called *monotone* if each of its computations that starts from an initial configuration is monotone.

Here we want to compare the expressive power of the various types of monotone restarting automata to each other. We use the prefix mon- to denote the classes of monotone restarting automata. The first result characterizes the expressive power of the monotone restarting automata with auxiliary symbols.

Theorem 3.1 [21] $\mathcal{L}(\text{mon-RLWW}) = \mathcal{L}(\text{mon-RRWW}) = \mathcal{L}(\text{mon-RWW}) = \mathsf{CFL}$.

Proof. From a context-free grammar G in Chomsky normal form we can construct a monotone RWW-automaton M such that, given an input word w, M tries to construct a rightmost derivation of w in G in reverse order. M will succeed if and only if such a derivation exists, that is, if and only if $w \in L(G)$. Hence, $L(M) = L(G)$, which shows that each context-free language is accepted by a monotone RWW-automaton.

Conversely, assume that the language L is accepted by a monotone RLWW-automaton. Because of Theorem 2.6 there is a monotone RRWW-automaton M' that also accepts the language L. Let w be an input string, and let $C_1, C_2, \ldots, C_m$ be the sequence of cycles that corresponds to a computation of M' that starts from the initial configuration $q_0¢w\$$. For each cycle the tape contents can be divided into three parts: a prefix w_1, the part u within the read/write window during the execution of the actual Rewrite step, and the remaining suffix w_2. As M' is monotone, the positions where the Rewrite steps are performed move from left to right across the tape. Accordingly, the prefixes w_1 can be stored in a pushdown store, and the suffices w_2 can be interpreted as the remaining part of the input. Thus, M' can be simulated by a pushdown automaton (see [21] for the details which are technically quite involved, as M' can already scan the suffix w_2 completely in each cycle), that is, $L(M)$ is context-free. □

If the monotone RRWW-automaton M is deterministic, then the above construction can be used to show that the language $L(M)$ is deterministic context-free. On the other hand, based on the characterization of the deterministic context-free languages by LR(0)-grammars (see, e.g., [14]) it can be shown that already the monotone det-R-automata accept all deterministic context-free languages, which yields the following characterization.

Theorem 3.2 [18, 19, 20, 21]
For all types $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$, $\mathcal{L}(\mathsf{det\text{-}mon\text{-}X}) = \mathsf{DCFL}$.

Example 2.2 shows that in the following sequence of inclusions

$$\mathsf{DCFL} = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RR}) \subset \mathcal{L}(\mathsf{det\text{-}mon\text{-}RL}) \subseteq \mathcal{L}(\mathsf{det\text{-}mon\text{-}RLW}) \subseteq \mathcal{L}(\mathsf{det\text{-}mon\text{-}RLWW}),$$

the first one is proper. To investigate the classes of nondeterministic monotone restarting automata without auxiliary symbols, we consider the example language L_1 from the previous section and the following languages:

$$\begin{aligned} L_2 &:= \{ a^n b^n \mid n \geq 0 \} \cup \{ a^n b^m \mid m > 2n \geq 0 \}, \\ L_3 &:= \{f, ee\} \cdot \{ a^n b^n \mid n \geq 0 \} \cup \{g, ee\} \cdot \{ a^n b^m \mid m > 2n \geq 0 \}, \\ L_4 &:= \{ a^n b^m \mid 0 \leq n \leq m \leq 2n \}. \end{aligned}$$

Lemma 3.3 $L_1 \in \mathcal{L}(\mathsf{mon\text{-}RR}) \setminus \mathcal{L}(\mathsf{RW})$.

Proof. In Example 2.2 a deterministic RL-automaton is presented for the language L_1. It is easily verified that this automaton is monotone. Hence, by Theorem 2.6 there exists a monotone (nondeterministic) RR-automaton for this language.

Now assume that L_1 were accepted by some RW-automaton M_1 with read/write window of size k. For each input of the form $a^n b^{2n} d$, M_1 has an accepting computation. If $n > k$ is sufficiently large, then the corresponding computation must contain Restart steps, that is, it consists of a sequence of cycles followed by a tail computation. During the first cycle M_1 must delete a factor of the form $a^r b^{2r}$ because of the correctness preserving property. Here $3r \leq k < n$. As M_1 restarts immediately after performing a Rewrite step, it can apply the same Rewrite operation to the input $a^n b^{n+r} c$, which yields the cycle $q_0¢a^n b^{n+r} c\$ \vdash^c_{M_1} q_0¢a^{n-r} b^{n-r} c\$$. Now $a^{n-r} b^{n-r} c \in L_1$, while $a^n b^{n+r} c \notin L_1$, contradicting the error preserving property. Hence, L_1 is not accepted by any RW-automaton. □

Using the same kind of reasoning it can be shown that the context-free language L_2 is not in $\mathcal{L}(\mathsf{RRW})$, and that $L_3 \in \mathcal{L}(\mathsf{mon\text{-}RW}) \setminus \mathcal{L}(\mathsf{RR})$ and $L_4 \in \mathcal{L}(\mathsf{mon\text{-}R}) \setminus \mathcal{L}(\mathsf{det\text{-}RLW})$ hold. Together with the above results these facts yield all the relations displayed in Figure 3.

Finally we turn to the problem of deciding whether a given restarting automaton is monotone. The solution rests on the following two technical results, the first of which is a straightforward generalization of the result that each monotone RRWW-automaton can be simulated by a pushdown automaton.

Lemma 3.4 *Given an* RRWW*-automaton* M *with input alphabet* Σ, *a pushdown automaton* P_M *can be constructed such that*

$$L(P_M) = \{ w \in \Sigma^* \mid \text{There exists an accepting monotone computation for } M \text{ on input } w \}.$$

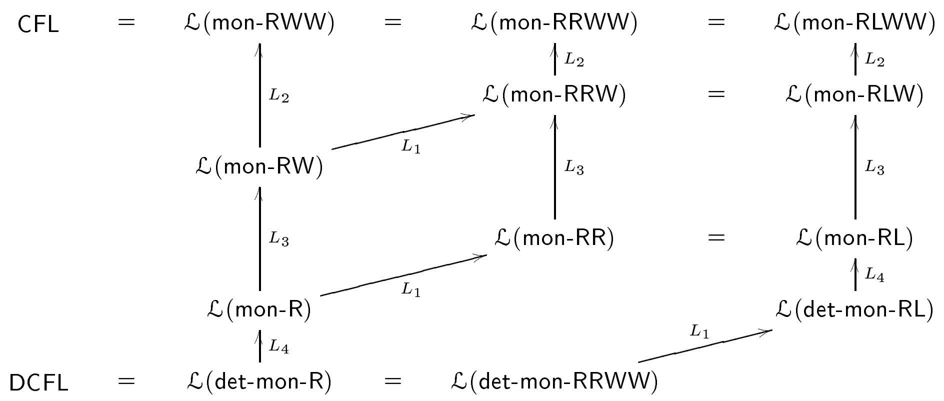

Figure 3. The taxonomy of monotone restarting automata

Let M be an RRWW-automaton with tape alphabet Γ. We say that a word $w \in \Gamma^*$ *causes a non-monotone computation* of M, if there exist two cycles

$$C_1 : q_0¢w\$ \vdash_M^c q_0¢v\$ \quad \text{and} \quad C_2 : q_0¢v\$ \vdash_M^c q_0¢z\$$$

such that $D_r(C_1) < D_r(C_2)$. By $L_{\mathrm{nm}}(M)$ we denote the language consisting of all words $w \in \Gamma^*$ that cause a non-monotone computation of M.

Lemma 3.5 *Given an* RRWW*-automaton M, a finite-state acceptor can be constructed for the language $L_{\mathrm{nm}}(M)$.*

Proof. The finite-state acceptor simulates two successive cycles C_1 and C_2 of M in parallel. It accepts if and only if the cycles simulated satisfy the condition that $D_r(C_1) < D_r(C_2)$. □

It follows in particular that the language $L_{\mathrm{nm}}(M)$ is regular. From Lemma 3.4 we obtain a pushdown automaton P'_M for M such that, for each input word $w \in \Sigma^*$ of M and each restarting configuration $q_0¢z\$$ that M can reach from the initial configuration $q_0¢w\$$ through a monotone sequence of cycles, P'_M has an accepting computation that ends with its pushdown store containing the word z. Let $SC_{P'_M}(\Sigma^*)$ denote the language that consists of those words z for which there exists an accepting computation of P'_M on some input $w \in \Sigma^*$ that ends with the pushdown store containing z. Then M is *not* monotone, if and only if the language $SC_{P'_M}(\Sigma^*)$ contains a word that causes a non-monotone computation of M, that is, if and only if $SC_{P'_M}(\Sigma^*) \cap L_{\mathrm{nm}}(M)$ is non-empty. According to [13] the language $SC_{P'_M}(\Sigma^*)$ is regular, and a finite-state acceptor for it can be constructed effectively from P'_M. Thus, we have the following decidability result.

