On CD-systems of stateless deterministic R-automata with window size one

Benedek Nagya, Friedrich Otto

Abstract

Here we study cooperating distributed systems (CD-systems) of restarting automata that are very restricted: they are deterministic, they cannot rewrite, but only delete symbols, they restart immediately after performing a delete operation, they are stateless, and they have a read/write window of size 1 only, that is, these are stateless deterministic R(1)-automata. We study the expressive power of these systems by relating the class of languages that they accept by mode $= 1$ computations to other well-studied language classes, showing in particular that this class only contains semi-linear languages. Our model can be viewed as a nondeterministic finite-state acceptor with translucent letters, that is, it processes its input in a different way than the usual left-to-right order. In this way all commutative semi-linear languages, and in fact all rational trace languages, can be accepted. In addition, we investigate the closure and non-closure properties of the class of languages accepted by our model and some of its algorithmic properties.

1. Introduction

Cooperating distributed systems (CD-systems) of restarting automata have been defined in [17], and in [18,19] various types of deterministic CD-systems of restarting automata have been studied. As expected CD-systems are much more expressive than their component automata themselves. For example, already the marked copy language $L_{\text{copy}} = \{wcw \mid w \in \{a, b\}^*\}$ is accepted by a CD-system consisting of only two deterministic R-automata, although this language is not even growing context-sensitive [5,15], that is, it is not even accepted by any deterministic RRWW-automaton. On the other hand, stateless restarting automata, that is, restarting automata with only a single state, have been introduced and studied in [12–14]. In the monotone case and in the deterministic case, they are just as expressive as the corresponding restarting automata with states, provided that auxiliary symbols are available. Without the latter, however, stateless restarting automata are in general much less expressive than their corresponding counterparts with states.

Here we study deterministic restarting automata that are stateless and that have a read/write window of a fixed size $k > 0$, and CD-systems of such automata extending the work reported in [20]. In fact, we mainly concentrate on CD-systems of stateless deterministic R-automata with window size 1. The restarting automata of this type are really very...
restricted, and accordingly their expressive power is very limited. However, by combining several such automata into a CD-system we obtain a device that is surprisingly expressive, as we will see.

We first consider stateless deterministic R-automata, showing that we obtain an infinite hierarchy of language classes based on the window size. In fact, the different levels of this hierarchy can be separated from one another by regular languages. As already stateless deterministic R-automata of window size 2 can accept the Dyck language $D_n^*$ for all $n \geq 1$ (see, e.g., [3]), this shows that, for all $k \geq 2$, the class $\mathcal{L}$(stl-det-R($k$)) of languages accepted by stateless deterministic R-automata of window size $k$ is incomparable under inclusion to the class $\text{REG}$ of regular languages. However, all regular languages are accepted by stateless deterministic R-automata. Further, each stateless deterministic R-automaton of window size 2 is necessarily monotone, which implies that it accepts a deterministic context-free language. On the other hand, the class $\mathcal{L}$(stl-det-R(9)) contains a non-context-free language. As already stateless deterministic automata of this form for rational trace languages provide an effective calculus for rational trace languages in that from systems of this form for rational trace languages we can effectively construct systems for the rational trace language.

Then we restate the definition of CD-systems of restarting automata, and turn to our main topic, the CD-systems of stateless deterministic R(1)-automata. The definition of these CD-systems is quite technical (see below). However, such a system can be pictured informally as a nondeterministic finite-state acceptor (NFA) with translucent letters. Let $A = (Q, \Sigma, q_0, F, \delta)$ be an NFA, where $Q$ is a finite set of internal states, $\Sigma$ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. Assume that with each state $q \in Q$, we associate two disjoint subsets $\Sigma(q)$ and $\Sigma^*(q)$ of $\Sigma$. When $A$ is in state $q$, then it reads (and deletes) the first occurrence of a letter $a \in \Sigma^*(q)$ from its input tape, provided that this letter is preceded only by letters from $\Sigma(q)$. In this sense the letters from $\Sigma^*(q)$ are translucent for state $q$. Now the states of $A$ correspond to the component automata of a CD-system of stateless deterministic R(1)-automata. Also our CD-systems of stateless deterministic R(1)-automata can be related to a special variant of the binding-blocking automaton of Balan [2].

We compare the class of languages that are accepted by our CD-systems through mode = 1 computations to other well-known language classes. In particular, we show that in mode = 1 these systems only accept languages with semi-linear Parikh image, including all regular languages, but that they also accept some languages that are not even context-free. In fact, these systems accept all rational trace languages. Accordingly they can also be interpreted as a refinement of the so-called multiset finite automata of [7], which accept all regular macroses, that is, the commutative closures of all regular languages. In addition, we present a syntactic restriction for CD-systems of stateless deterministic R-automata of window size 1 such that the corresponding systems characterize the class of rational trace languages. These systems actually yield an effective calculus for rational trace languages in that from systems of this form for rational trace languages $S_1$ and $S_2$ we can effectively construct systems for the rational trace language $S_1 \cup S_2$, $S_1 \cdot S_2$ and $S_1^*$. Then we study closure and non-closure properties of the class of languages accepted by CD-systems of stateless deterministic R(1)-automata. We prove that this class is closed under union, product, Kleene-star, and inverse projections, but that it is neither closed under intersection with regular languages nor under $\epsilon$-free morphisms. Finally we address some algorithmic problems for CD-systems of stateless deterministic R(1)-automata like the emptiness problem, the finiteness problem, and the equivalence problem. The paper closes with a short summary and some open problems for future work.

2. Stateless R-automata with constant window size

We first describe in short the types of restarting automata we will be dealing with. More details can be found in [21].

A one-way restarting automaton, abbreviated as RW$W$-automaton, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \epsilon, \mathcal{E}, q_0, k, \delta)$, where $Q$ is a finite set of states, $\Sigma$ is a finite input alphabet, $\Gamma$ is a finite tape alphabet containing $\Sigma$, the symbols $\epsilon, \mathcal{E} \notin \Gamma$ serve as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the read/write window, and $\delta$ is a transition relation that associates a finite set of transition steps to each pair $(q, u)$ consisting of a state $q \in Q$ and a possible contents $u$ of the read/write window. There are four types of transition steps: move-right steps (MVR), which shift the window one position to the right and change the internal state, rewrite steps, which replace the content $u$ of the read/write window by a shorter word, thereby also shortening the tape, and change the internal state, restart steps (Restart), which place the read/write window over the left end of the tape, and reset the internal state to the initial state $q_0$, and accept steps (Accept), which cause $M$ to halt and accept. Here some restrictions apply in that the border markers $\epsilon$ and $\mathcal{E}$ cannot be removed from the tape, and the window cannot move to the right across the $\mathcal{E}$-symbol.

In general, $M$ is nondeterministic, as $\delta(q, u)$ may contain more than one transition step for some pairs $(q, u)$. If that is not the case, then $M$ is deterministic.

A configuration of $M$ is described by a string $\alpha \beta q$, where $q \in Q$, and either $\alpha = \epsilon$ (the empty word) and $\beta \in \{\epsilon\} \cdot \Gamma^\ast \cdot \{\mathcal{E}\}$ or $\alpha \in \{\epsilon\} \cdot \Gamma^\ast$ and $\beta \in \{\mathcal{E}\} \cdot \mathcal{E}^\ast$. Here $q$ represents the current state, $\alpha \beta$ is the current content of the tape, and it is understood that the head scans the first $k$ symbols of $\beta$ or all of $\beta$ when $|\beta| \leq k$. A restarting configuration is of the form $q_0 \epsilon w\mathcal{E}S$, where $w \in \Gamma^\ast$; if $w \in \Sigma^\ast$, then $q_0 \epsilon w\mathcal{E}S$ is an initial configuration. By $\vdash \downarrow_M$ we denote the single-step computation relation that $M$ induces on the set of its configurations, and $\vdash \downarrow_M$ denotes the reflexive transitive closure of $\vdash \downarrow_M$.

The automaton $M$ proceeds as follows. Starting from a restarting configuration $q_0 \epsilon w\mathcal{E}S$, the window is moved to the right by a sequence of move-right operations until a configuration of the form $\epsilon x q y u \mathcal{E}S$ is reached such that $\delta(q, u)$ contains a rewrite step that rewrites $u$ to $v$, that is, $(p, v) \in \delta(q, u)$ for some state $p \in Q$ and some word $v \in \Gamma^\ast$. Observe that then
\(|v| < |w|\) holds by definition. If this particular transition is now chosen, then the latter configuration is transformed into the configuration \(\varepsilon xvy\text{\$}\). Then \(M\) performs some more move-right steps until a restart step is executed, which then yields the restarting configuration \(q_0\varepsilon xvy\text{\$}\). This computation, which is called a cycle, is expressed as \(w \overset{i-M}{\Rightarrow} xvy\). A computation of \(M\) consists of a finite sequence of cycles that is followed by a tail computation, which consists of a sequence of move-right operations that may include a single rewrite step, and that is completed by either an accept step, or that reaches a configuration in which \(M\) cannot perform another transition step. In the former case we say that \(M\) accepts, while in the latter it rejects. A word \(w \in \Gamma^*\) is accepted by \(M\), if there is a computation of \(M\) which starts with the configuration \(q_0\varepsilon w\text{\$}\), and which finishes by executing an accept step. By \(L_c(M)\) we denote the language consisting of all words accepted by \(M\). It is called the characteristic language of \(M\), and \(L(M) = L_c(M) \cap \Sigma^*\) is the (input) language of \(M\).

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

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

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

In \([12]\) the stateless variants of \(RWW\)-automata are studied, where an \(RWW\)-automaton \(M = (Q, \Sigma, \Gamma, \varepsilon, \text{\$}, q_0, k, \delta)\) is called stateless if \(Q = \{q_0\}\) holds. Thus, in this case \(M\) can simply be described by the 6-tuple \(M = (\Sigma, \Gamma, \varepsilon, \text{\$}, k, \delta)\). In the original definition it was required that a stateless \(RWW\)-automaton may execute an accept instruction only at the right end of the tape, that is, when it sees the right delimiter \(\text{\$}\), but this is actually just a convenience, as shown by the following proposition.

**Proposition 2.1.** (See \([14]\).) From a stateless \(RWW\)-automaton \(M = (\Sigma, \Gamma, \varepsilon, \text{\$}, k, \delta)\), one can construct a stateless \(RWW\)-automaton \(M' = (\Sigma, \Gamma, \varepsilon, k + 1, \delta)\) that executes accept instructions only at the right end of the tape, and that accepts the same characteristic language as \(M\). If \(M\) is an \(R\)-automaton or an \(R\)-automaton, then so is \(M'\), and if \(M\) is deterministic, then so is \(M'\).

In \([12]\) the following results were obtained, where the prefix \(stl\)- is used to denote stateless types of restarting automata, the prefix \(det\)- is used to denote deterministic types of restarting automata, and the prefix \(mon\)- is used to denote restarting automata that are monotone. Here a restarting automaton \(M\) is called monotone, if the distance from the place of rewriting to the right end of the tape does not increase from one cycle to the next in any computation of \(M\). We use the notation \(L(X)\) to denote the class of (input) languages that are accepted by automata of type \(X\).

**Theorem 2.2.** (See \([12, 14]\).)

(a) \(L(stl-det-mon-RWW) = DCFL\).
(b) \(L(stl-mon-RWW) = CFL\).
(c) \(L(stl-det-RWW) = CFL\).
(d) \(L(stl-det-mon-R) \supseteq \text{REG}\).

Here \(\text{CFL}\) denotes the class of Church-Rosser languages of McNaughton et al. \([16]\), \(DCFL\) is the class of deterministic context-free languages, and \(\text{REG}\) denotes the class of regular languages.

We are interested in stateless \(R\)-automata with a fixed window size. For each positive integer \(k\), we denote by \(stl-det-R(k)\) the class of stateless deterministic \(R\)-automata that have a read/write window of size \(k\). We will see that there is an infinite hierarchy of language classes \(L(stl-det-R(k))\) based on the value of the parameter \(k\).