Theorem 3.6 [22] *It is decidable whether a given* RRWW*-automaton is monotone.*

By Theorem 2.6 we can associate with each RLWW-automaton M an RRWW-automaton M_R such that, in each computation, M_R executes the same sequence of cycles as M. Hence, M is monotone if and only if M_R is. Thus, Theorem 3.6 holds for RLWW-automata in general.

4 Deterministic Restarting Automata

As we will see, there is a close correspondence between certain types of deterministic restarting automata and the Church-Rosser languages. Therefore, we shortly restate the definition of the Church-Rosser languages and their main properties.

Let Σ be a finite alphabet. A *string-rewriting system* R on Σ is a subset of $\Sigma^* \times \Sigma^*$. It induces several binary relations on Σ^*, the most basic of which is the *single-step reduction relation* $\rightarrow_R := \{ (u\ell v, urv) \mid u, v \in \Sigma^*, (\ell \rightarrow r) \in R \}$. Its reflexive and transitive closure is the *reduction relation* $\rightarrow_R^*$ induced by R. If $u \rightarrow_R^* v$, then u is an *ancestor* of v, and v is a *descendant* of u. If there is no $v \in \Sigma^*$ such that $u \rightarrow_R v$ holds, then the string u is called *irreducible* (mod R). By $\mathrm{IRR}(R)$ we denote the set of all irreducible strings. If R is finite, then $\mathrm{IRR}(R)$ is obviously a regular language. The string-rewriting system R is called

– *length-reducing* if $|\ell| > |r|$ holds for each rule $(\ell \to r) \in R$,

– *confluent* if, for all $u, v, w \in \Sigma^*$, $u \to_R^* v$ and $u \to_R^* w$ imply that v and w have a common descendant.

If a string-rewriting system R is length-reducing, then each reduction sequence starting with a string of length n has itself at most length n. If, in addition, R is confluent, then each string $w \in \Sigma^*$ has a unique irreducible descendant $\hat{w} \in \mathrm{IRR}(R)$, which can be computed from w in linear time (see, e.g., [3]). This observation was one of the main reasons to introduce the Church-Rosser languages in [29, 32].

Definition 4.1 *A language $L \subseteq \Sigma^*$ is a* Church-Rosser language *if there exist an alphabet $\Gamma \supsetneq \Sigma$, a finite, length-reducing, confluent string-rewriting system R on Γ, two strings $t_1, t_2 \in (\Gamma \smallsetminus \Sigma)^* \cap \mathrm{IRR}(R)$, and a symbol $Y \in (\Gamma \smallsetminus \Sigma) \cap \mathrm{IRR}(R)$ such that, for all $w \in \Sigma^*$, $t_1wt_2 \to_R^* Y$ if and only if $w \in L$.*

By CRL we denote the class of Church-Rosser languages. It is known that the inclusions $\mathsf{DCFL} \subset \mathsf{CRL} \subset \mathsf{GCSL} \subset \mathsf{CSL}$ are proper, and that the classes CRL and CFL are incomparable under set inclusion [5, 29, 32]. The Church-Rosser languages have been characterized by certain types of two-pushdown automata.

Definition 4.2 *A* two-pushdown automaton (TPDA) *with* pushdown windows of size k *is a nondeterministic automaton $P = (Q, \Sigma, \Gamma, \delta, k, q_0, \perp, t_1, t_2, F)$, where Q is the finite set of states, Σ is the finite input alphabet, Γ is the finite tape alphabet containing Σ, $q_0 \in Q$ is the initial state, $\perp \in \Gamma \smallsetminus \Sigma$ is the bottom marker of the pushdown stores, $t_1, t_2 \in (\Gamma \smallsetminus \Sigma)^*$ is the preassigned contents of the first and second pushdown store, respectively, $F \subseteq Q$ is the set of final (or halting) states, and δ is the transition relation. To each triple (q, u, v), where $q \in Q$ is a state, $u \in \Gamma^k \cup \{\perp\} \cdot \Gamma^{<k}$ is the contents of the topmost part of the first pushdown store, and $v \in \Gamma^k \cup \Gamma^{<k} \cdot \{\perp\}$ is the contents of the topmost part of the second pushdown store, it associates a finite set of triples from $Q \times \Gamma^* \times \Gamma^*$. The automaton P is a* deterministic two-pushdown automaton (DTPDA), *if δ is a (partial) function.*

A configuration of a (D)TPDA is described by uqv, where $q \in Q$ is the actual state, $u \in \Gamma^*$ is the contents of the first pushdown store with the first symbol of u at the bottom and the last symbol of u at the top, and $v \in \Gamma^*$ is the contents of the second pushdown store with the last symbol of v at the bottom and the first symbol of v at the top. For an input string $w \in \Sigma^*$, the corresponding initial configuration is $\perp t_1 q_0 w t_2 \perp$. The computation relation of a (D)TPDA P is denoted by $\vdash_P^*$. The (D)TPDA P accepts with empty pushdown stores, that is, $L(P) := \{\, w \in \Sigma^* \mid \perp t_1 q_0 w t_2 \perp \ \vdash_P^* q$ for some $q \in F \,\}$ is the *language accepted by* P. Observe that the input is provided to a TPDA as the initial contents of its second pushdown store, and that in order to accept a TPDA is required to empty its pushdown stores. Thus, it is forced to consume its input completely.

A (D)TPDA is called *shrinking* if there exists a *weight-function* $\varphi : Q \cup \Gamma \to \mathbb{N}_+$ such that, for all transitions $(p, u', v') \in \delta(q, u, v)$, $\varphi(u'pv') < \varphi(uqv)$. Here φ is extended to a morphism $\varphi : (Q \cup \Gamma)^* \to \mathbb{N}$ in the obvious way. A (D)TPDA is called *length-reducing* if $|u'v'| < |uv|$ holds for all transitions $(p, u', v') \in \delta(q, u, v)$. Obviously, the length-reducing TPDA is a special case of the shrinking TPDA.

From the definitions it is easily seen that a language is growing context-sensitive, if it is accepted by a length-reducing TPDA, and that it is Church-Rosser, if it is accepted by a length-reducing DTPDA. Actually, the following characterizations have been established in [5, 33, 34].

Theorem 4.3 [33]

(a) *A language is Church-Rosser, if and only if it is accepted by a shrinking* DTPDA, *if and only if it is accepted by a length-reducing* DTPDA.

(b) *A language is growing context-sensitive, if and only if it is accepted by a shrinking* TPDA, *if and only if it is accepted by a length-reducing* TPDA.

A (deterministic) TPDA P that is length-reducing can be simulated by a (deterministic) RWW-automaton M [36]. A configuration $\bot uqv\bot$ of P, where q is a state, is encoded by the tape contents $¢\hat{u}qv\$$ of M, where $\hat{u}$ is a copy of u that consists of marked symbols. M simply moves its read/write window to the right until it encounters the left-hand side of a transition of P, which it then simulates by applying a rewrite step. We must, however, ensure that during the first cycle of a computation, which starts with the tape contents $¢w\$$ for some input word w, M not only simulates the first step of P on input w, but that it also inserts the encoding of the corresponding state of P into its tape contents.

Conversely, it can be shown that each deterministic RRWW-automaton M can be simulated by a shrinking DTPDA P [38]. The key observation is the following. If M rewrites $¢xuy\$$ into $¢xvy\$$, then in the next cycle M must read all of x and at least the first symbol of $vy\$$ before it can execute another Rewrite step. However, while reading the prefix x, M will undergo the same state transitions as in the previous cycle. Thus, instead of reading x again, P simply stores the information on the states of M together with the symbols of x on its first pushdown store. Further, after performing the Rewrite step $¢xpuy\$ \vdash ¢xvqy\$$, where p and q are the corresponding states, M will continue with MVR-steps until it reaches the right delimiter \$, and then it either restarts, halts and accepts, or halts without accepting. Hence, with the string y, we can associate two subsets $Q_+(y)$ and $Q_{\rm rs}(y)$ of the state set Q of M as follows:

A state $q \in Q$ is contained in $Q_+(y)$ ($Q_{\rm rs}(y)$) if, starting from the configuration $¢qy\$$, M makes only MVR-steps until it scans the \$-symbol, and halts and accepts (respectively, restarts) then.

The DTPDA P stores the string y on its second pushdown. In addition, for each suffix z of y, its stores encodings of the sets $Q_+(z)$ and $Q_{\rm rs}(z)$ under the first symbol of z. Using this information it does not need to simulate the MVR-steps of M step-by-step, but it can immediately simulate a restart or halt and accept (or reject). As P starts with the input on its second pushdown, it needs a preprocessing stage to determine and to store the sets $Q_+(z)$ and $Q_{\rm rs}(z)$, but this is easily realized by a shrinking DTPDA. This yields the following results.

Theorem 4.4 [36, 38] (a) $\mathsf{CRL} = \mathcal{L}(\text{det-RWW}) = \mathcal{L}(\text{det-RRWW})$.
(b) $\mathsf{GCSL} \subseteq \mathcal{L}(\text{RWW}) \subseteq \mathcal{L}(\text{RRWW})$.