First we consider stateless deterministic \(R\)-automata with window size 1. For these automata we introduce the following notions that we will repeatedly use throughout the paper.

**Definition 2.3.** Assume that \(M = (\Sigma, \varepsilon, 1, 1)\) is a stateless deterministic \(R\)-automaton of window size 1. Then we can partition the alphabet \(\Sigma\) into four disjoint subalphabets:

1. \(\Sigma_1 = \{a \in \Sigma \mid \delta(a) = \text{MVR}\}\),
2. \(\Sigma_2 = \{a \in \Sigma \mid \delta(a) = \varepsilon\}\),
3. \(\Sigma_3 = \{a \in \Sigma \mid \delta(a) = \text{Accept}\}\),
4. \(\Sigma_4 = \{a \in \Sigma \mid \delta(a) = \emptyset\}\).

Thus, \(\Sigma_1\) is the set of letters that \(M\) just moves across, \(\Sigma_2\) is the set of letters that \(M\) deletes, \(\Sigma_3\) is the set of letters which cause \(M\) to accept, and \(\Sigma_4\) is the set of letters on which \(M\) will get stuck.
Then the following characterization holds.

**Proposition 2.4.** Let \( M = (\Sigma, \Sigma, \epsilon, S, 1, \delta) \) be a stateless deterministic \( R(1) \)-automaton, and assume that the subalphabets \( \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4 \) are defined as above. Then the simple language \( S(M) \) of words accepted by \( M \) in tail computations is characterized as

\[
S(M) = \begin{cases}
\emptyset, & \text{if } \delta(\epsilon) = \emptyset, \\
\Sigma^*, & \text{if } \delta(\epsilon) = \text{Accept}, \\
\Sigma_1^* \cdot \Sigma_3 \cdot \Sigma^*, & \text{if } \delta(\epsilon) = \text{MVR and } \delta(S) \neq \text{Accept}, \\
\Sigma_1^* \cdot ((\Sigma_3 \cdot \Sigma^*) \cup \{\epsilon\}), & \text{if } \delta(\epsilon) = \text{MVR and } \delta(S) = \text{Accept}.
\end{cases}
\]

and the language \( L(M) \) is characterized as

\[
L(M) = \begin{cases}
\emptyset, & \text{if } \delta(\epsilon) = \emptyset, \\
\Sigma^*, & \text{if } \delta(\epsilon) = \text{Accept}, \\
(\Sigma_1 \cup \Sigma_2)^* \cdot \Sigma_3 \cdot \Sigma^*, & \text{if } \delta(\epsilon) = \text{MVR and } \delta(S) \neq \text{Accept}, \\
(\Sigma_1 \cup \Sigma_2)^* \cdot ((\Sigma_3 \cdot \Sigma^*) \cup \{\epsilon\}), & \text{if } \delta(\epsilon) = \text{MVR and } \delta(S) = \text{Accept}.
\end{cases}
\]

**Proof.** If \( \delta(\epsilon) = \emptyset \), then \( M \) gets immediately stuck for each word \( w \in \Sigma^* \), and if \( \delta(\epsilon) = \text{Accept} \), then \( M \) accepts each word \( w \in \Sigma^* \) in a tail computation. Thus, we can concentrate on the case that \( \delta(\epsilon) = \text{MVR} \) holds. Obviously, \( M \) will then accept each word from \( \Sigma_1^* \cdot \Sigma_3 \cdot \Sigma^* \) in a tail computation, and if \( \delta(S) = \text{Accept} \), it will also accept each word from \( \Sigma_1^* \) in a tail computation. Further, each word \( w = uav \), where \( u \in \Sigma_1^*, a \in \Sigma_2, \) and \( v \in \Sigma^* \) will cause a cycle of the form \( w = uav \uparrow_M uv \). Hence, one by one those letters from \( \Sigma_2 \) are removed from \( w \) that in \( w \) are only preceded by letters from \( \Sigma_1 \cup \Sigma_2 \). This yields the above description for the language \( L(M) \). \( \Box \)

It is easily seen that a stateless finite-state acceptor with input alphabet \( \Sigma \) accepts a language of the form \( \Sigma_0^* \), where \( \Sigma_0 \) is a subalphabet of \( \Sigma \). Thus, we have the following easy consequence.

**Corollary 2.5.** A language \( L \) is accepted by a stateless deterministic \( R(1) \)-automaton that only accepts on reaching the right delimiter \( S \), if and only if \( L \) is the simple language of a stateless deterministic \( R(1) \)-automaton that only accepts on reaching the right delimiter \( S \), if and only if \( L \) is accepted by a stateless finite-state acceptor.

**Proof.** Let \( M = (\Sigma, \Sigma, \epsilon, S, 1, \delta) \) be a stateless deterministic \( R(1) \)-automaton that only accepts on reaching the right delimiter \( S \). Then \( \Sigma_3 = \emptyset \), and hence we see from the above proposition that \( L(M) = (\Sigma_1 \cup \Sigma_2)^* \). On the other hand, if \( A \) is a stateless finite-state acceptor on \( \Sigma \) that accepts the language \( \Sigma_0^* \), then we obtain a stateless deterministic \( R(1) \)-automaton \( M = (\Sigma, \Sigma, \epsilon, S, 1, \delta) \) by defining \( \delta(\epsilon) = \text{MVR}, \delta(a) = \text{MVR} \) for all letters \( a \in \Sigma_0, \delta(b) = \emptyset \) for all letters \( b \in \Sigma \setminus \Sigma_0 \), and \( \delta(S) = \text{Accept} \). Then \( L(M) = S(M) = \Sigma_0^* \). \( \Box \)

Thus, stateless deterministic \( R(1) \)-automata can be seen as stateless deterministic finite-state acceptors that are enabled to accept without having read their input completely. Next we turn to stateless deterministic \( R \)-automata of window size 2.

**Lemma 2.6.** The Dyck language \( D_{R,\omega}^n \) is accepted by a stateless deterministic \( R(2) \)-automaton for each integer \( n \geq 1 \).

**Proof.** The Dyck language \( D_{R,\omega}^n \) is defined over the alphabet \( T_1 = \{a, b\} \). It is generated by the context-free grammar \( G_1 = ([S, A], T_1, P_1, S) \), where \( P_1 \) contains the productions \( S \rightarrow AS, S \rightarrow \epsilon \), and \( A \rightarrow aSb \) (see, e.g., [3]). In fact, a word \( w \in T_1^* \) belongs to \( D_{R,\omega}^1 \) if and only if \( |w|_a = |w|_b \) and for each proper prefix \( x \) of \( w \) we have \( |x|_a \geq |x|_b \). Thus, it is easily seen that \( D_{R,\omega}^1 \) is accepted by the stateless deterministic \( R(2) \)-automaton \( M_1 \) that is defined by the following transition function:

1. \( \delta(\epsilon S) = \text{Accept} \)
2. \( \delta(aa) = \text{MVR} \)
3. \( \delta(ab) = \text{MVR} \)
4. \( \delta(ab) = \epsilon \).

Here we assume that \( \delta(u) = \emptyset \) for all other cases. Thus, we see that \( D_{R,\omega}^1 \in \mathcal{L}((\text{stl-det-R}(2))) \) holds. It can be shown analogously that each Dyck language \( D_{R,\omega}^n \), \( n \geq 2 \), is accepted by a stateless deterministic \( R(2) \)-automaton. \( \Box \)

If \( M = (\Sigma, \Sigma, \epsilon, S, 2, \delta) \) is a stateless deterministic \( R(2) \)-automaton, then each cycle \( w \uparrow_M w' \) has the form \( w = uabv \) and \( w' = ucv \), where \( u, v \in \Sigma^*, a, b \in \Sigma, \) and \( c = a, c = b, \) or \( c = \epsilon \). As \( M \) is deterministic, it must scan the prefix \( uc \) of \( w' \) completely before it can apply another delete step. Hence, we see that \( M \) is necessarily monotone. As monotone deterministic \( R \)-automata accept deterministic context-free languages only (see, e.g., [21]), this observation has the following consequence.
Lemma 2.7. \(L(\text{stl-det-R}(2)) \subseteq \text{DCFL}\).

Actually the above inclusion is a proper one. This follows immediately from the following result.

Lemma 2.8. For each integer \(k \geq 1\), there exists a regular language \(L_k \subseteq \{a, b\}^*\) such that \(L_k \in L(\text{stl-det-R}(k+1)) \setminus L(\text{stl-det-R}(k))\), that is, \(L_k\) is accepted by a stateless deterministic \(R\)-automaton of window size \(k+1\), but it is not accepted by any stateless deterministic \(R\)-automaton of window size \(k\).

**Proof.** For each integer \(k \geq 1\), let \(L_k = \{(ab^2)^i \mid i \geq 0\}\). Then \(L_k\) is obviously regular. Further, it is accepted by the stateless deterministic \(R\)-automaton of window size \(k+1\) that is defined by the following transition function \(\delta_{k+1}\):

1. \(\delta_{k+1}(c\$) = \text{Accept}\).  
2. \(\delta_{k+1}(cab^{k-1}) = \text{MVR}\).  
3. \(\delta_{k+1}(ab^k) = \varepsilon\).

Here again we use the convention that \(\delta_{k+1}(u) = \emptyset\) for all other cases.

On the other hand, if \(M = (\Sigma, \Sigma, \varepsilon, S, k, \delta)\) is a stateless deterministic \(R\)-automaton that accepts the language \(L_k\), then on input \(ab^2ab^k\), \(M\) will have to accept. If \(\delta(cab^{k-2}) = \text{Accept}\), then \(M\) would accept on input \(ab^{k-2} \notin L_k\). If \(\delta(cab^{k-2}) = e\), then \(M\) would execute the cycle \(ab^k \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} ab^2\), and as \(M\) is deterministic and \(ab^k \in L_k\), \(M\) would have to accept the word \(vb^2\). However, \(v\) is obtained from the word \(ab^{k-2}\) by deleting one or more letters, that is, \(v = ab^j\) for some \(i \leq k - 3\) or \(v = b^i\) for some \(i \leq k - 2\). In any case it follows that \(vb^2 \notin L_k\), a contradiction. Hence, it follows that \(\delta(cab^{k-2}) = \text{MVR}\). Thus, after the first step \(M\) reaches the configuration \(eqab^2ab^k\$\), where \(q\) symbolizes the unique state of \(M\). Now \(\delta(ab^{k-1})\) has to be applied. Again it can neither be an accept nor a delete operation, that is, \(\delta(ab^{k-1}) = \text{MVR}\), and so \(M\) reaches the configuration \(cab^2ab^k\$\). Continuing in this way we see that \(M\) will just move across its tape inscription, that is, \(\delta(z) = \text{MVR}\) for all \(z \in \{ab^{k-1}, b^{k-1}a, b^{k-2}ab, \ldots, bab^{k-2}\}\). Finally \(M\) will reach the configuration \(cab^2ab^k\$\), and it will have to accept. However, it will then also accept the word \(ab^{k+1}ab^k\) that does not belong to the language \(L_k\). Hence, \(L_k\) is not accepted by any stateless deterministic \(R\)-automaton of window size \(k\). \(\square\)

Together with Lemma 2.6 this yields the following results.

**Corollary 2.9.** The language classes \(L(\text{stl-det-R}(k))\) for \(k \geq 1\) form an infinite strictly increasing hierarchy. For all \(k \geq 2\), the class \(L(\text{stl-det-R}(k))\) is incomparable under inclusion to the class \(\text{REG}\) of regular languages.

Stateless deterministic \(R\)-automata of window size 2 only accept certain deterministic context-free languages. Next we will see that with larger window size these automata do even accept some languages that are not context-free.