Finally we turn to the relationships between the language classes that are defined by the various types of deterministic restarting automata. It was pointed out in Section 3 that the language $L_2 = \{\, a^nb^n \mid n \geq 0 \,\} \cup \{\, a^nb^m \mid m > 2n \geq 0 \,\}$ is not accepted by any RRW-automaton. However, it is accepted by a deterministic RWW-automaton because of the theorem above and the following result.

Lemma 4.5 *The language L_2 is a Church-Rosser language.*

Proof. It suffices to describe a shrinking DTPDA P for the language L_2. This DTPDA proceeds as follows, given an input of the form a^nb^m:

(1.) By repeatedly dividing the numbers of occurrences of a's and b's by two, the binary representations $\mathsf{bin}(n)$ and $\mathsf{bin}(m)$ of the exponents n and m are determined. The string $\mathsf{bin}(n)$ is collected on the bottom of the first pushdown, while the string $\mathsf{bin}(m)$ is collected on the bottom of the second pushdown. Thus, after a finite number of steps P has transformed its input into the combined stack contents $\mathsf{bin}(n)^R\#\mathsf{bin}(m)$. Here # is a special symbol marking the border between the two binary representations, and $\mathsf{bin}(n)^R$ denotes the mirror image of the string $\mathsf{bin}(n)$.

(2.) Next P checks whether $n = m$ or whether $m > 2n$ holds. Let $k := |\mathsf{bin}(n)|$ and $l := |\mathsf{bin}(m)|$. Obviously $n = m$ if and only if $\mathsf{bin}(n) = \mathsf{bin}(m)$. Further, $m > 2n$ if and only if

– $l > k + 1$, or

– $l = k + 1$ and (i) $\mathsf{bin}(m) = \mathsf{bin}(n)1$ or
(ii) $\mathsf{bin}(m) = \mathsf{bin}(r)d_0$ for some integer $r > n$ and any digit d_0.

Comparing $\mathsf{bin}(n)$ and $\mathsf{bin}(m)$ digit by digit, P can check these conditions. Further, it is easily seen that this computation can be realized in a weight-reducing manner. □

For establishing further separation results, we consider the following example languages:

$$
\begin{aligned}
L_5 &:= \{\, a^{2^n} \mid n \geq 0 \,\},\\
L_6 &:= L_{6,1} \cup L_{6,2} \cup L_{6,3},\\
L_{6,1} &:= \{\, (ab)^{2^n - i} c (ab)^i \mid n \geq 0, 0 \leq i \leq 2^n \,\},\\
L_{6,2} &:= \{\, (ab)^{2^n - 2i} (abb)^i \mid n \geq 1, 0 \leq i \leq 2^{n-1} \,\},\\
L_{6,3} &:= \{\, (abb)^{2^n - i} (ab)^i \mid n \geq 0, 0 \leq i \leq 2^n \,\},\\
L_7 &:= L_{7,1} \cup L_{7,2},\\
L_{7,1} &:= \{\, a^{2^n - 2i} c a^i \mid n \geq 1, 0 \leq 2i < 2^n \,\},\\
L_{7,2} &:= \{\, a^i d a^{2^n - 2i} \mid n \geq 1, 0 \leq 2i < 2^n \,\}.
\end{aligned}
$$

It can be shown that $L_6 \in \mathcal{L}(\mathsf{det\text{-}RR}) \setminus \mathcal{L}(\mathsf{RW})$, and $L_7 \in \mathcal{L}(\mathsf{det\text{-}RW}) \setminus \mathcal{L}(\mathsf{RL})$. Further, based on L_5 a non-context-free language L_5' is described in [18] that belongs to $\mathcal{L}(\mathsf{det\text{-}R})$. In addition, based on the RRWW-automaton for the language L_{copy} described in Example 2.16, a deterministic RLWW-automaton can be obtained for this language. Hence, by Theorem 2.10 there exists a language $L'_{\text{copy}} \in \mathcal{L}(\mathsf{det\text{-}RLW})$ and a regular language R such that $L_{\text{copy}} = L'_{\text{copy}} \cap R$, and by Theorem 2.14 there exists a morphism $\varphi_{m,k}$ such that the language $\tilde{L}_{\text{copy}} := \varphi_{m,k}(L'_{\text{copy}}) \in \mathcal{L}(\mathsf{det\text{-}RL})$. As L_{copy} is not growing context-sensitive, and as the class CRL is closed under the operation of intersection with regular languages and under inverse morphisms, it follows that none of these languages is Church-Rosser. The same reasoning shows that the language $\tilde{L}_{\text{copy}}$ is not context-free. Together with the above results these facts yield the proper inclusions displayed in Figure 4.

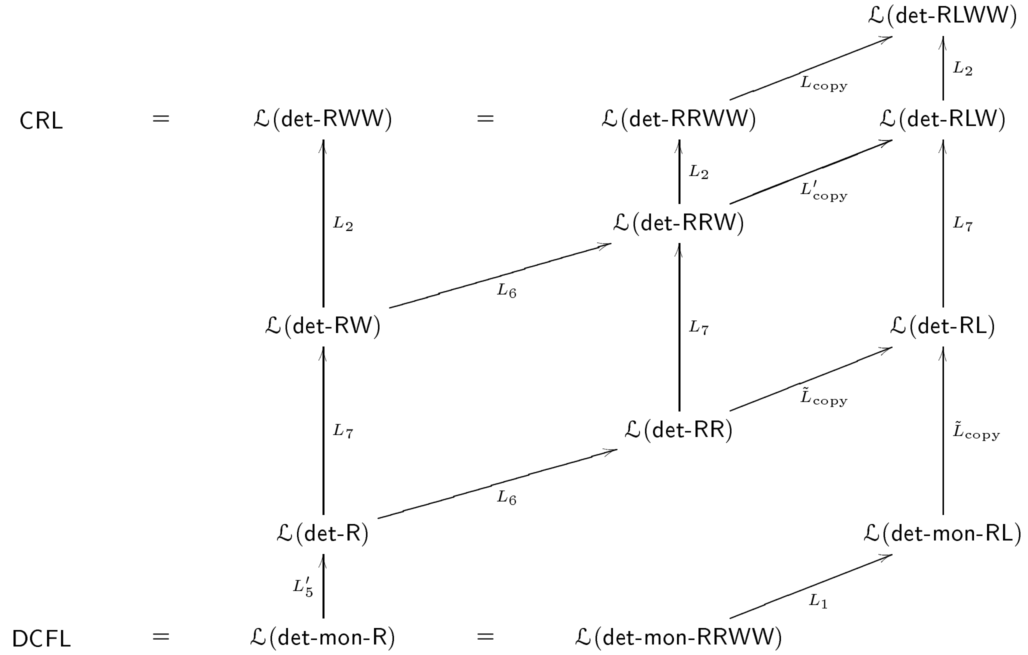

Figure 4. The taxonomy of deterministic restarting automata

As the language L_5' is not context-free, and as the classes CRL and CFL are incomparable under inclusion, we obtain the following consequence.

Corollary 4.6 *The language classes* $\mathcal{L}(\mathsf{det\text{-}R})$ *and* CFL *are incomparable with respect to inclusion.*

As $\mathcal{L}(\mathsf{det\text{-}mon\text{-}RL}) \subset \mathcal{L}(\mathsf{mon\text{-}RLWW}) = \mathsf{CFL}$, it follows in particular that $\mathcal{L}(\mathsf{det\text{-}R})$ is not contained in $\mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$. On the other hand, the language L_1 is not accepted by any deterministic RRW-automaton [38], implying that $\mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$ is not contained in $\mathcal{L}(\mathsf{det\text{-}RRW})$. However, the following additional inclusion result holds.

Theorem 4.7 $\mathcal{L}(\mathsf{det\text{-}mon\text{-}RL}) \subset \mathsf{CRL}$.

Proof. A monotone deterministic RL-automaton M can be simulated by a deterministic TPDA that is shrinking. The simulation is essentially the same as the one used to derive Theorem 4.4. Here the monotonicity is only used to guarantee that the right distance does not increase by an arbitrarily large amount from one cycle to the next. Hence, $\mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$ is contained in CRL. The language L_7 shows that this inclusion is proper. □

In order to derive still another separation result, we consider the linear language $L_{\mathrm{pal}} := \{ ww^R \mid w \in \{a,b\}^* \}$. In [23] it is shown that L_{pal} is not a Church-Rosser language. However, the following result holds.

Lemma 4.8 $L_{\mathrm{pal}} \in \mathcal{L}(\mathsf{mon\text{-}RWW})$.