Let \(L_{\text{expo}}\) and \(L^{(\psi)}_{\text{expo}}\) be the following languages over \([a, b]\):

\[
L_{\text{expo}} = \left\{a^{i_0}ba^{i_1}b \cdots a^{i_{n-1}}ba^{i_n} \mid n \geq 0, i_0, \ldots, i_n \geq 0, \text{ and } \exists m \geq 0: \sum_{j=0}^{n} 2^j \cdot i_j = 2^m \right\} \cup b^*.
\]

\[
L^{(\psi)}_{\text{expo}} = \varphi(L_{\text{expo}}),
\]

where \(\varphi\) is the morphism induced by \(a \mapsto ab\) and \(b \mapsto b\). These languages are not context-free, as \(L_{\text{expo}} \cap a^* = \{a^{2n} \mid n \geq 0\}\) and \(L^{(\psi)}_{\text{expo}} \cap (ab)^* = \{a^{2n} \mid n \geq 0\}\). On the other hand, it is shown in [14] that the language \(L^{(\psi)}_{\text{expo}}\) is accepted by a stateless deterministic \(R\)-automaton. In fact, the particular \(R\)-automaton for this language that is presented there has window size 9. Together with Corollary 2.9 this yields the following consequence.

**Corollary 2.10.** For all \(k \geq 9\), the class \(L(\text{stl-det-R}(k))\) is incomparable under inclusion to the class \(\text{CFL}\) of context-free languages.

**Open Problem 1.** What is the smallest integer \(k\) such that the language class \(L(\text{stl-det-R}(k))\) contains a non-context-free language? From our results above we know that \(3 \leq k \leq 9\) holds, but it is open whether already the class \(L(\text{stl-det-R}(3))\) contains a non-context-free language.

In [12] it is shown that the deterministic linear language

\[L_d = \{ca^nb^n \mid n \geq 0\} \cup \{da^nb^n \mid n \geq 0\}\]

is not accepted by any stateless \(RW\)-automaton. This yields the following non-inclusion result.

**Corollary 2.11.** \(\text{DCFL} \not\subseteq L(\text{stl-det-R}).\)

Hence, the class \(L(\text{stl-det-R})\) is incomparable to the class of (deterministic) context-free languages. The diagram in Fig. 1 summarizes the inclusion relations between the language classes that are accepted by the various types of stateless deterministic \(R\)-automata and some classical language families.
3. CD-systems of restarting automata

Here we restate the definition of a CD-system of restarting automata from [17] in short. These systems can be seen as an adaptation of the notion of a CD-grammar system with external control (see, e.g., [6]) to restarting automata. For these systems the way of choosing the next component is similar to the way of choosing the next production in a programmed grammar without appearance checking (see, e.g., [8]).

A cooperating distributed system of RRWW-automata (or a CD-RRWW-system, for short) consists of a finite collection \( \mathcal{M} = ((M_i, \sigma_i), i \in I) \) of RRWW-automata \( M_i = (Q_i, \Sigma, \Gamma, \iota, S, \delta_i) \) \((i \in I)\), successor relations \( \sigma_i \subseteq I \) \((i \in I)\), and a subset \( I_0 \subseteq I \) of initial indices. Here it is required that \( Q_i \cap Q_j = \emptyset \) for all \( i, j \in I \), \( i \neq j \), that \( I_0 \neq \emptyset \), that \( \sigma_i \neq \emptyset \) for all \( i \in I \), and that \( i \notin \sigma_i \) for all \( i \in I \).

Various modes of operation have been introduced and studied, but here we are only interested in mode \( = 1 \) computations. A computation of \( \mathcal{M} \) in mode \( = 1 \) on an input word \( w \) proceeds as follows. First an index \( i_0 \in I_0 \) is chosen nondeterministically. Then the RRWW-automaton \( M_{i_0} \) starts the computation with the initial configuration \( q_0^{(i_0)} \in \omega \Sigma \), and executes exactly one cycle. Thereafter an index \( i_1 \in \sigma_{i_0} \) is chosen nondeterministically, and \( M_{i_1} \) continues the computation by executing exactly one cycle. This continues until, for some \( l \geq 0 \), the automaton \( M_{i_l} \) accepts. Should at some stage the chosen automaton \( M_{i_l} \) be unable to execute a cycle or to accept, then the computation fails.

By \( L_{=1}(\mathcal{M}) \) we denote the language that the CD-RRWW-system \( \mathcal{M} \) accepts in mode \( = 1 \). It consists of all words \( w \in \Sigma^* \) that are accepted by \( \mathcal{M} \) in mode \( = 1 \) as described above. If \( X \) is any of the above types of restarting automata, then a CD-X-system is a CD-RRWW-system for which all component automata are of type \( X \).

A CD-system of restarting automata \( \mathcal{M} = ((M_i, \sigma_i), i \in I) \) is called stateless if all component automata \( M_i \) \((i \in I)\) are stateless. Here we are interested in CD-systems of stateless deterministic R-automata. We call these systems stl-det-local-CD-R-systems in accordance with the notation introduced in [19]. Observe that the computations of such a CD-system are not completely deterministic, as the starting component and the respective successor components are still chosen nondeterministically from among all available component automata. By \( L_{=1}(\text{stl-det-local-CD-R}(i)) \) we denote the class of languages that are accepted by mode \( = 1 \) computations of stl-det-local-CD-R-systems with window size \( i \). The following example illustrates the expressive power of these systems.

**Example 3.1.** We consider the marked copy language \( L_{\text{copy}} = \{wcw \mid w \in \{a, b\}^*\} \) on \( \Sigma = \{a, b, c\} \). It is well known that this language is not even growing context-sensitive (see, e.g., [21]), and so it is not accepted by any deterministic RRWW-automaton. However, we will see that it is accepted by a stl-det-local-CD-R(2)-system with four components working in mode \( = 1 \).

Let \( \mathcal{M} = ((M_i, \sigma_i), i \in I) \), where \( I = \{a, b, -, +\}, I_0 = \{a, b, +\}, \sigma_a = \{-\} = \sigma_b, \sigma_- = \{a, b, +\}, \sigma_+ = \{-\}, \) and \( M_a, M_b, M_-, \) and \( M_+ \) are the stateless deterministic R(2)-automata that are given by the following transition functions:

\[
M_a: \quad
\begin{align*}
(1) \quad & \delta_a(\epsilon a) = \text{MVR}, \\
(2) \quad & \delta_a(xy) = \text{MVR} \quad \text{for all } x \in \{a, b\} \text{ and } y \in \Sigma, \\
(3) \quad & \delta_a(ca) = c,
\end{align*}
\]
CD-systems of stateless deterministic R-automata with window size 1

As already CD-systems of stateless deterministic R-automata of window size 2 can accept some languages that are not even growing context-sensitive, we now concentrate on a class of CD-systems of restarting automata that are still more restricted: CD-systems of stateless deterministic R-automata of window size 1. As shown by Proposition 2.4 stateless deterministic R-automata of window size 1 can only accept regular languages of a rather restricted form. So it is certainly of interest to investigate the expressive power of CD-systems of restarting automata of this particular form. We start our investigation by presenting two examples of non-regular languages that are accepted by CD-systems of this form.

Proposition 4.1. The Dyck language $D_1^*$ is accepted by a CD-system of stateless deterministic R-automata of window size 1 working in mode $= 1$.

Proof. Let $\mathcal{M} = (((M_i, \sigma_i)_{i \in I}, I_0)$, where $I = \{a, b, +\}$, $I_0 = \{a, +\}$, $\sigma_a = \{b\}$, $\sigma_b = \{a, +\}$, $\sigma_+ = \{a\}$, and $M_a$, $M_b$, and $M_+$ are the stateless deterministic R-automata of window size 1 that are given by the following transition functions:

\begin{align*}
M_a: & \quad (1) \quad \delta_a(\varepsilon) = \text{MVR}, \\
& \quad (2) \quad \delta_a(a) = \varepsilon, \\
M_b: & \quad (3) \quad \delta_b(\varepsilon) = \text{MVR}, \\
& \quad (4) \quad \delta_b(a) = \text{MVR}, \\
& \quad (5) \quad \delta_b(b) = \varepsilon, \\
M_+: & \quad (6) \quad \delta_+(\varepsilon) = \text{MVR}, \\
& \quad (7) \quad \delta_+(\$) = \text{Accept}.
\end{align*}

Let $w \in \{a, b\}^*$ be given as input. The automaton $M_+$ accepts the empty word and rejects (that is, gets stuck on) all other inputs. As $+ \in I_0$, we see that the empty word is accepted by $\mathcal{M}$ working in mode $= 1$. If $w \neq \varepsilon$, then the computation starts with $M_a$. If $w = aw_1$, then $M_a$ simply deletes the first occurrence of $a$ in $w$, otherwise, it gets stuck. Then $M_b$ takes over, which deletes the first occurrence of the letter $b$, provided $|w_1|_b > 0$. Now this sequence consisting of two cycles is repeated until either the empty word is reached, and then the computation finishes with $M_+$ accepting, or until a non-empty word is reached that does not start with the letter $a$, or that does not contain any occurrences of the letter $b$, and then the computation gets stuck. From the properties of $D_1^*$ restated in the proof of Lemma 2.6 it now follows that $L_{\text{stl}, \text{det-local-CD-R}}(2)$ holds. □

Proposition 4.2. The language $L_{\text{abc}} = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c \geq 0\}$ is accepted by a CD-system of stateless deterministic R-automata of window size 1 working in mode $= 1$.

Proof. Let $\mathcal{M} = (((M_i, \sigma_i)_{i \in I}, I_0)$, where $I = \{a, b, c, +\}$, $I_0 = \{a, +\}$, $\sigma_a = \{b, c\}$, $\sigma_b = \{a, c\}$, $\sigma_+ = \{a\}$, and $M_a$, $M_b$, $M_c$, and $M_+$ are the stateless deterministic R-automata of window size 1 that are given by the following transition functions:

\begin{align*}
M_a: & \quad (1) \quad \delta_a(\varepsilon) = \text{MVR}, \\
& \quad (2) \quad \delta_a(x) = \text{MVR} \quad \text{for all } x \in \{b, c\}, \\
& \quad (3) \quad \delta_a(a) = \varepsilon .
\end{align*}
Let $w \in [a, b, c]^*$ be given as input. The automaton $M_\rightarrow$ accepts the empty word and rejects (that is, gets stuck on) all other inputs. As $+ \in I_0$, we see that the empty word is accepted by $M$ working in mode $= 1$. If $w \neq \varepsilon$, then the computation starts with $M_a$. If $|w|_a > 0$, then $M_a$ simply deletes the first occurrence of $a$ in $w$, otherwise, it gets stuck. Then $M_b$ takes over, which deletes the first occurrence of the letter $b$, provided $|w|_b > 0$. Finally, $M_c$ deletes the first occurrence of the letter $c$, if $|w|_c > 0$. Now this sequence consisting of three cycles is repeated until either the empty word is reached, and then the computation finishes with $M_\rightarrow$ accepting, or until a non-empty word is reached that does not contain occurrences of all three letters, and then the computation gets stuck. It follows that $L_{\rightarrow} = L_{abc}$ holds. \hfill\(\square\)

Observe that the CD-system above for accepting the language $L_{abc}$ consists of only four R(1)-automata. As the language $L_{abc}$ is not context-free, we have the following consequence.

**Corollary 4.3.** The language class $L_{\rightarrow} = \{\text{stl-det-local-CD-R(1)}\}$ contains languages that are not context-free.

In addition, all regular languages are accepted by $\text{stl-det-local-CD-R(1)}$-systems working in mode $= 1$.