Proof. Let $M_{\mathrm{pal}} = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be the RWW-automaton that is specified through $\Sigma := \{a,b\}$, $\Gamma := \Sigma \cup \Gamma' \cup \{\#\}$, where $\Gamma' := \{ [c,d] \mid c,d \in \{a,b\} \}$, $k := 4$, and the following sequence of meta-instructions, where $c,d \in \{a,b\}$:

$$\begin{array}{llll}
(1) & (¢\$, \mathsf{Accept}), & (4) & (¢ \cdot \Gamma'^*, cc \to \#), \\
(2) & (¢\#\$, \mathsf{Accept}), & (5) & (¢ \cdot \Gamma'^*, cddc \to \#), \\
(3) & (¢ \cdot \Gamma'^*, cd \to [c,d]), & (6) & (¢ \cdot \Gamma'^*, [c,d]\#dc \to \#).
\end{array}$$

Using the meta-instruction (3) repeatedly, M_{pal} encodes the prefix of the given input. Then by (4) or (5) it guesses the middle of the input, and using (6) repeatedly the prefix and the suffix are compared to each other. It follows easily that $L(M_{\mathrm{pal}}) = L_{\mathrm{pal}}$ holds, and it is obvious that M_{pal} is monotone. □

Now let M'_{pal} be the monotone RW-automaton that is obtained from M_{pal} by declaring all symbols to input symbols, and let $L'_{\mathrm{pal}} := L(M'_{\mathrm{pal}})$. Finally, let $\tilde{L}_{\mathrm{pal}} := \varphi_{4,7}(L'_{\mathrm{pal}})$, where $\varphi_{4,7}$ is the encoding of Theorem 2.14. Then $\tilde{L}_{\mathrm{pal}}$ is accepted by a monotone R-automaton. However, as the class CRL is closed under the operation of intersection with regular languages and inverse morphisms, $\tilde{L}_{\mathrm{pal}}$ is not Church-Rosser, either. As the classes CRL and CFL are incomparable [5], this yields the following incomparablily result.

Corollary 4.9
The language classes $\mathcal{L}(\mathsf{mon\text{-}R})$ *and* CRL *are incomparable with respect to inclusion.*

It is further shown in [35, 36] that $\mathcal{L}(\mathsf{det\text{-}R})$ forms a *quotient basis* for the recursively enumerable languages. Actually, this result is shown for the class of *confluent internal contextual languages*, which is a restricted version of the *internal contextual languages*, for which a corresponding result is proved in [10]. It follows that $\mathcal{L}(\mathsf{det\text{-}R})$ is already quite an expressive class of languages.

5 Weakly Monotone Restarting Automata

As seen in Theorem 4.4, the language class GCSL is contained in $\mathcal{L}(\mathsf{RWW})$. On the other hand, by Corollary 2.18 there exist R-automata that accept languages which are not growing context-sensitive. Hence, we have the following result.

Corollary 5.1 *The class* GCSL *is properly contained in* $\mathcal{L}(\mathsf{RWW})$.

In addition, it has been shown that $\mathcal{L}(\mathsf{RWW})$ contains some NP-complete languages [24]. By combining this result with the reductions of Corollaries 2.12 and 2.15 we obtain the following consequence, where $\mathsf{LOG}(\mathcal{L})$ denotes the closure of a language class $\mathcal{L}$ under log-space reductions.

Theorem 5.2 $\mathcal{L}(R)$ *contains* NP*-complete languages. In particular,* $\mathsf{LOG}(\mathcal{L}(\mathsf{R})) = \mathrm{NP}$.

In order to obtain a characterization of the class GCSL in terms of restarting automata, a weaker notion of monotonicity has been introduced in [24]. Let M be an RLWW-automaton, and let $c \geq 0$ be an integer. We say that a sequence of cycles $C_1, C_2, \ldots, C_n$ of M is *weakly c-monotone* if $D_r(C_{i+1}) \leq D_r(C_i) + c$ holds for all $i = 1, \ldots, n-1$. A computation of M is called *weakly c-monotone* if the corresponding sequence of cycles is weakly c-monotone. Observe that the tail of the computation is not taken into account. Finally, the RLWW-automaton M is called *weakly c-monotone* if, for each restarting configuration $q_0¢w\$$ of M, each computation of M starting with $q_0¢w\$$ is weakly c-monotone. Note that here we do not only consider computations that start with an initial configuration, but that we explicitly consider all computations that start with an arbitrary restarting configuration. The RLWW-automaton M is called *weakly monotone* if it is weakly c-monotone for some integer constant $c \geq 0$. The prefix wmon- is used to denote the classes of weakly monotone restarting automata.

Let M be a deterministic RRWW-automaton, and let $C_1, C_2, \ldots, C_n$ be the sequence of cycles of a computation of M. If C_i contains the Rewrite step $¢xquy\$ \to ¢xvq'y\$$, where q, q' are states, then $D_r(C_i) = |y|+1$. In the next cycle, C_{i+1}, M cannot perform a Rewrite step before it sees at least the first symbol of $vy\$$ in its read/write window, as M is deterministic. Thus, $D_r(C_{i+1}) \leq |vy\$| - 1 = D_r(C_i) + |v| - 1$. By taking the constant $c := \max(\{\, |v| - 1 \mid u \to v \text{ is a Rewrite step of } M \,\} \cup \{0\})$, we see that M is necessarily weakly c-monotone. Thus, each deterministic RRWW-automata is necessarily weakly monotone. This, however, is not true for deterministic RLWW-automata in general. For example, the deterministic RLWW-automaton for the language L_{copy} mentioned in Section 4 is not weakly monotone. Further, we will see that for the various types of nondeterministic restarting automata, weak monotonicity makes a big difference. However, it has been observed in [31] that an RLWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is weakly monotone if and only if it is weakly c-monotone for some constant $c < |Q|^2 \cdot |\Gamma|^k + 2k$. Also the following result has been established.

Theorem 5.3 [31] *It is decidable whether a given* RLWW*-automaton M is weakly monotone. In the affirmative the smallest integer c can be determined for which M is weakly c-monotone.*

For each language $L \in$ GCSL, there exists a length-reducing TPDA P such that $L = L(P)$ (Theorem 4.3). This TPDA can be simulated by an RWW-automaton M that encodes a configuration $\perp uqv \perp$ of P by the tape contents $¢\hat{u}qv\$$ (see the discussion following Theorem 4.3). The automaton M simply moves its read/write window from left to right across its tape until it discovers the left-hand side of a transition of P, which it then simulates by a Rewrite step. As this Rewrite step includes the unique state symbol of P contained on the tape, we see that M is weakly monotone.

On the other hand, the simulation of a deterministic RRWW-automaton by a shrinking DTPDA carries over to nondeterministic RRWW-automata that are weakly monotone. Just notice the following two facts, where M denotes a weakly monotone RRWW-automaton:

(i) While performing MVR-steps M behaves like a finite-state acceptor. Hence, the first part of each cycle can be simulated deterministically. Nondeterminism comes in as soon as a Rewrite step is enabled.

(ii) As pointed out in Section 4, with each string $w \in \Gamma^*$, two subsets $Q_+(w)$ and $Q_{\text{rs}}(w)$ can be associated. For the deterministic case these sets are necessarily disjoint for each string w. If M is nondeterministic, then this is not true anymore. However, if $Q_+(w)$ is nonempty, then M will accept, and so the simulation simply accepts, and if $Q_+(w)$ is empty, but $Q_{\text{rs}}(w)$ is nonempty, then a Restart step must be simulated.

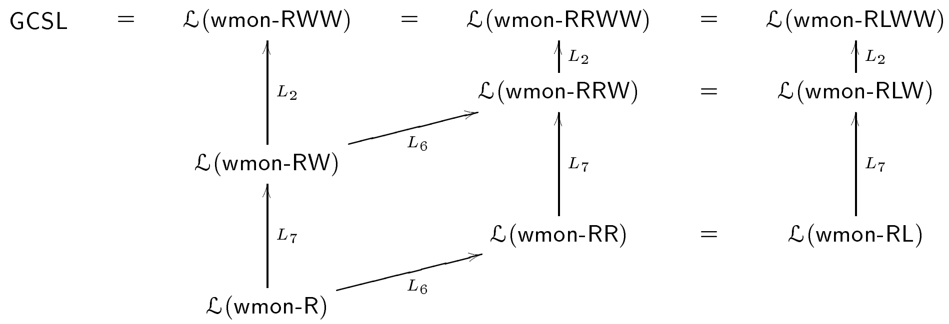

Figure 5. The taxonomy of weakly monotone restarting automata

Because of Theorem 4.3 (and Theorem 2.6), this yields the following characterization.

Theorem 5.4 $\mathsf{GCSL} = \mathcal{L}(\mathsf{wmon\text{-}RWW}) = \mathcal{L}(\mathsf{wmon\text{-}RRWW}) = \mathcal{L}(\mathsf{wmon\text{-}RLWW})$.

For each type $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$, the classes $\mathcal{L}(\mathsf{det\text{-}X})$ and $\mathcal{L}(\mathsf{mon\text{-}X})$ are obviously contained in the class $\mathcal{L}(\mathsf{wmon\text{-}X})$. By Corollaries 4.6 and 4.9 all these inclusions are proper, and no class of the form $\mathcal{L}(\mathsf{wmon\text{-}X})$ is contained in any class of the form $\mathcal{L}(\mathsf{det\text{-}Y})$ or $\mathcal{L}(\mathsf{mon\text{-}Y})$, where $\mathsf{Y} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$. On the other hand, the language class $\mathcal{L}(\mathsf{det\text{-}RL})$ is not contained in GCSL because of the language $\tilde{L}_{\text{copy}}$ (see Section 4). Figure 5 describes the relationships between the various classes of weakly monotone restarting automata.

6 On General RWW- and RRWW-Automata

Next we discuss the classes of languages that are characterized by the various types of unrestricted nondeterministic restarting automata.

The languages L_6 and L_7 show that among the classes $\mathcal{L}(\mathsf{R})$, $\mathcal{L}(\mathsf{RR})$, $\mathcal{L}(\mathsf{RW})$, and $\mathcal{L}(\mathsf{RRW})$ only the trivial inclusions hold, and that they are proper. The language L_2 is context-free and Church-Rosser (Lemma 4.5), but not in $\mathcal{L}(\mathsf{RRW})$, and the language $L_{6,2} \cup L_{6,3}$ is not context-free, but it belongs to $\mathcal{L}(\mathsf{R})$ [20]. This shows that CFL is incomparable to all the above classes, and that the inclusion of $\mathcal{L}(\mathsf{R(R)W})$ in $\mathcal{L}(\mathsf{R(R)WW})$ is proper. Thus, for the classes of languages characterized by the various types of nondeterministic restarting automata we have the taxonomy presented in Figure 6.

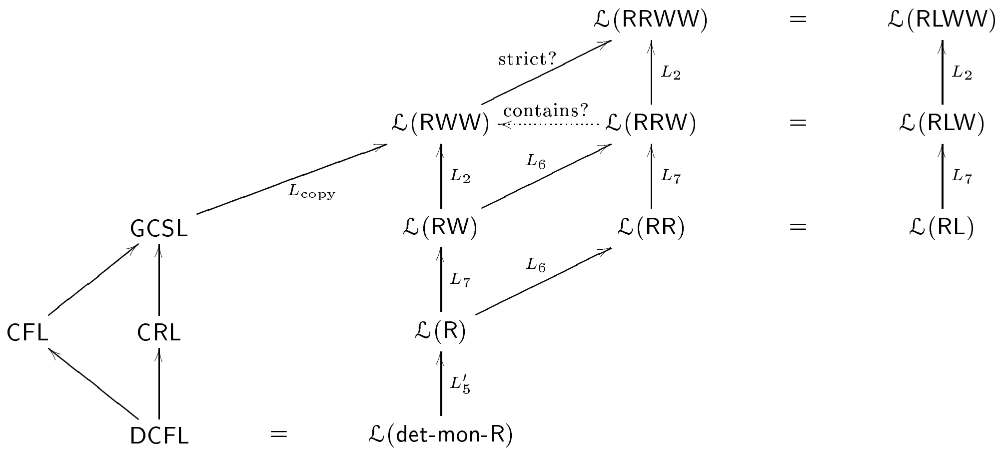

Figure 6. The taxonomy of nondeterministic restarting automata

In addition, we conclude from Theorem 2.10 that the language classes $\mathcal{L}(\mathsf{RW})$ and $\mathcal{L}(\mathsf{RRW})$ are not closed under the operation of intersection with regular languages. Further, from Corollary 2.18 we know that the class $\mathcal{L}(\mathsf{R})$ is not even contained in GCSL. This yields the following consequence.

Corollary 6.1
$\mathcal{L}(\mathsf{R})$ *is incomparable to the language classes* CFL, CRL, *and* GCSL *with respect to inclusion.*

It is still open whether the inclusion $\mathcal{L}(\mathsf{RWW}) \subseteq \mathcal{L}(\mathsf{RRWW})$ is strict, and whether $\mathcal{L}(\mathsf{RRW})$ is contained in $\mathcal{L}(\mathsf{RWW})$. If $\mathcal{L}(\mathsf{RRW}) \subseteq \mathcal{L}(\mathsf{RWW})$, then by Theorem 2.10 we have $\mathcal{L}(\mathsf{RWW}) = \mathcal{L}(\mathsf{RRWW})$, and if $\mathcal{L}(\mathsf{RRW}) \not\subseteq \mathcal{L}(\mathsf{RWW})$, then the inclusion $\mathcal{L}(\mathsf{RWW}) \subseteq \mathcal{L}(\mathsf{RRWW})$ is obviously strict, that is, these two open problems are closely related. Also it is open whether $\mathcal{L}(\mathsf{RR}) \subset \mathcal{L}(\mathsf{RWW})$ holds. Further, the exact relationship of the language classes $\mathcal{L}(\mathsf{det\text{-}RL(W)(W)})$ to the various nondeterministic classes is still open. In particular, it is currently not known whether or not the class CFL is contained in $\mathcal{L}(\mathsf{det\text{-}RLWW})$.

Finally we turn to the question of the expressive power of the RWW-, RRWW-, and RLWW-automata. As seen above these automata are more powerful than the class GCSL, but they can only accept certain context-sensitive languages (Proposition 2.1). Here we describe in short some other language classes between GCSL and CSL and restate a number of open problems.

Let $G = (N, T, S, P)$ be a grammar. By $\hat{L}(G)$ we denote the set of sentential forms that are derivable from the start symbol S in G. For each word $w \in \hat{L}(G)$, let $t_G(w)$ denote the length of a shortest derivation of w from S in G. The *time function* T_G of G is then the following partial function $T_G : \mathbb{N} \to \mathbb{N}$ [2]:

$$T_G(n) := \begin{cases} \max\{\, t_G(w) \mid w \in \hat{L}(G) \cap (N \cup T)^n \,\}, & \text{if } \hat{L}(G) \cap (N \cup T)^n \neq \emptyset, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

By $\mathsf{CSL}_{\mathsf{lin}}$ we denote the class of all *linear-time bounded context-sensitive languages.* A language L belongs to this class, if there exists a monotone grammar $G = (N, T, S, P)$ that generates L and a constant $c \geq 1$ such that, for all $n \in \mathbb{N}$, if $T_G(n)$ is defined, then $T_G(n) \leq c \cdot n$ [1, 2]. Finally by $\mathsf{CSL}_{\mathsf{quad}}$ we denote the class of all *quadratical-time bounded context-sensitive languages.* A language L belongs to this class, if there exists a monotone grammar $G = (N, T, S, P)$ that generates L and a constant $c \geq 1$ such that, for all $n \in \mathbb{N}$, if $T_G(n)$ is defined, then $T_G(n) \leq c \cdot n^2$.

Proposition 6.2 ([1] Theorem 2.3.3) $\mathsf{CSL}_{\mathsf{quad}} = \mathsf{NTIMESPACE}_1(n^2, n)$.

Here $\mathsf{NTIMESPACE}_1(n^2, n)$ denotes the class of languages that are accepted by one-tape nondeterministic Turing machines in quadratic time with linear space. It is easily seen that an RLWW-automaton can be simulated by such a Turing machine, that is, $\mathcal{L}(\mathsf{RLWW}) \subseteq \mathsf{CSL}_{\mathsf{quad}}$. Finally by Q we denote the class of *quasi-realtime languages.* This is the class of languages that are accepted by nondeterministic multitape Turing machines in realtime, that is, $\mathsf{Q} = \mathsf{NTIME}(\mathrm{n})$.

Proposition 6.3 [1, 4] $\mathsf{GCSL} \subset \mathsf{CSL}_{\mathsf{lin}} \subset \mathsf{Q} = \mathsf{NTIME}(\mathrm{lin})$.

For proving that GCSL is only a proper subset of $\mathsf{CSL}_{\mathsf{lin}}$, the language

$$L_{\mathrm{COPY(n^2)}} = \{\, ww\#^{4|w|^2-2|w|} \mid w \in \{a, b\}^* \,\}$$

is considered in [4]. This language is a member of $\mathsf{CSL}_{\mathsf{lin}}$ ([4] Satz VIII.4), but it is not growing context-sensitive ([4] Satz VII.4 and Satz VIII.5).

A nondeterministic multitape Turing machine M that runs in realtime only uses linear space. Hence, it can be simulated by a nondeterministic one-tape Turing machine that is linear space-bounded and quadratically time-bounded. On the other hand, M can be simulated by a deterministic (multi- or one-tape) Turing machine that runs in exponential time, but that only uses linear space. Thus, we obtain the following inclusions, where $\mathsf{DCSL} = \mathsf{DSPACE}(n)$. It is an open problem which (if any) of these inclusions are proper.

Proposition 6.4 $\mathsf{Q} \subseteq \mathsf{CSL}_{\mathsf{quad}} \subseteq \mathsf{CSL}$ *and* $\mathsf{Q} \subseteq \mathsf{DCSL}$.