**Proposition 4.4.** $\text{REG} \subseteq L_{\rightarrow} = \{\text{stl-det-local-CD-R(1)}\}$.

**Proof.** Let $L \subseteq \Sigma^*$ be a regular language, and let $A = (Q, \Sigma, p_0, F, \delta)$ be a complete deterministic finite-state acceptor for $L$. From $A$ we construct a $\text{stl-det-local-CD-R(1)}$-system $M = ((M_i, \sigma_i)_{i \in I}, I_0)$ as follows:

- the set of indices is $I = (Q \times \Sigma) \cup (Q' \times \Sigma) \cup \{+\}$, where $Q' = \{q' \mid q \in Q\}$ is a copy of $Q$ such that $Q \cap Q' = \emptyset$.
- the set of initial indices is

\[
I_0 = \begin{cases} 
(p_0, a) \mid a \in \Sigma, & \text{if } \varepsilon \notin L, \\
(p_0, a) \mid a \in \Sigma \cup \{+\}, & \text{if } \varepsilon \in L,
\end{cases}
\]

- the successor relations are defined by

\[
\begin{align*}
\sigma(q, a) &= \{ (\delta(q, a), b) \mid b \in \Sigma \cup \{+\}, \text{ if } \delta(q, a) \neq q \text{ and } \delta(q, a) \notin F, \\
&\quad \{ (\delta(q, a), b) \mid b \in \Sigma, \text{ if } \delta(q, a) \neq q \text{ and } \delta(q, a) \notin F, \\
&\quad \{ (q', b) \mid b \in \Sigma \cup \{+\}, \text{ if } \delta(q, a) = q \text{ and } q \in F, \\
&\quad \{ (q', b) \mid b \in \Sigma, \text{ if } \delta(q, a) = q \text{ and } q \notin F, \\
&\quad \{ (\delta(q, a), b) \mid b \in \Sigma \cup \{+\}, \text{ if } \delta(q, a) \notin F, \\
&\quad \{ (p_0, a) \mid a \in \Sigma, \text{ if } \delta(q, a) \notin F, \\
\end{align*}
\]

- and the $\text{stl-det-R(1)}$-automata $M_{(q, a)}$, $M_{(q', a)}$, and $M_{\rightarrow}$ are defined by the following transition functions:

\[
\begin{align*}
M_{(q, a)}: & \quad \delta_{(q, a)}(\varepsilon) = \text{MVR}, \\
& \quad \delta_{(q, a)}(a) = \varepsilon, \\
M_{(q', a)}: & \quad \delta_{(q', a)}(\varepsilon) = \text{MVR}, \\
& \quad \delta_{(q', a)}(a) = \varepsilon, \\
M_{\rightarrow}: & \quad \delta_{\rightarrow}(\varepsilon) = \text{MVR}, \\
& \quad \delta_{\rightarrow}(\$) = \text{Accept}.
\end{align*}
\]

Here the components $M_{(q', a)}$ ($q' \in Q', a \in \Sigma$) are only needed to ensure that $(q, a) \notin \sigma_{(q, a)}$ for any pair $(q, a)$, that is, no component $M_{(q, a)}$ is a successor of itself.
It can now be checked easily that the accepting mode $= 1$ computations of $\mathcal{M}$ correspond one-to-one to the accepting computations of the finite-state acceptor $A$. In fact, if $A$ executes the transition $\delta(q, a) = p$, then the component automaton $M(q, a)$ (or $M(q', a)$) must be active. It simply deletes the first letter to the right of the left delimiter $\epsilon$ (provided that is an $a$), and then the component automaton $M(p, b)$ (or $M(p', b)$, if $p = q$) becomes active, where it is guessed that the next letter to be processed by $A$ is a $b$. Thus, it follows that $L = L(A) = L_{w1}(\mathcal{M})$ holds. $\square$

Observe that the proof above crucially depends on the fact that in a mode $= 1$ computation of a stl-det-local-CD-R(1)-system, the initial component automaton and the successor automata are chosen nondeterministically from among the corresponding sets. Also notice that the stl-det-local-CD-R(1)-system constructed above does not read across any letters, that is, it processes its input strictly from left-to-right, not using any translucent letters.

Open Problem 2. Observe that the above simulation of a deterministic finite-state acceptor by a stl-det-local-CD-R(1)-system is rather inefficient, as we have used $O(|Q| \cdot |\Sigma|)$ many component automata. Is there a more efficient (that is, more succinct) simulation?

Currently we have no answer to this question, but we can at least show that in some instances stl-det-local-CD-R(1)-systems are much more succinct than even nondeterministic finite-state acceptors. Here we take the number of component automata of a CD-system as its (static) complexity measure.

Example 4.5. Let $\Sigma = \{a, b, c\}$, and let $n \geq 1$. We define the language $L_{wn} \subseteq \Sigma^*$ as follows

$$L_{wn} = \{ w \in \Sigma^* \mid |w|_a = n = |w|_b \}.$$  

We can easily construct a stl-det-local-CD-R(1)-system $\mathcal{M}$ with $2n + 1$ components that accepts the language $L_{wn}$ in mode $= 1$. We just need $n$ component automata that each simple delete one occurrence of the letter $a$, while moving right across occurrences of the letters $b$ and $c$, we need another $n$ component automata that each simply delete one occurrence of the letter $b$, while moving right across occurrences of the letter $c$, and we need a final component that accepts all words from $c^*$.

Now assume that $A = (Q, \Sigma, q_0, F, \delta)$ is a nondeterministic finite-state acceptor for $L_{wn}$. We claim that $A$ has at least $(n + 1)^3$ many states. Just consider the words $x_{i,j} = a^ib^j$ and $y_{i,j} = a^ib^ja^nb^{n-j}$ for all $i, j = 0, 1, \ldots, n$. Then $x_{i,j}y_{i,j} = a^ib^ja^nb^{n-j} \in L_{wn}$ for all $i, j$, while $x_{i,j}y_{i',j} \notin L_{wn}$, whenever $i' \neq i$ or $j' \neq j$. Thus, the set of pairs $(x_{i,j}, y_{i,j})_{i,j=0,\ldots,n}$ is a fooling set for $L_{wn}$. Accordingly it follows that $|Q| \geq (n + 1)^2$ [4].

Analogously for the finite language

$$L'_{wn} = \{ w \in \Sigma^* \mid |w|_a = |w|_b = |w|_c = n \}$$

we have a stl-det-local-CD-R(1)-system consisting of $3n + 1$ component automata, while an NFA for this language needs at least $(n + 1)^3$ many states.

Open Problem 3. Can we realize an exponential trade-off between stl-det-local-CD-R(1)-systems and nondeterministic finite-state acceptors?

If $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ is a stl-det-local-CD-R(1)-system, then those components $M_i$ of $\mathcal{M}$ for which $\delta_i(\epsilon) = \emptyset$ are useless, as they cannot be used in any accepting computation of $\mathcal{M}$. Thus, in what follows we can assume that $\delta_i(\epsilon) \in \{\text{MVR}, \text{Accept}\}$ for all component automata $M_i$ of $\mathcal{M}$. In fact, we can impose even stronger restrictions on stl-det-local-CD-R(1)-systems as expressed by the following normal form.

Definition 4.6. A stl-det-local-CD-R(1)-system $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ on alphabet $\Sigma$ is in normal form, if it satisfies the following three conditions for all $i \in I$, where $\Sigma_1^{(i)}, \Sigma_2^{(i)}, \Sigma_3^{(i)}, \Sigma_4^{(i)}$ is the partitioning of alphabet $\Sigma$ from Definition 2.3 for automaton $M_i$:

1. For the component automaton $M_i$, we have $|\Sigma_2^{(i)}| \leq 1$, that is, there is at most one letter that $M_i$ deletes.
2. All accept instructions are executed on the $\epsilon$-symbol, that is, $\delta_i(\epsilon) = \text{MVR}$ and $\Sigma_3^{(i)} = \emptyset$.
3. $M_i$ does not have both, rewrite instructions and accept instructions, that is, if $\delta_i(\$) = \text{Accept}$, then $\Sigma_2^{(i)} = \emptyset$.

If $\mathcal{M}$ is in normal form, and $\Sigma_1^{(i)} = \emptyset$ and $\delta_i(\$) \neq \text{Accept}$ for some index $i$, then $M_i$ cannot be used in any accepting computation of $\mathcal{M}$, that is, we could simply drop $M_i$ from $\mathcal{M}$. Hence, we can assume that $\delta_i(\$) = \text{Accept}$ if and only if $\Sigma_1^{(i)} = \emptyset$. 


Lemma 4.7. From a slt-det-local-CD-R(1)-system $\mathcal{M}$ one can construct a slt-det-local-CD-R(1)-system $\mathcal{M}'$ such that the system $\mathcal{M}'$ is in normal form and $L_{=1}(\mathcal{M}') = L_{=1}(\mathcal{M})$.

Proof. Let $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ be a slt-det-local-CD-R(1)-system. First we split every component automaton $M_i$ into $|\Sigma_2^{(i)}|+1$ many parts, $\hat{M}_i$ for $a \in \Sigma_2^{(i)}$, and $M_i^{(+)2}$, where the former is responsible for executing the cycles of $M_i$ in which an occurrence of the letter $a$ is deleted, while the latter takes care of the accepting tail computations of $M_i$. In detail, for each $a \in \Sigma_2^{(i)}$, and all $b, c \in \Sigma$,

$$
\delta_i^{(0)}(c) = \emptyset, \quad \text{if } \delta_i(c) = \text{Accept},
\delta_i^{(0)}(c) = \text{MVR}, \quad \text{if } \delta_i(c) = \text{MVR},
\delta_i^{(0)}(b) = \text{MVR}, \quad \text{if } \delta_i(b) = \text{MVR},
\delta_i^{(0)}(a) = \varepsilon,
\delta_i^{(+)}(c) = \text{Accept}, \quad \text{if } \delta_i(c) = \text{Accept},
\delta_i^{(+)}(c) = \text{MVR}, \quad \text{if } \delta_i(c) = \text{MVR},
\delta_i^{(+)}(b) = \text{MVR}, \quad \text{if } \delta_i(b) = \text{MVR},
\delta_i^{(+)}(c) = \text{Accept}, \quad \text{if } \delta_i(c) = \text{Accept},
\delta_i^{(+S)}(S) = \text{Accept}, \quad \text{if } \delta_i(S) = \text{Accept}.
$$

Then we adjust the successor relations $\sigma_i$ ($i \in I$) as follows

$$
\sigma_i^{(a)} = \sigma_i^{(+)} = \{ j(b), j^{(+)} \mid j \in \sigma_i, b \in \Sigma_{2 \times I_i} \}.
$$

Observe, however, that the successor relations $\sigma_i^{(+)}$ are never used in any computation. Finally, we take $\hat{\mathcal{M}} = ((\hat{M}_i^{(a)}, \sigma_i^{(a)})_{i \in I, a \in \Sigma_2^{(i)}}, \cup (M_i^{(+)2}, \sigma_i^{(+)2}(c)), \hat{I}_0)$, where $\hat{I}_0 = \{i^{(0)}, i^{(+)} \mid i \in I_0, a \in \Sigma_2^{(i)}\}$.

Then $\hat{\mathcal{M}}$ simply simulates the computations of $\mathcal{M}$. Each time a successor automaton $M_j$ is chosen in a computation of $\mathcal{M}$, one has to guess whether another cycle will be executed, and if so, which rewrite instruction will be applied, or whether the next component automaton will accept in a tail computation. Then in the simulating computation of $\mathcal{M}$, one must simply choose the corresponding component $M_j^{(0)}$ or $M_j^{(+)2}$. It follows easily that $L_{=1}(\hat{\mathcal{M}}) = L_{=1}(\mathcal{M})$.

In order to obtain the intended system in normal form, we delete all components $M_i^{(a)}$ satisfying $\delta_i^{(a)}(c) = \emptyset$, and we modify the accepting component automata $M_i^{(+)2}$ ($i \in I$). Actually we need to distinguish three cases.

If $\delta_i^{(+)}(c) = \text{Accept}$, then $M_i^{(+2)}$ will accept all words from $\Sigma^*$. Accordingly, we define $\delta_i^{(+)}$ as follows

$$
\delta_i^{(+)}(c) = \text{MVR}, \quad \delta_i^{(+)}(a) = \text{MVR} \quad \text{for all } a \in \Sigma, \quad \delta_i^{(+)}(S) = \text{Accept}.
$$

Then $M_i^{(+)}$ accepts all words from $\Sigma^*$, but it executes an accept instruction only on the $S$-symbol.