Finally, we turn to a generalization of the TPDA considered in [41]. An *alternating* TPDA, ATPDA for short, is a TPDA P for which a subset U of the set of states Q is designated as the set of *universal states.* The states in the difference set $Q \setminus U$ are accordingly called *existential states.* An ATPDA is called *shrinking*, if the underlying TPDA is shrinking. By sATPDA we denote the class of all shrinking ATPDAs.

As each sTPDA is a sATPDA without universal states, we see that $\mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{sATPDA})$ holds. On the other hand, the Gladkij language $L_{\mathrm{GL}} := \{\, w\#w^R\#w \mid w \in \{a, b\}^* \}$, which is not growing context-sensitive [4], is accepted by an sATPDA, which implies that this inclusion is proper. Actually, the following results on $\mathcal{L}(\mathsf{sATPDA})$ are known.

Proposition 6.5 [41]

(a) $\mathsf{GCSL} \subset \mathcal{L}(\mathsf{sATPDA}) \subseteq \mathsf{DCSL}$.

(b) $\mathsf{LOG}(\mathcal{L}(\mathsf{sATPDA})) = \mathsf{PSPACE}$.

The diagram in Figure 7 summarizes the language classes considered here and the corresponding inclusion results.

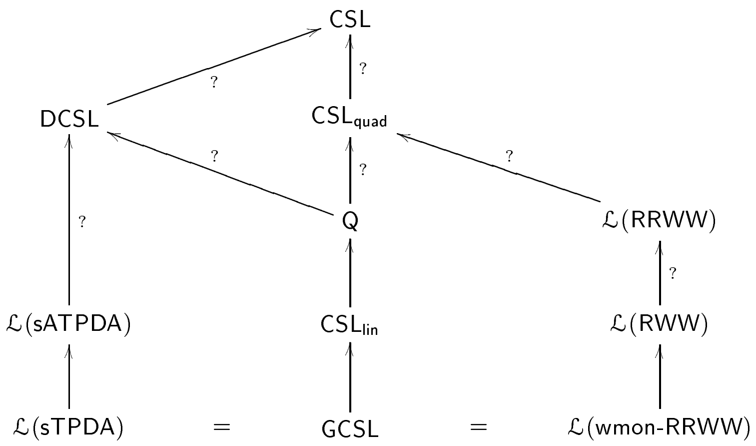

Figure 7. An unmarked arrow indicates that the inclusion is proper, while a question mark indicates that it is an open problem whether the corresponding inclusion is proper. For those classes that are not connected via directed paths in the diagram it is open whether any inclusions hold.

7 Left-Monotone Restarting Automata

The notion of monotonicity considered in Section 3 is based on the requirement that, within a computation of a restarting automaton, the distance between the place where a Rewrite step is performed and the *right end* of the tape must not increase from one cycle to the next. Here we study the seemingly symmetric notion of *left-monotonicity*.

We say that a *sequence of cycles* $Sq = (C_1, C_2, \cdots, C_n)$ of an RLWW-automaton M is *left-monotone* if $D_l(C_1) \geq D_l(C_2) \geq \cdots \geq D_l(C_n)$. A computation of M is *left-monotone*, if the corresponding sequence of cycles is left-monotone. As before the tail of the computation does not play any role here. Finally, the RLWW-automaton M is called *left-monotone* if each of its computations that starts from an initial configuration is left-monotone. The prefix left-mon- is used to denote the various classes of left-monotone restarting automata.

First we study the influence that left-monotonicity has on the various types of deterministic restarting automata. We begin with a simple example.

Example 7.1 Let $M := (Q, \Sigma, \Gamma, \text{¢}, \$, q_0, 4, \delta)$ be the deterministic RWW-automaton that is defined by $\Sigma := \{a, b, c, d\}$ and $\Gamma := \Sigma \cup \{C, D\}$, where δ is given through the following meta-instructions:

$$
\begin{array}{llll}
(1) & (\text{¢} \cdot a^+ \cdot b^*, \quad bbc\$ \to C\$), & (6) & (\text{¢} \cdot a^+ \cdot b^*, \quad bbd\$ \to D\$),\\
(2) & (\text{¢} \cdot a^+ \cdot b^*, \quad bbC \to CC), & (7) & (\text{¢} \cdot a^+ \cdot b^*, \quad bbD \to DD),\\
(3) & (\text{¢} \cdot a^+, \quad abC \to C), & (8) & (\text{¢} \cdot a^+, \quad aDD \to D),\\
(4) & (\text{¢} \cdot a^+, \quad aaCC \to C), & (9) & (\text{¢} \cdot aD \cdot \$, \quad \mathsf{Accept}),\\
(5) & (\text{¢} \cdot aaC \cdot \$, \quad \mathsf{Accept}), & (10) & (\text{¢} \cdot x \cdot \$, \quad \mathsf{Accept}) \quad \text{for all } x \in \{c, d, abc\}.
\end{array}
$$

It is easily seen that M accepts the language L_1 of Example 2.2, and that M is left-monotone.

Observe that the deterministic RL-automaton presented in Example 2.2 is not only monotone, but that it is also left-monotone, showing that $L_1 \in \mathcal{L}(\text{det-left-mon-RL})$ holds. As L_1 is not deterministic context-free, the above example shows further that $\mathcal{L}(\text{det-mon-RWW})$ and $\mathcal{L}(\text{det-left-mon-RWW})$ do not coincide (see Theorem 3.2). Further, we have the following result, which represents another difference between the behaviour of the monotone deterministic classes and the left-monotone deterministic classes.

Theorem 7.2 [40] $\mathcal{L}(\text{det-left-mon-RWW}) = \mathcal{L}(\text{det-left-mon-RRWW}) = \mathcal{L}(\text{det-left-mon-RLWW})$.

It follows that the class $\mathcal{L}(\text{det-left-mon-RWW})$ contains the reversal of every deterministic context-free language. In fact, it also contains some additional languages, as it contains the reversal of every language from the class $\mathcal{L}(\text{det-mon-RLWW})$, which properly contains the class DCFL.

For separating the class $\mathcal{L}$(det-left-mon-RWW) from those classes of languages that are accepted by the more restricted types of left-monotone deterministic restarting automata, we introduce some additional example languages:

$$\begin{array}{lcll}
L_8 & := & L_{8,1} \cup L_{8,2} \cup L_{8,3} \cup L_{8,4}, & \\
L_{8,1} & := & \{\, a^m a^n (bc)^n f a^m & \mid \ m,n > 0\,\}, \\
L_{8,2} & := & \{\, a^m a^n (bc)^i b^j a^m & \mid \ m,n,j > 0,\ i \geq 0,\ n = i+j\,\}, \\
L_{8,3} & := & \{\, a^n (bc)^i c^j & \mid \ n,j > 0,\ i \geq 0,\ n = 2(i+j)\}, \\
L_{8,4} & := & \{\, a^m (bc)^n f a^k f & \mid \ m,n,k > 0\,\}, \\
L_9 & := & \{\, a^m b^m a^n b^n \mid m,n > 0\,\}, & \\
L_{10} & := & \{\, a_5^n a_4^n a_3^m a_2^m a_1^p \mid n,m,p > 0\,\} \cup \{\, a_5^n a_4^l a_3^m a_2^p a_1^p \mid n \neq l, n,l,m,p > 0\,\}. &
\end{array}$$

The language L_8 is accepted by a left-monotone deterministic RR-automaton, but it cannot be accepted by any deterministic RW-automaton. On the other hand, the language L_9 is accepted by a left-monotone deterministic RWW-automaton, but not by any left-monotone deterministic RRW-automaton [45]. Based on the reduction underlying Corollary 2.12 we obtain a language L_9' from L_9 such that $L_9' \in \mathcal{L}$(det-left-mon-RW) $\smallsetminus$ $\mathcal{L}$(det-left-mon-RR). Thus, we have the taxonomy described by Figure 8.

On the other hand, it is shown in [40] that $\mathcal{L}$(det-left-mon-RWW) does not include the deterministic context-free language L_{10}, implying the following non-inclusion result.

Corollary 7.3 DCFL $\not\subseteq$ $\mathcal{L}$(det-left-mon-RLWW).