If $\delta_i^{(+)}(c) = \text{MVR}$, and $\delta_i^{(+)}(S)$ is undefined, then $M_i^{(+)}$ accepts all words from $\Sigma_1^{(i)\ast} \cdot \Sigma_3^{(i)} \cdot \Sigma^*$. Accordingly, we define $\delta_i^{(+)}$ as follows

$$
\delta_i^{(+)}(c) = \text{MVR},
\delta_i^{(+)}(a) = \text{MVR} \quad \text{for all } a \in \Sigma_1^{(i)},
\delta_i^{(+)}(a) = \varepsilon \quad \text{for all } a \in \Sigma_3^{(i)}.
$$

Also we define another component automaton $M_i^{n(+)}$ as follows

$$
\delta_i^{n(+)}(c) = \text{MVR},
\delta_i^{n(+)}(a) = \text{MVR} \quad \text{for all } a \in \Sigma,
\delta_i^{n(+)}(S) = \text{Accept},
$$

where $M_i^{n(+)}$ is the only successor of $M_i^{(+)}$. Then together they accept the same words as $M_i^{(+)}$, but an accept instruction is only executed on the $S$-symbol.

Finally, if $\delta_i^{(+)}(c) = \text{MVR}$, and $\delta_i^{(+)}(S) = \text{Accept}$, then $M_i^{(+)}$ accepts all words from $\Sigma_1^{(i)\ast} \cdot \Sigma_3^{(i)} \cdot \Sigma^* \cup \Sigma_1^{(i)\ast}$. Accordingly, we define $M_i^{(+)}$ and $M_i^{n(+)}$ as above, but we define a third component $\hat{M}_i^{(+)}$ as follows

$$
\delta_i^{(+)}(c) = \text{MVR},
\delta_i^{(+)}(a) = \text{MVR} \quad \text{for all } a \in \Sigma,
\delta_i^{(+)}(S) = \text{Accept}.
$$
δ_i^{(+)}(ε) = MVR, \\
δ_i^{(+)}(a) = MVR \text{ for all } a \in Σ_1^{(i)}, \\
δ_i^{(+)}(S) = \text{Accept.}

Then, in each successor set we replace M_i^{(+)} by both, M_i^{(+)}, and M_i^{(+)}, and take M_i^{(+)}, as the only successor of M_i^{(+)}.

Then together these three components accept the same words as M_i^{(+)}, but an accept instruction is only executed on the S-symbol.

Finally we again split each component automaton M_i^{(+)} that contains more than one rewrite instruction into several automata, one for each letter that is deleted by a rewrite instruction. Then, the resulting stl-det-local-CD-R(1)-system is in normal form, and in mode = 1 it accepts the same language as the original system M. □

Actually by again splitting the components M_i^{(+)}, M_i^{(+)}, and M_i^{(+)}, into corresponding subcomponents, we can even obtain the following strong normal form.

Remark 4.8. From a stl-det-local-CD-R(1)-system M one can construct a stl-det-local-CD-R(1)-system M' such that the system M' is in normal form, it has a single accepting component M_+ only that only accepts the empty word, and L_{=1}(M') = L_{=1}(M).

Thus, in M' there is only a single component M_+ that contains an Accept instruction, and the transition function of M_+ is defined by δ_+(ε) = MVR and δ_+(S) = Accept.

We have seen that the language class L_{=1}(stl-det-local-CD-R(1)) contains all regular languages and some languages that are not even context-free. Our next result implies that all languages from this class are semi-linear, that is, if L ∈ Σ* belongs to this language class, and if |Σ| = n, then the Parikh image ψ(L) of L is a semi-linear subset of N^n.

Theorem 4.9. Each language L ∈ L_{=1}(stl-det-local-CD-R(1)) contains a regular sublanguage E such that ψ(L) = ψ(E) holds. In fact, a finite-state acceptor for E can be constructed effectively from a stl-det-local-CD-R(1)-system for L.

Proof. Let M = ((M_i, σ_i)_{i∈I}, I_0) be a stl-det-local-CD-R(1)-system over Σ, and let L = L_{=1}(M). By Lemma 4.7 we can assume that M is in normal form. From M we construct a nondeterministic finite-state acceptor (NFA) A over Σ such that the language L(A) is letter-equivalent to L. Essentially A behaves like M without being able to read across translucent letters.

For each index i ∈ I, let M_i = (Σ, Q_i, δ_i), and let Σ = Σ_1^{(i)} ∪ Σ_2^{(i)} ∪ Σ_3^{(i)} ∪ Σ_4^{(i)} be the partitioning of Σ associated with M_i (see Definition 2.3). As M is in normal form, we see that Σ_3^{(i)} = Σ_1^{(i)} = ∅ and Σ_2^{(i)} = Σ_4^{(i)} = ∅ for each i ∈ I. Further, we know that δ_i(ε) = MVR, and we can assume that δ_i(S) = Accept if and only if Σ_2^{(i)} = ∅.

We now define the announced NFA A = (Q, Σ, q_0, F, δ_A) as follows:

• The set of states Q and the set of final states F are defined by
  
  Q = I ∪ {q_0} ∪ {q_Δ | Δ ⊆ Σ} \text{ and } F = {q_Δ | Δ ⊆ Σ},

  that is, for each component automaton M_i, A has a particular state i, it has initial state q_0, and for each subalphabet Δ of Σ, it has an accepting state q_Δ.

• The transition relation δ_A is defined by:

  1. δ_A(q_0, ε) = I_0,

  2. δ_A(i, a) = σ_i \text{ for all } i ∈ I \text{ such that } a ∈ Σ_2^{(i)},

  3. δ_A(i, ε) = [q_{Σ_1^{(i)}}] \text{ for all } i ∈ I \text{ such that } δ_i(S) = \text{Accept},

  4. δ_A(q_Δ, a) = [q_Δ] \text{ for all } Δ ⊆ Σ \text{ and } a ∈ Δ.

Then A is an NFA with ε-transitions that is easily constructed from M. Hence, L(A) is a regular language over Σ. It remains to prove that L(A) is a sublanguage of the language L = L_{=1}(M) that is letter-equivalent to L. We first establish the following related technical result.

Claim 1. If w = w_0 ⊢_M i_1, w_1 ⊢_M i_2, ..., w_k ⊢_M i_k, w_k ⊢_M i_{k+1} \text{ Accept is a mode = 1 computation of } M, \text{ then there exists a word } z ∈ Σ^* \text{ such that } i_1z ⊢_A q ∈ F \text{ holds, and } ψ(z) = ψ(w).
Thus, \( z \) follows that there exists an accepting mode \( M = \contents \) and \( \delta_1(\$) = \text{Accept} \). Hence, \( A \) can perform the following computation:

\[
i_1w \vdash (3)_A q_{\Sigma(z)}w \vdash (4)_A q_{\Sigma(z)} \in F.\]

Thus, \( A \) accepts starting from \( i_1w \).

If \( w = xay \vdash_{i_1} x_{M_1} y \), then \( x \in \Sigma^{(i_1)*}, \, a \in \Sigma^{(i_1)} \), and \( i_2 \in \sigma_{i_1} \). Thus, \( A \) can perform the following step:

\[
i_1axy \vdash (2)_A i_2xy.\]

From the induction hypothesis we see that there exists a word \( z_1 \in \Sigma^* \) that is accepted by \( A \) starting from the configuration \( i_2z_1 \), and that is letter-equivalent to \( w_1 = xy \). Hence, the word \( z = az_1 \) is accepted by \( A \) starting from the configuration \( i_1az_1 \), and \( az_1 \) is letter-equivalent to \( axy \) and therewith to \( w = xay \). This completes the proof of Claim 1. \( \Box \)

If \( w \in L_{=1}(M) \), then there exists an accepting mode \( = 1 \) computation of \( M \) of the following form:

\[
w = w_0 \vdash_{M_1} \cdots \vdash_{M_i} \vdash_{M_{i+1}} w_{i+1} \vdash_{M_{i+1}} \text{Accept}.\]

Then \( i_1 \in I_0 \), and from Claim 1 we see that there exists a word \( z \in \Sigma^* \) such that \( z \) is letter-equivalent to \( w \), and \( A \) accepts starting from the configuration \( i_1z \). But then \( i_1 \in \delta_{I}(q_0, \varepsilon) \) implies that \( A \) accepts starting from the initial configuration \( q_0z \). Thus, \( z \in L(A) \), that is, for each word \( w \in L_{=1}(M) \), there exists a word \( z \in L(A) \) such that \( z \) and \( w \) are letter-equivalent.

The proof of Theorem 4.9 is now completed by establishing the following claim.

**Claim 2.** If \( z \in \Sigma^* \) and \( i \in I \) such that \( A \) accepts starting from the configuration \( iz \), then \( M \) has an accepting mode \( = 1 \) computation in which component automaton \( M_i \) starts with the initial tape contents \( \varepsilon z \$ \).

**Proof.** We proceed by induction on the number of steps of group (2) that are applied in the accepting computation of \( A \).

If no such step is applied at all, then the accepting computation of \( A \) has the following form:

\[
i_z \vdash (3)_A q_{\Sigma(z)}z \vdash (4)_A q_{\Sigma(z)} \in F.\]

From the definition of \( A \) we see that \( \delta_i(\$) = \text{Accept} \), and hence, component automaton \( M_i \) will accept starting from the tape contents \( \varepsilon z \$ \).

Now assume that the accepting computation of \( A \) looks as follows

\[
i_z = iav \vdash (2)_A jv \vdash_{A} q_\Delta,\]

where \( a \in \Sigma \), and \( \Delta \subseteq \Sigma \). From the definition of \( A \) we see that \( \delta_i(a) = \varepsilon \), and that \( j \in \sigma_i \). Further, from the induction hypothesis we know that \( M \) has an accepting mode \( = 1 \) computation in which \( M_i \) starts with the tape contents \( \varepsilon v \$ \). It follows that there exists an accepting mode \( = 1 \) computation of \( M \) in which \( M_i \) starts with tape contents \( \varepsilon av \$ = \varepsilon z \$ \). \( \Box \)

It follows that each word \( z \in L(A) \) belongs to the language \( L_{=1}(M) \). Thus, \( L(A) \) is indeed a regular sublanguage of \( L \) that is letter-equivalent to \( L \). \( \Box \)

As all regular languages have semi-linear Parikh image, this yields the following important result.

**Corollary 4.10.** The language class \( L_{\leq1}(\text{stl-det-local-CD-R}(1)) \) only contains semi-linear languages, that is, if a language \( L \) over \( \Sigma = \{a_1, \ldots, a_n\} \) is accepted by a CD-system of stateless deterministic \( R \)-automata of window size 1, then its Parikh image \( \psi(L) \) is a semi-linear subset of \( \mathbb{N}^n \).

As the deterministic linear language \( L = \{a^n b^n \mid n \geq 0\} \) does not contain a regular sublanguage that is letter-equivalent to the language itself, we obtain the following non-inclusion result.

**Proposition 4.11.** The language \( L = \{a^n b^n \mid n \geq 0\} \) is not accepted by any stl-det-local-CD-R(1)-system working in mode \( = 1 \).

It follows analogously that the language \( L_3 = \{a^n b^n c^n \mid n \geq 0\} \) is not accepted by any stl-det-local-CD-R(1)-system working in mode \( = 1 \). As \( L_3 = L_{abc} \cap a^* \cdot b^* \cdot c^* \), this implies the following in combination with Proposition 4.2.

**Corollary 4.12.** The language class \( L_{\leq1}(\text{stl-det-local-CD-R}(1)) \) is not closed under intersection with regular languages.
Corollary 4.13. The language class $L =_{\text{stl-det-CD-R(1)}}$ is incomparable to the classes DLIN, LIN, DCFL, and CFL with respect to inclusion.

Here LIN denotes the class of linear languages and DLIN the class of deterministic linear languages, that is, the class of languages that are accepted by deterministic one-turn pushdown automata (see, e.g., [1]).

Lemma 4.7 suggests to describe CD-systems of stateless deterministic $R$-automata of window size 1 by a graphical representation.