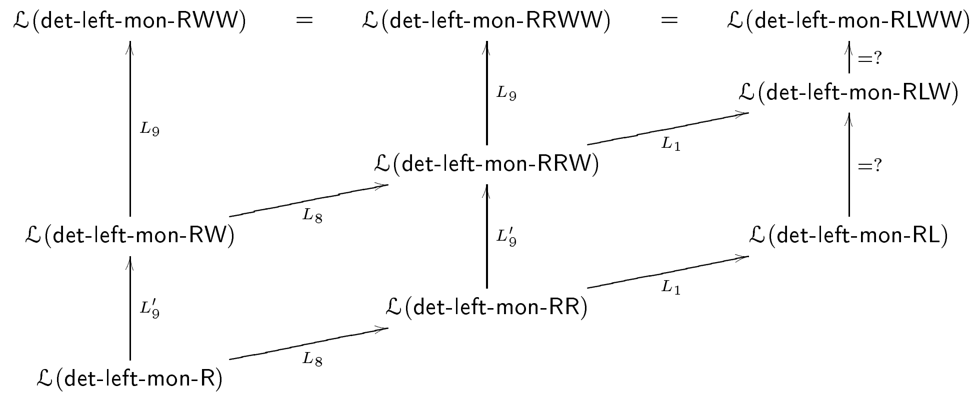

Figure 8. The taxonomy of left-monotone deterministic restarting automata

We see that the classes of languages specified through the various types of left-monotone deterministic restarting automata show a completely different structure than those classes that are defined through the various types of monotone deterministic restarting automata. By the way, from Theorems 4.4 and 7.2 we obtain the inclusion $\mathcal{L}$(det-left-mon-RLWW) $\subset$ CRL, which is proper, because $\mathcal{L}$(det-left-mon-RLWW) contains only context-free languages, as we will see below. In addition, we conjecture that the inclusions

$$\mathcal{L}(\text{det-left-mon-RL}) \subseteq \mathcal{L}(\text{det-left-mon-RLW}) \subseteq \mathcal{L}(\text{det-left-mon-RLWW})$$

are in fact equalities, thus extending Theorem 7.2.

Now we turn to the nondeterministic types of left-monotone restarting automata. In symmetry to the proof of Theorem 3.1 a left-monotone RWW-automaton M can be constructed from a context-free grammar G in Chomsky normal form such that, given an input word w, M tries to construct a leftmost derivation of w in G in reverse order. Conversely, assume that L is accepted by a left-monotone RLWW-automaton. Then its reversal L^R is accepted by an RLWW-automaton that is monotone. Hence, by Theorem 3.1, L^R is context-free, which in turn implies that L is context-free. Thus, we have the following analogue to the aforementioned theorem.

Theorem 7.4 $\mathcal{L}$(left-mon-RLWW) = $\mathcal{L}$(left-mon-RRWW) = $\mathcal{L}$(left-mon-RWW) = CFL.

We take $L_3' := \{\, a^n \cdot f \cdot b^n, a^n \cdot ee \cdot b^n \mid n \geq 0 \,\} \cup \{\, a^n \cdot g \cdot b^m, a^n \cdot ee \cdot b^m \mid m > 2n \geq 0 \,\}$, which is a slight variant of the language L_3. Then it can be shown that L_3' is accepted by a left-monotone RW-automaton, but just as L_3, it is not accepted by any RR-automaton. Further, the language L_1 is not accepted by any RW-automaton, although it is easily seen to be accepted by a left-monotone RR-automaton. Finally, we noticed before that the context-free language L_2 is not accepted by any RRW-automaton. Thus, we have the taxonomy given in Figure 9.

It follows in particular that the class $\mathcal{L}(\mathsf{det\text{-}left\text{-}mon\text{-}RLWW})$ is contained in the intersection of the language classes CFL and CRL. Obviously, this is a proper inclusion, as according to Corollary 7.3, $\mathcal{L}(\mathsf{det\text{-}left\text{-}mon\text{-}RLWW})$ does not even contain all deterministic context-free languages. However, the relationships of the language classes defined by the various types of left-monotone (deterministic) restarting automata to each other and to the language classes defined by the other types of restarting automata have not yet been clarified completely.

$$
\begin{tikzcd}
\mathsf{CFL} \arrow[r, phantom, "="] & \mathcal{L}(\text{left-mon-RWW}) \arrow[r, phantom, "="] & \mathcal{L}(\text{left-mon-RRWW}) \arrow[r, phantom, "="] & \mathcal{L}(\text{left-mon-RLWW}) \\
& & \mathcal{L}(\text{left-mon-RRW}) \arrow[u, "L_2"'] \arrow[r, phantom, "="] & \mathcal{L}(\text{left-mon-RLW}) \arrow[u, "L_2"'] \\
& \mathcal{L}(\text{left-mon-RW}) \arrow[uu, "L_2"'] \arrow[ur, "L_1"'] & & \\
& & \mathcal{L}(\text{left-mon-RR}) \arrow[uu, "L_3'"'] \arrow[r, phantom, "="] & \mathcal{L}(\text{left-mon-RL}) \arrow[uu, "L_3'"'] \\
& \mathcal{L}(\text{left-mon-R}) \arrow[uu, "L_3'"'] \arrow[ur, "L_1"'] & &
\end{tikzcd}
$$

Figure 9. The taxonomy of left-monotone nondeterministic restarting automata

8 Further Developments and Open Problems

The variants of the restarting automaton considered up to here are obtained through the following types of restrictions:

- deterministic versus nondeterministic automata,
- automata with or without auxiliary symbols,
- monotone, weakly monotone, left-monotone or non-monotone automata.

Below we discuss in short some results that are obtained by either looking at more fine-grained versions of the above restrictions, or by different variations of the underlying basic model.

8.1 Look-ahead Hierarchies

One of the parameters describing a restarting automaton is the size of its read/write window. The influence of this parameter on the expressive power of the various types of restarting automata without auxiliary symbols has been studied in [30]. The results obtained show that with the size of the read/write window also the expressive power of these types of restarting automata increases. It remains open, however, whether corresponding results also hold for the various types of restarting automata with auxiliary symbols.

8.2 The Degree of Weak Monotonicity

Another parameter that allows a quantitative analysis is the degree of weak monotonicity. As defined in Section 5 a restarting automaton is weakly monotone if it is weakly c-monotone for some integer $c \geq 0$. For each language L that is accepted by some weakly monotone restarting automaton of a certain type X, we can associate the smallest integer $c(L)$ such that L is accepted by a weakly $c(L)$-monotone restarting automaton of type X. This integer can be interpreted as the *degree of weak monotonicity* of the language L with respect to the type X.

It is shown in [31] that the degree of weak monotonicity yields infinite hierarchies for the deterministic as well as for the nondeterministic R-, RR-, RW-, and RRW-automata. Again, it is still open whether corresponding results hold for the various types of restarting automata with auxiliary symbols.

8.3 Degrees of Non-Monotonicity

Another generalization of monotonicity is discussed in [44]. Let $j \in \mathbb{N}_+$ be a constant, and let M be a restarting automaton. A sequence of cycles $C_1, C_2, \ldots, C_n$ of M is called *j-monotone* if there is a partition of this sequence into at most j (scattered) subsequences such that each of these subsequences is monotone. Obviously, $C_1, C_2, \ldots, C_n$ is not j-monotone if and only if there exist indices $1 \leq i_1 < i_2 < \cdots < i_{j+1} \leq n$ such that $D_r(C_{i_1}) < D_r(C_{i_2}) < \cdots < D_r(C_{i_{j+1}})$ holds. A computation of M is *j-monotone* if the corresponding sequence of cycles is j-monotone, and M is called *j-monotone* if all its computations that start from an initial configuration are j-monotone. It is shown in [44] that by increasing the value of the parameter j, the expressive power of RR- and RRW-automata is increased, and the same is true for R- and RW-automata [45]. Also an analogous generalization of the notion of left-monotonicity is studied in [45]. However, in all these cases it remains open whether infinite hierarchies exist for the various types of restarting automata with auxiliary symbols.

8.4 Generalizations of Restarting Automata

Further generalizations of the basic model of the restarting automaton are being considered. Here we only mention two of them.

The first one deals with the number of Rewrite steps that a restarting automaton may perform within a single cycle. Motivated by the 'analysis by reduction' each model of restarting automaton considered so far performs exactly one Rewrite step in each cycle of each computation. However, as a restarting automaton can be seen as a special type of linear-bounded automaton, it is only natural to also consider variants that perform up to c Rewrite steps in each cycle for some constant $c > 1$. How will an increase in the number c influence the expressive power of the various models of restarting automata? A further generalization might allow $f(n)$ Rewrite steps per cycle, where f is a sublinear function and n denotes the length of the input.

Length-reducing TPDAs are as powerful as shrinking TPDAs (Theorem 4.3). As the Rewrite transitions of a restarting automaton are length-reducing, one might also consider restarting automata for which the Rewrite transitions are only required to be weight-reducing with respect to some weight function. For the deterministic case it can be shown that these automata still characterize the class CRL (in the presence of auxiliary symbols), but it is currently open whether or not this variant increases the expressive power for the nondeterministic types of restarting automata.

Acknowledgement. The author wants to thank Tomasz Jurdziński, Hartmut Messerschmidt, František Mráz, Martin Plátek, and Heiko Stamer for many fruitful discussions regarding restarting automata and related topics.

References

[1] R.V. Book. Grammars with Time Functions. *PhD dissertation*, Harvard University, Cambridge, Massachusetts, 1969.

[2] R.V. Book. Time-bounded grammars and their languages. *J. Computer System Sciences* 5 (1971) 397–429.

[3] R.V. Book and F. Otto. *String-Rewriting Systems.* Springer, New York, 1993.

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

[5] G. Buntrock and F. Otto. Growing context-sensitive languages and Church-Rosser languages. *Information and Computation* 141 (1998) 1–36.