Let $M = (\langle M_i, \sigma_i \rangle)_{i \in I}, I_0)$ be a stl-det-CD-R(1)-system in normal form on alphabet $\Sigma$, and for each $i \in I$, let $\Sigma = \Sigma_1^{(i)} \cup \Sigma_2^{(i)} \cup \Sigma_3^{(i)} \cup \Sigma_4^{(i)}$ be the partitioning of $\Sigma$ associated with $M_i$ (see Definition 2.3). Then we can describe $M$ by a diagram that contains a vertex for each component automaton $M_i$ and a special vertex “Accept”. For all $i \in I$, if $\delta_i(\$) = Accept, then $M_i$ accepts all words from $\Sigma_1^{(i)}$, and accordingly, there only is an edge labelled $\varepsilon \cdot \Sigma_1^{(i)} \cdot \$ from vertex $i$ to vertex “Accept” (see Fig. 2). On the other hand, if $\delta_i(\varepsilon) = \varepsilon$, then $M_i$ deletes the leftmost occurrence of the letter $a$, provided it is preceded only by a word over $\Sigma_1^{(i)}$. Accordingly, there is an edge labelled $\epsilon \cdot \Sigma_1^{(i)} \cdot a$ from vertex $i$ to vertex $j$ for all $j \in \sigma_i$ (see Fig. 3). Finally, vertex $i$ is specifically marked for all initial indices $i \in I_0$. We illustrate this way of describing stl-det-CD-R(1)-systems by an example.

Example 4.14. Let $M = (\langle M_i, \sigma_i \rangle)_{i \in I}, I_0)$ be the following system, where $I = \{1, 1', 2, 3, 4, +\}$, $I_0 = \{1\}$, $\sigma_1 = \{1', 2\}$, $\sigma_1' = \{1, 2\}$, $\sigma_2 = \{3\}$, $\sigma_3 = \{4\}$, $\sigma_4 = \{2, +\}$, $\sigma_+ = \{4\}$, and the various $R$-automata are given by the following transition functions:

$$
\begin{align*}
M_1: & \quad \delta_1(\epsilon) = \text{MVR}, \\
& \quad \delta_1(\varepsilon) = \varepsilon, \\
M_2: & \quad \delta_2(\epsilon) = \text{MVR}, \\
& \quad \delta_2(\varepsilon) = \varepsilon, \\
M_3: & \quad \delta_3(\epsilon) = \text{MVR}, \\
& \quad \delta_3(\varepsilon) = \varepsilon, \\
M_4: & \quad \delta_4(\epsilon) = \text{MVR}, \\
& \quad \delta_4(\varepsilon) = \text{MVR}.
\end{align*}
$$

Then using the component automata $M_1$ and $M_1'$, $M$ deletes a positive number of a's, and then using component automata $M_2$, $M_3$, and $M_4$ it deletes an equal number of a's, b's, and c's, before it accepts the empty word by component automaton $M_+$. Thus,

$$
L_{\text{stl-det-CD-R(1)}} = \{a^n w \mid n \geq 1, w \in \{a, b, c\}^* \text{ satisfying } |w|_a = |w|_b = |w|_c \}.
$$

Now this CD-system of stateless $R$-automata of window size 1 can be described more compactly by the diagram given in Fig. 4.
As another example, we consider the language \( L_2 = \{ wan \mid n \geq 1, w \in \{a, b, c\}^+ \text{ satisfying } |w|_a = |w|_b = |w|_c \} \).

**Example 4.15.** Let \( M \) be the CD-system of stateless deterministic R-automata of window size 1 that is described by the diagram in Fig. 5.

The system \( M \) consists of 23 component automata, 7 of which are initial automata. Automaton \( M_{23} \) is the only one with an accept instruction. It accepts the language \( a^* \). Accordingly, computations that begin with the initial automaton \( M_4 \) accept the language \( abc \cdot a^+ \), and analogously for those computations that begin with \( M_5, M_6, M_7, M_8 \) or \( M_9 \). It follows that in combination these computations accept the language \( L'_2 = \{ wan \mid n \geq 1, w \in \{a, b, c\}^+ \text{ satisfying } |w|_a = |w|_b = |w|_c = 1 \} \).

An accepting computation that begins with the initial automaton \( M_1 \) consists of two parts: first it cycles through the automata \( M_1, M_2, \) and \( M_3, \) in each round deleting the first \( a, b, \) and \( c \) from the left, and then it continues with a computation that accepts a word \( w_1 \cdot a^n \) from \( L'_2 \), where \( |w_1|_a = |w_1|_b = |w_1|_c = 1 \). Since at that moment there is at most a single \( a \) to the left of the last remaining letters \( b \) and \( c \), it follows that all deletions in the first phase of this computation where executed to the left of the suffix \( a^n \). Hence, the input \( w \) does indeed belong to the language \( L_2 \), which implies that \( L_{\text{in}}(M) = L_2 \) holds.

The diagrams in Fig. 4 and in Fig. 5 can also be interpreted as descriptions of NFAs with translucent letters. Each vertex represents a state, with the vertex labelled “Accept” being the unique final state. Further, an edge labelled \((\epsilon \cdot \Sigma^*, x)\) denotes a transition that reads (and removes) the first occurrence of the letter \( x \), provided it is only preceded by occurrences of letters from \( \Sigma \), which are interpreted as being translucent for this transition. Finally, an edge labelled \((\epsilon \cdot \Sigma^* \cdot \$)\) denotes a transition to a subautomaton that accepts all words from \( \Sigma^* \). Also a variant of the binding-blocking automaton of Balan [2] can be used to model this behaviour. Each transition of the form \((\epsilon \cdot \Sigma^* , x)\) would be realized by first blocking the longest prefix of the current word that belongs to \( \Sigma^* \), then the letter \( x \) would be read, and then the factor from \( \Sigma^* \) would be unblocked again. Thus, our CD-systems of stateless deterministic R(1)-automata can be interpreted as a very special variant of these binding-blocking automata.

**5. Rational trace languages**

Let \( \Sigma \) be a finite alphabet, and let \( D \) be a binary relation on \( \Sigma \) that is reflexive and symmetric, that is, \((a,a) \in D\) for all \( a \in \Sigma \), and \((a, b) \in D\) implies that \((b, a) \in D\), too. Then \( D \) is called a dependency relation on \( \Sigma \), and the relation...
$I_D = (\Sigma \times \Sigma)^{\ast} \setminus D$ is called the corresponding independence relation. Obviously, the relation $I_D$ is irreflexive and symmetric. The dependency relation $D$ (or rather its associated independence relation $I_D$) induces a binary relation $\equiv_D$ on $\Sigma^\ast$ that is defined as the smallest congruence relation containing the set of pairs $\{(ab, ba) \mid (a, b) \in I_D\}$. For $w \in \Sigma^\ast$, the congruence class of $w$ mod $\equiv_D$ is denoted by $[w]_D$, that is, $[w]_D = \{z \in \Sigma^\ast \mid w \equiv_D z\}$. These equivalence classes are called traces, and the factor monoid $M(D) = \Sigma^\ast \equiv_D$ is a trace monoid. In fact, $M(D)$ is the free partially commutative monoid presented by $(\Sigma, D)$ (see, e.g., [9]). By $\varphi_D$ we denote the morphism $\varphi_D : \Sigma^\ast \rightarrow M(D)$ that is defined by $w \mapsto [w]_D$ for all words $w \in \Sigma^\ast$.

To simplify the notation in what follows, we introduce the following notions. For $w \in \Sigma^\ast$, we use $\text{Alph}(w)$ to denote the set of all letters that occur in $w$, that is,

$$\text{Alph}(w) = \{a \in \Sigma \mid |w|_a > 0\}.$$ 

Now we extend the independence relation from letters to words by defining, for all words $u, v \in \Sigma^\ast$,

$$(u, v) \in I_D \text{ if and only if } \text{Alph}(u) \times \text{Alph}(v) \subseteq I_D.$$ 

As $\text{Alph}(\varepsilon) = \emptyset$, we see that $(\varepsilon, w) \in I_D$ for every word $w \in \Sigma^\ast$. The following technical result (see, e.g., [9, Claim A in the proof of Proposition 6.2.2]) will be useful in what follows.

**Proposition 5.1.** For all words $x, y, u, v \in \Sigma^\ast$ and all letters $a \in \Sigma$, if $xay \equiv_D au$ and $|x|_a = 0$, then $(a, x) \in I_D$, $xay \equiv_D axy$, and $xy \equiv_D u$.

A subset $S$ of a trace monoid $M(D)$ is called recognizable if there exist a finite monoid $N$, a morphism $\alpha : M(D) \rightarrow N$, and a subset $P$ of $N$ such that $S = \alpha^{-1}(P)$ (see, e.g., [3]). Accordingly, this property can be characterized as follows (see [9, Proposition 6.110]).

**Proposition 5.2.** Let $M(D)$ be the trace monoid presented by $(\Sigma, D)$, and let $\varphi_D : \Sigma^\ast \rightarrow M(D)$ be the corresponding morphism. Then a set $S \subseteq M(D)$ is recognizable if and only if the language $\varphi_D^{-1}(S)$ is a regular language over $\Sigma$.

By $\text{REC}(M(D))$ we denote the set of recognizable subsets of $M(D)$.

A subset $S$ of a trace monoid $M(D)$ is called rational if it can be obtained from singleton sets by a finite number of unions, products, and star operations (see, e.g., [3]). This property can be characterized more conveniently as follows.

**Proposition 5.3.** Let $M(D)$ be the trace monoid presented by $(\Sigma, D)$, and let $\varphi_D : \Sigma^\ast \rightarrow M(D)$ be the corresponding morphism. Then a set $S \subseteq M(D)$ is rational if and only if there exists a regular language $R$ over $\Sigma$ such that $S = \varphi_D(R)$.

By $\text{RAT}(M(D))$ we denote the set of rational subsets of $M(D)$. Concerning the relationship between the recognizable subsets of $M(D)$ and the rational subsets of $M(D)$ the following results are known (see, e.g., [9]).

**Proposition 5.4.** $\text{REC}(M(D)) \subseteq \text{RAT}(M(D))$ for each trace monoid $M(D)$, and these two sets are equal if and only if $I_D = \emptyset$.

Thus, each recognizable subset of a trace monoid $M(D)$ is necessarily rational, but the converse only holds if $I_D$ is empty, that is, if $D = \Sigma \times \Sigma$, which means that the congruence $\equiv_D$ is the identity. Thus, the free monoids are the only trace monoids for which the recognizable subsets coincide with the rational subsets.

We call a language $L \subseteq \Sigma^\ast$ a rational trace language, if there exists a dependency relation $D$ on $\Sigma$ such that $L = \varphi_D^{-1}(S)$ for a rational subset $S$ of the trace monoid $M(D)$ presented by $(\Sigma, D)$. From Proposition 5.3 it follows that $L$ is a rational trace language if and only if there exist a trace monoid $M(D)$ and a regular language $R \subseteq \Sigma^\ast$ such that $L = \varphi_D^{-1}(\varphi_D(R)) = \bigcup_{w \in R} [w]_D$. By $\text{LRAT}(M(D))$ we denote the set of rational trace languages $\varphi_D^{-1}(\text{RAT}(M(D)))$, and $\text{LRAT}$ is the class of all rational trace languages. The next theorem states that all these languages are accepted by stl-det-local-CD-R(1)-systems.

**Theorem 5.5.** Let $M(D)$ be the trace monoid presented by $(\Sigma, D)$, where $D$ is a dependency relation on the finite alphabet $\Sigma$. Then

$$\text{LRAT}(M(D)) \subseteq \mathcal{L}_{=1}(\text{stl-det-local-CD-R}(1)),$$

that is, the language $\varphi_D^{-1}(S)$ is accepted by a stl-det-local-CD-R(1)-system working in mode $= 1$ for each rational set $S \subseteq M(D)$ of traces.