[6] M.P. Chytil, M. Plátek, and J. Vogel. A note on the Chomsky hierarchy. *Bulletin of the EATCS* 27 (1985) 23–30.

[7] E. Dahlhaus and M. Warmuth. Membership for growing context-sensitive grammars is polynomial. *J. Computer System Sciences* 33 (1986) 456–472.

[8] A. Dikovsky and L. Modina. Dependencies on the other side of the curtain. *Traitement Automatique des Langues* 41 (2000) 79–111.

[9] A. Ehrenfeucht, G. Păun, and G. Rozenberg. Contextual grammars and formal languages. In: G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Vol. 2. Springer, Berlin, 1997, 237–293.

[10] A. Ehrenfeucht, G. Păun, and G. Rozenberg. On representing recursively enumerable languages by internal contextual languages. *Theoretical Computer Science* 205 (1998) 61–83.

[11] A.W. Gladkij. On the complexity of derivations for context-sensitive grammars. *Algebra i Logika* 3 (1964) 29–44 (In Russian).

[12] R. Gramatovici and C. Martín-Vide. 1-contextual grammars with sorted dependencies. In: F. Spoto, G. Scollo, and A. Nijholt (eds.), *3rd AMAST Workshop on Language Processing, Proc.*, Universiteit Twente, Enschede, 2003, 99-109.

[13] S. Greibach. A note on pushdown store automata and regular systems. *Proc. American Math. Soc.* 18 (1967) 263–268.

[14] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, M.A., 1979.

[15] P. Jančar, F. Mráz, and M. Plátek. A taxonomy of forgetting automata. In: A.M. Borzyszkowski and ST. Sokolowski (eds.), *MFCS'93, Proc.*, *Lecture Notes in Computer Science 711*, Springer, Berlin, 1993, 527–536.

[16] P. Jančar, F. Mráz, and M. Plátek. Forgetting automata and context-free languages. *Acta Informatica* 33 (1996) 409–420.

[17] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In: H. Reichel (ed.), *FCT'95, Proc.*, *Lecture Notes in Computer Science 965*, Springer, Berlin, 1995, 283–292.

[18] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On restarting automata with rewriting. In: G. Păun and A. Salomaa (eds.), *New Trends in Formal Languages*, *Lecture Notes in Computer Science 1218*, Springer, Berlin, 1997, 119–136.

[19] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Monotonic rewriting automata with a restart operation. In: F. Plášil and K.G. Jeffery (eds.), *SOFSEM'97, Proc.*, *Lecture Notes in Computer Science 1338*, Springer, Berlin, 1997, 505–512.

[20] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Different types of monotonicity for restarting automata. In: V. Arvind and R. Ramanujam (eds.), *FSTTCS'98, Proc.*, *Lecture Notes in Computer Science 1530*, Springer, Berlin, 1998, 343–354.

[21] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On monotonic automata with a restart operation. *J. Automata, Languages and Combinatorics* 4 (1999) 287–311.

[22] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Monotonicity of restarting automata. *J. Automata, Languages and Combinatorics*, to appear.

[23] T. Jurdziński and K. Loryś. Church-Rosser languages vs. UCFL. In: P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo (eds.), *ICALP'02, Proc.*, *Lecture Notes in Computer Science 2380*, Springer, Berlin, 2002, 147–158.

[24] T. Jurdziński, K. Loryś, G. Niemann, and F. Otto. Some results on RRW- and RRWW-automata and their relationship to the class of growing context-sensitive languages. *Mathematische Schriften Kassel*, no. 14/01, Fachbereich Mathematik/Informatik, Universität Kassel, 2001. To appear in *J. Automata, Languages and Combinatorics* in revised form.

[25] T. Jurdziński, F. Otto, F. Mráz, and M. Plátek. On the complexity of 2-monotone restarting automata. *Mathematische Schriften Kassel*, no. 4/04, Fachbereich Mathematik/Informatik, Universität Kassel, 2004.

[26] C. Lautemann. One pushdown and a small tape. In: K.W. Wagner (ed.), *Dirk Siefkes zum 50. Geburtstag*, Technische Universität Berlin and Universität Augsburg, 1988, 42–47.

[27] S. Marcus. Contextual grammars and natural languages. In: G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Vol. 2, Springer, Berlin, 1997, 215–235.

[28] R. McNaughton. An insertion into the Chomsky hierarchy? In: J. Karhumäki, H. Maurer, G. Păun, and G. Rozenberg (eds.), *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, Springer, Berlin, 1999, 204–212.

[29] R. McNaughton, P. Narendran, and F. Otto. Church-Rosser Thue systems and formal languages. *J. Assoc. Comput. Mach.* 35 (1988) 324–344.

[30] F. Mráz. Lookahead hierarchies of restarting automata. *J. Automata, Languages and Combinatorics* 6 (2001) 493–506.

[31] F. Mráz and F. Otto. Hierarchies of weakly monotone restarting automata. *Mathematische Schriften Kassel*, no. 8/03, Fachbereich Mathematik/Informatik, Universität Kassel, 2003. To appear in *RAIRO - Theoretical Informatics and Applications* in revised form.

[32] P. Narendran. Church-Rosser and related Thue systems. *PhD thesis*, Rensselaer Polytechnic Institute, Troy, New York, 1984.

[33] G. Niemann. Church-Rosser languages and related classes. *Doctoral Dissertation*, Universität Kassel, 2002.

[34] G. Niemann and F. Otto. The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages. In: M. Nivat (ed.), *FoSSaCS'98, Proc.*, *Lecture Notes in Computer Science 1378*, Springer, Berlin, 1998, 243–257

[35] G. Niemann and F. Otto. Confluent internal contextual languages. In: C. Martin-Víde and G. Păun (eds.), *Recent Topics in Mathematical and Computational Linguistics*, The Publishing House of the Romanian Academy, Bucharest, 2000, 234–244.

[36] G. Niemann and F. Otto. Restarting automata, Church-Rosser languages, and representations of r.e. languages. In: G. Rozenberg and W. Thomas (eds.), *DLT'99, Proc.*, World Scientific, Singapore, 2000, 103–114.

[37] G. Niemann and F. Otto. On the power of RRWW-automata. In: M. Ito, G. Păun, and S. Yu (eds.), *Words, Semigroups, and Transductions. Essays in Honour of Gabriel Thierrin, On the Occasion of His 80th Birthday*, World Scientific, Singapore, 2001, 341–355.

[38] G. Niemann and F. Otto. Further results on restarting automata. In: M. Ito and T. Imaoka (eds.), *Words, Languages and Combinatorics III, Proc.*, World Scientific, Singapore, 2003, 353–369.

[39] K. Oliva, P. Květoň, and R. Ondruška. The computational complexity of rule-based part-of-speech tagging. *TSD 2003, Proc.*, *Lecture Notes in Artificial Intelligence*, Springer, Berlin, to appear.

[40] F. Otto and T. Jurdzinski. On left-monotone restarting automata. *Mathematische Schriften Kassel*, no. 17/03, Fachbereich Mathematik/Informatik, Universität Kassel, 2003.

[41] F. Otto and E. Moriya. Shrinking alternating two-pushdown automata. *IEICE Transactions on Information and Systems* E87-D (2004) 959–966.

[42] M. Plátek. Two-way restarting automata and j-monotonicity. In: L. Pacholski and P. Ružička (eds.), *SOFSEM'01, Proc.*, *Lecture Notes in Computer Science 2234*, Springer, Berlin, 2001, 316–325.

[43] M. Plátek, M. Lopatková, and K. Oliva. Restarting automata: motivations and applications. In: M. Holzer (ed.), *Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten', Proc.*, Institut für Informatik, Technische Universität München, 2003, 90-96.

[44] M. Plátek and F. Mráz. Degrees of (non)monotonicity of RRW-automata. In: J. Dassow and D. Wotschke (eds.), *Preproceedings of the 3rd Workshop on Descriptional Complexity of Automata, Grammars and Related Structures,* Report No. 16, Fakultät für Informatik, Universität Magdeburg, 2001, 159–165.

[45] M. Plátek, F. Otto, F. Mráz, and T. Jurdzinski. Restarting automata and variants of j-monotonicity. *Mathematische Schriften Kassel*, no. 9/03, Fachbereich Mathematik/Informatik, Universität Kassel, 2003.

[46] S.H. von Solms. The characterization by automata of certain classes of languages in the context sensitive area. *Information and Control* 27 (1975) 262–271.

[47] M. Straňáková. Selected types of pg-ambiguity. *The Prague Bulletin of Mathematical Linguistics* 72 (1999) 29–57.

[48] M. Straňáková. Selected types of pg-ambiguity: Processing based on analysis by reduction. In: P. Sojka, I. Kopeček, and K. Pala (eds.), *Text, Speech and Dialogue, 3rd Int. Workshop, TSD 2000, Proc.*, *Lecture Notes in Computer Science 1902*, Springer, Berlin, 2000, 139–144.