**Proof.** Let $S$ be a rational subset of $M(D)$. Then there exists a regular language $R$ over $\Sigma$ such that $S = \varphi_D(R)$. As $R \subseteq \Sigma^\ast$ is a regular language, there exists a complete deterministic finite-state acceptor $A = (Q, \Sigma, P_0, F, \delta)$ for $R$. From $A$ we now construct a stl-det-local-CD-R(1)-system $M = ((M_i, \sigma_i)_{i \in I}, I_D)$ as follows (cf. the proof of Proposition 4.4):
the set of indices is \( I = (Q \times \Sigma) \cup (Q' \times \Sigma) \cup \{+, 0\} \), where \( Q' = \{ q' \mid q \in Q \} \) is a copy of \( Q \) such that \( Q \cap Q' = \emptyset \),

- the set of initial indices is
  \[
  I_0 = \left\{ (p_0, a) \mid a \in \Sigma \right\}, \quad \text{if } e \notin L,
  \]
  \[
  \left\{ (p_0, a) \mid a \in \Sigma \cup \{+, 0\} \right\}, \quad \text{if } e \in L,
  \]

- the successor relations are defined by
  \[
  \sigma_{(q, a)} = \left\{ \begin{array}{ll}
  \{(\delta(q, a), b) \mid b \in \Sigma \cup \{+, 0\}\}, & \text{if } \delta(q, a) = q' \text{ and } \delta(q, a) \in F, \\
  \{(\delta(q, a), b) \mid b \in \Sigma\}, & \text{if } \delta(q, a) = q' \text{ and } \delta(q, a) \notin F,
  \end{array} \right.
  \]
  \[
  \sigma_{(q', a)} = \left\{ \begin{array}{ll}
  \{\delta(q', a), b) \mid b \in \Sigma \cup \{+, 0\}\}, & \text{if } \delta(q', a) = q \text{ and } \delta(q', a) \in F, \\
  \{(\delta(q', a), b) \mid b \in \Sigma\}, & \text{if } \delta(q', a) = q \text{ and } \delta(q', a) \notin F,
  \end{array} \right.
  \]
  \[
  \sigma_+ = \{ (p_0, a) \mid a \in \Sigma \},
  \]

- and the stl-det-R(1)-automata \( M_{(q, a)} \), \( M_{(q', a)} \), and \( M_+ \) are defined by the following transition functions:

  \[
  M_{(q, a)} : \begin{align*}
  \delta_{(q, a)}(e) &= \text{MVR}, \\
  \delta_{(q, a)}(b) &= \text{MVR} \quad \text{for all } b \in \Sigma \text{ satisfying } (b, a) \in I_D, \\
  \delta_{(q, a)}(a) &= e, \\
  \end{align*}
  \]

  \[
  M_{(q', a)} : \begin{align*}
  \delta_{(q', a)}(e) &= \text{MVR}, \\
  \delta_{(q', a)}(b) &= \text{MVR} \quad \text{for all } b \in \Sigma \text{ satisfying } (b, a) \in I_D, \\
  \delta_{(q', a)}(a) &= e, \\
  \end{align*}
  \]

  \[
  M_+ : \begin{align*}
  \delta_+(e) &= \text{MVR}, \\
  \delta_+(S) &= \text{Accept}.
  \end{align*}
  \]

It remains to show that \( L_{=1}(M) = \varphi^{-1}_D(S) = \bigcup_{u \in R}[u]D \).

**Claim 1.** \( \bigcup_{u \in R}[u]D \subseteq L_{=1}(M) \).

**Proof.** Assume that \( w \in \bigcup_{u \in R}[u]D \). Then there exists a word \( u \in R \) such that \( w \equiv_D u \), and so there exists a sequence of words \( u = w_0, w_1, \ldots, w_n = w \) such that, for each \( i = 1, \ldots, n \), \( w_i \) is obtained from \( w_{i-1} \) by replacing a factor \( ab \) by \( ba \) for some pair of letters \( (a, b) \in I_D \). We now prove that \( w_i \in L_{=1}(M) \) for all \( i \) by induction on \( i \).

For \( i = 0 \) we have \( w_0 = u \in R \). Thus, \( w_0 \) is accepted by the finite-state acceptor \( A \), and it follows from the proof of Proposition 4.4 that \( w_0 \) is also accepted by a mode = 1 computation of \( M \).

Now assume that \( w_i \in L_{=1}(M) \) for some \( i > 0 \), and that \( w_i = xaby \) and \( w_{i+1} = xbay \) for a pair of letters \( (a, b) \in I_D \). By our hypothesis \( M \) has an accepting mode = 1 computation for \( w_i = xaby \), which is of one of the following two forms:

\[
w_i = xaby \vdash_{M_{(q, a)}} x'y \vdash_{M_{(q', a)}} x'y' \vdash_{M_+} e \vdash_{M_+} \text{Accept},
\]

or

\[
w_i = xaby \vdash_{M_{(q, a)}} x'y \vdash_{M_{(q', a)}} x'y' \vdash_{M_+} e \vdash_{M_+} \text{Accept},
\]

where in the first \( m \) cycles some letters from \( x \) and \( y \) are deleted, in this way reducing these factors to \( x' \) and \( y' \), respectively, and \( q \in Q \cup Q' \) is a state (or a copy of a state) of \( A \). However, as \( (a, b) \in I \), the component automaton \( M_{(q, a)} \) (or \( M_{(q', a)} \)) can read across the letter \( b \) (or \( a \)) when looking for the leftmost occurrence of the letter \( a \) (or \( b \)). Thus, \( M \) also has an accepting mode = 1 computation for \( w_{i+1} = xbay \), which is of one of the following two forms:

\[
w_{i+1} = xbay \vdash_{M_{(q, a)}} x'y \vdash_{M_{(q', a)}} x'y' \vdash_{M_+} e \vdash_{M_+} \text{Accept},
\]

or

\[
w_{i+1} = xbay \vdash_{M_{(q, a)}} x'y \vdash_{M_{(q', a)}} x'y' \vdash_{M_+} e \vdash_{M_+} \text{Accept},
\]

implying that \( w_{i+1} \in L_{=1}(M) \). This completes the proof of Claim 1. \( \square \)

**Claim 2.** \( L_{=1}(M) \subseteq \bigcup_{u \in R}[u]D \).
Proof. Let \( w \in L_{=1}(\mathcal{M}) \), and let

\[
\begin{align*}
  w &= w_n \vdash_{M_{\mathcal{M}(q_n, a_n)}}^c \cdots \vdash_{M_{\mathcal{M}(q_2, a_2)}}^c \vdash_{M_{\mathcal{M}(q_1, a_1)}}^c W_0 = \varepsilon \vdash_{M_0}^c \text{ Accept}
\end{align*}
\]

be an accepting mode = 1 computation of \( \mathcal{M} \) on input \( w \), where \( q_n, q_{n-1}, \ldots, q_1 \) are states of \( A \) (or copies thereof) and \( (q_n, a_n) \in I_0 \). We claim that, for each \( i = 1, \ldots, n \), there exists a word \( u_i \in \Sigma^* \) such that \( u_i \equiv_D w_i \) and \( \delta(q_i, u_i) \in F \), that is, the finite-state acceptor \( A \) accepts the word \( u_i \) when starting from state \( q_i \).

We prove this claim by induction on \( i \). For \( i = 1 \) we have \( w_1 = a_1 \), and + \( \in \sigma(q_1, a_1) \). From the definition of \( \mathcal{M} \) we conclude that \( \delta(q_1, a_1) \in F \), that is, we can simply take \( u_1 = a_1 = w_1 \). Now assume that, for some \( i \geq 1 \), \( u_i \equiv_D w_i \) and \( \delta(q_i, u_i) \in F \) hold.

The above computation of \( \mathcal{M} \) contains the cycle \( w_{i+1} \vdash_{M_{\mathcal{M}(q_{i+1}, a_{i+1})}}^c w_i \) and \( (q_i, a_i) \in \sigma(q_{i+1}, a_{i+1}) \). Again from the definition of \( \mathcal{M} \) we see that \( \delta(q_{i+1}, a_{i+1}) = q_i \), and that \( w_{i+1} = xa_{i+1}y \) and \( w_i = xy \) for some words \( x, y \in \Sigma^* \) such that \( (x, a_{i+1}) \in I_0 \). Let \( u_{i+1} \) be the word \( u_{i+1} = a_{i+1}u_i \). Then

\[
\begin{align*}
  u_{i+1} &= a_{i+1}u_i \\
  &\equiv_D a_{i+1}w_i = a_{i+1}xy \\
  &\equiv_D xa_{i+1}y = w_{i+1},
\end{align*}
\]

and \( \delta(q_{i+1}, u_{i+1}) = \delta(q_{i+1}, a_{i+1}u_i) = \delta(\delta(q_{i+1}, a_{i+1}), u_i) = \delta(q_i, u_i) \in F \).

For \( i = n \) we obtain a word \( u \) in \( \Sigma^* \) such that \( u \equiv_D w \), and \( A \) accepts \( u \) starting from state \( q_n = p_0 \). Hence, \( u \in R \), and it follows that \( L_{1}(\mathcal{M}) \subseteq \bigcup_{u \in R}[u]D \) holds.

Now Claims 1 and 2 together show that \( L_{=1}(\mathcal{M}) = \bigcup_{u \in R}[u]D = \varphi^{-1}_D(S) \), which completes the proof of Theorem 5.5.

Observe that the Dyck language \( D^*_+ \) is not a rational trace language. Thus, the language class \( L_{=1}(\text{stl-det-local CD-R}(1)) \) is a proper superclass of the class of all rational trace languages. Next we present a restricted class of \( \text{stl-det-local CD-R}(1) \)-systems that accept exactly the rational trace languages by mode = 1 computations.

Definition 5.6. Let \( \mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0) \) be a \( \text{stl-det-local CD-R}(1) \)-system in normal form on \( \Sigma \) that satisfies the following condition:

\[ (*) \quad \forall i, j \in I : \Sigma_2^{(i)} = \Sigma_2^{(j)} \text{ implies that } \Sigma_1^{(i)} = \Sigma_1^{(j)}, \]

that is, if two component automata erase the same letter, then they also read across the same subset of \( \Sigma \), which means that they have the same set of translucent letters. With \( \mathcal{M} \) we associate a binary relation

\[ I_\mathcal{M} = \bigcup_{i \in I} (\Sigma_1^{(i)} \times \Sigma_2^{(i)}), \]

that is, \( (a, b) \in I_\mathcal{M} \) if and only if there exists a component automaton \( M_i \) such that \( \delta_i(a) = MVR \) and \( \delta_i(b) = \varepsilon \). Further, by \( D_\mathcal{M} \) we denote the relation \( D_\mathcal{M} = (\Sigma \times \Sigma) \setminus I_\mathcal{M} \).

Observe that the relation \( I_\mathcal{M} \) defined above is necessarily irreflexive, but that it will in general not be symmetric. For example, consider the system \( \mathcal{M} \) from the proof of Proposition 4.1. It is in normal form, but the corresponding relation \( I_\mathcal{M} = \{(a, b)\} \) is not symmetric. And indeed, the language \( L_{=1}(\mathcal{M}) \) is the Dyck language \( D^*_+ \), which is not a rational trace language.

Theorem 5.7. Let \( \mathcal{M} \) be a \( \text{stl-det-local CD-R}(1) \)-system above satisfying condition \( (*) \) above. If the associated relation \( I_\mathcal{M} \) is symmetric, then \( L_{=1}(\mathcal{M}) \) is a rational trace language over \( \Sigma \). In fact, from \( \mathcal{M} \) one can construct a finite-state acceptor \( B \) over \( \Sigma \) such that \( L_{=1}(\mathcal{M}) = \varphi^{-1}_B(\varphi_{D_\mathcal{M}}(L(B))) \).

Proof. Let \( \mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0) \) be a \( \text{stl-det-local CD-R}(1) \)-system in normal form on \( \Sigma \) that satisfies condition \( (*) \), and assume that the associated relation \( I_\mathcal{M} = \bigcup_{i \in I} (\Sigma_1^{(i)} \times \Sigma_2^{(i)}) \) is symmetric. Then the relation \( D_\mathcal{M} = (\Sigma \times \Sigma) \setminus I_\mathcal{M} \) is reflexive and symmetric, and so it is a dependency relation on \( \Sigma \) with associated independence relation \( I_\mathcal{M} \). Without loss of generality we may assume that all letters from \( \Sigma \) do actually occur in some words of \( L_{=1}(\mathcal{M}) \), since otherwise we could simply remove these letters from \( \Sigma \). In addition, by Remark 4.8 we can assume that \( \mathcal{M} \) has only a single accepting component automaton \( M_+ \), and that \( M_+ \) only accepts the empty word. From the properties of \( \mathcal{M} \) we obtain the following consequences:

1. As all words \( w \in L_{=1}(\mathcal{M}) \) are first reduced to the empty word, which is then accepted by the accepting component automaton of \( \mathcal{M} \), we see that, for each letter \( a \in \Sigma \), there exists a component automaton \( M_i \) such that \( \Sigma_2^{(i)} = \{a\} \).
2. If \( (a, b) \in I_\mathcal{M} \), then \( a \in \Sigma_1^{(i)} \) for all component automata \( M_i \) for which \( \Sigma_2^{(i)} = \{b\} \) holds (by \( (*) \)).
3. If \( (a, b) \in I_\mathcal{M} \), then \( (b, a) \in I_\mathcal{M} \), too, and hence, \( b \in \Sigma_1^{(i)} \) for all component automata \( M_i \) for which \( \Sigma_2^{(i)} = \{a\} \) holds (by \( (*) \)).
Let \( L = L_1(\mathcal{M}) \). We claim that \( L \) is a rational trace language over the trace monoid defined by \((\Sigma, D_M)\), that is, \( L \in CRAT(D_M) \). To verify this claim we present a regular language \( R \subseteq \Sigma^* \) such that \( L = \bigcup_{u \in R} [u]_{D_M} \).

The regular language \( R \) will be defined through a nondeterministic finite-state acceptor (with \( \varepsilon \)-moves) \( B = (Q, \Sigma, p_0, p_+, \delta) \), where \( Q \) is a finite set of states, \( p_0 \in Q \) is the initial state, \( p_+ \in Q \) is the only final state, and \( \delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times Q) \) is the transition relation. This finite-state acceptor is obtained from \( \mathcal{M} \) as follows. Here \( I_+ = I \setminus \{+\} \) is the subset of \( I \) containing all component automata that perform a rewrite operation, \( i \in I_+ \), and \( a \in \Sigma \):

\[
Q = \{p_0, p_+\} \cup \{q_i \mid i \in I_+\},
\]

\[
\delta(p_0, \varepsilon) = \{q_i \mid i \in I_+\}, \quad \text{if } + \notin I_0,
\]

\[
\delta(q_i, \varepsilon) = \{q_j \mid i \in I_0 \cap I_+\} \cup \{p_+\}, \quad \text{if } + \in I_0,
\]

\[
\delta(q_j, a) = \{q_j \mid j \in \sigma_i\}, \quad \text{if } [a] = \Sigma_2^{(i)} \text{ and } + \notin \sigma_i,
\]

\[
\delta(q_j, a) = \{q_j \mid j \in \sigma_i \cap I_+\} \cup \{p_+\}, \quad \text{if } [a] = \Sigma_2^{(i)} \text{ and } + \in \sigma_i,
\]

\[
\delta(q, a) = \emptyset \quad \text{for all other cases}.
\]

Now \( R = L(B) \) is the announced regular language over \( \Sigma \). It remains to prove that \( L = \bigcup_{u \in R} [u]_{D_M} \) holds.

**Claim 1.** \( \bigcup_{u \in R} [u]_{D_M} \subseteq L \).

**Proof.** First we show that \( R \subseteq L \) holds. Indeed if we remove all MVR-operations that read across letters of \( \Sigma \) from all the rewriting component automata of \( \mathcal{M} \), then we obtain a stl-det-local-CD-R(1)-system \( \mathcal{M}' \) that deletes a word letter by letter from left to right. Now the finite-state acceptor \( B \) simply simulates the system \( \mathcal{M}' \), which implies that

\[
R = L(B) = L_1(\mathcal{M}') \subseteq L_1(\mathcal{M}) = L
\]

holds.

Let \( w \equiv_{D_M} u \in R \), and let \( w = w_0, w_1, \ldots, w_n = w \) be a sequence of words such that, for each \( i = 1, \ldots, n \), \( w_i \) is obtained from \( w_{i-1} \) by replacing a factor \( ab \) by \( ba \) for some pair of letters \( (a, b) \in I_M \). We now prove that \( w_i \in L \) for all \( i \) by induction on \( i \).

For \( i = 0 \) we have \( w_0 = u \in R \), and so \( w_0 \in L \) by the considerations in the previous paragraph. Now assume that \( w_i \in L \) for some \( i \geq 0 \), and that \( w_i = xaby \) and \( w_{i+1} = xbay \) for a pair of letters \( (a, b) \in I_M \). By our hypothesis \( \mathcal{M} \) has an accepting mode \( = 1 \) computation for \( w_i = xaby \), which is of one of the following two forms:

\[
w_i = xaby \vdash_{\mathcal{M}} x_1aby_1 \vdash_{M_1} x_1by_1 \vdash_{M_2} x_2by_2 \vdash_{M_j} x_2y_2 \vdash_{\mathcal{M}} \text{Accept},
\]

or

\[
w_i = xaby \vdash_{\mathcal{M}} x_1aby_1 \vdash_{M_j} x_1ay_1 \vdash_{M_2} x_2ay_2 \vdash_{M_j} x_2y_2 \vdash_{\mathcal{M}} \text{Accept},
\]

where in the first \( k \) cycles some letters from \( x \) and \( y \) are deleted, in this way reducing these factors to \( x_1 \) and \( y_1 \), respectively, \( \Sigma_2^{(i)} = \{a\} = \Sigma_2^{(f)} \) and \( \Sigma_2^{(j)} = \{b\} = \Sigma_2^{(f)} \), and in the latter \( l \) cycles some letters from \( x_1 \) and \( y_1 \) are deleted, reducing these factors to \( x_2 \) and \( y_2 \), respectively. As \( (a, b) \in I_M \), we see from the above stated properties of \( \mathcal{M} \) that \( b \in \Sigma_2^{(i)} \). Hence, in the former case we obtain the mode \( = 1 \) computation

\[
w_{i+1} = xbay \vdash_{\mathcal{M}} x_1bay_1 \vdash_{M_1} x_1by_1 \vdash_{M_2} x_2by_2 \vdash_{M_j} x_2y_2 \vdash_{\mathcal{M}} \text{Accept},
\]

while in the latter case we obtain the mode \( = 1 \) computation

\[
w_{i+1} = xbay \vdash_{\mathcal{M}} x_1bay_1 \vdash_{M_j} x_1ay_1 \vdash_{M_2} x_2ay_2 \vdash_{M_j} x_2y_2 \vdash_{\mathcal{M}} \text{Accept}.
\]

Thus, we see that \( w = w_n \) is accepted by a mode \( = 1 \) computation of \( \mathcal{M} \), which completes the proof of Claim 1. □

**Claim 2.** \( L \subseteq \bigcup_{u \in R} [u]_{D_M} \).

**Proof.** Let \( w \in L \), and let

\[
w = w_n \vdash_{M_n} w_{n-1} \vdash_{M_{n-1}} \cdots \vdash_{M_2} w_1 \vdash_{M_1} w_0 = \varepsilon \vdash_{M_{n+1}} \text{Accept}
\]

be an accepting mode \( = 1 \) computation of \( \mathcal{M} \) on input \( w \). We claim that, for each \( j = 1, \ldots, n \), there exists a word \( u_j \in \Sigma^* \) such that \( u_j \equiv_{D_M} w_j \) and \( p_+ \in \delta(q_i, u_j) \), that is, the finite-state acceptor \( B \) accepts the word \( u_j \) when starting from state \( q_i \).
We prove this claim by induction on \( j \). For \( j = 1 \) we have \( w_j = a_1 \), where \( \Sigma_{1j} = \{a_1\} \), and \( + \in \sigma_1 \). From the definition of \( B \) we conclude that \( p_+ \in \delta(q_1, a_1) \), that is, we can simply take \( u_1 = a_1 = w_1 \). Now assume that, for some \( j \geq 1 \), \( u_j \equiv_D M_j w_j \) and \( p_+ \in \delta(q_j, u_j) \) hold. The above computation of \( M \) contains the cycle \( w_j \mapsto M_{j+1} w_j \), that is, \( w_{j+1} = xa_{j+1}y \) and \( w_j = xy \) for some words \( x, y \in \Sigma^* \) and the letter \( a_{j+1} \) satisfying \( \Sigma_{2j+1} = \{a_{j+1}\} \), and \( i_j \in \sigma_{j+1} \). Also we see that \( (x, a_{j+1}) \in I_M \).

Again from the definition of \( B \) it follows that \( q_{j+1} \in \delta(q_{j+1}, a_{j+1}) \). Now let \( u_{j+1} \) be the word \( u_{j+1} = a_{j+1} u_j \). Then

\[
u_{j+1} = a_{j+1} u_j \equiv_D M_j a_{j+1} w_j = a_{j+1} x y \equiv_D M_j xa_{j+1} y = w_{j+1},\]

and \( \delta(q_{j+1}, u_{j+1}) = \delta(q_{j+1}, a_{j+1} u_j) = \delta(q_{j+1}, a_{j+1}, u_j) \supseteq \delta(q_j, u_j) \supseteq p_+ \). Finally, for \( j = n \) we obtain a word \( u \) such that \( u \equiv_D M w \) and \( p_+ \in \delta(p_0, u) \), which means that \( u \in R \). Thus, it follows that \( L \subseteq \bigcup_{u \in R} [u]_D \) holds. \( \square \)

Now Claims 1 and 2 together show that \( L = L_{=1}(M) = \bigcup_{u \in R} [u]_D \), which completes the proof of Theorem 5.7. \( \square \)

Observe that the system \( M \) constructed in the proof of Theorem 5.5 is in normal form, that it satisfies property (\( \ast \)), and that the associated relation \( I_M \) coincides with the relation \( I_D \), and hence, it is symmetric. Thus, Theorems 5.5 and 5.7 together yield the following characterization.

**Corollary 5.8.** A language \( L \subseteq \Sigma^* \) is a rational trace language if and only if there exists a \( \text{stl-det-local-CD-R}(1) \)-system \( M \) in normal form satisfying condition (\( \ast \)) such that the relation \( I_M \) is symmetric and \( L = L_{=1}(M) \).

In the proof of Theorem 5.5 we effectively constructed a \( \text{stl-det-local-CD-R}(1) \)-system for the rational trace language \( \varphi_D^{-1}(\varphi_D(R)) \) from a finite-state acceptor for the regular language \( R \). Hence, if \( S_1, S_2 \subseteq M(D) \) are rational subsets of the trace monoid \( M(D) \), then we can construct finite-state acceptors \( B_1 \) and \( B_2 \) from \( \text{stl-det-local-CD-R}(1) \)-systems \( M_1 \) for \( L_1 = \varphi_D^{-1}(S_1) \) and \( M_2 \) for \( L_2 = \varphi_D^{-1}(S_2) \) such that \( S_1 = \varphi_D(R_1) \) and \( S_2 = \varphi_D(R_2) \), where \( R_i = L(B_i) \), \( i = 1, 2 \). It is easily seen that \( S_1 \cup S_2 = \varphi_D(R_1 \cup R_2) \), \( S_1 \cdot S_2 = \varphi_D(R_1 \cdot R_2) \), and \( S_1^2 = \varphi_D(R_1^2) \). From \( R_1 \) and \( R_2 \) we can construct finite-state acceptors for the languages \( R_1 \cup R_2 \), \( R_1 \cdot R_2 \), and \( R_1^2 \). Thus, Theorem 5.7 shows that we can construct \( \text{stl-det-local-CD-R}(1) \)-systems for the languages \( \varphi_D^{-1}(S_1 \cup S_2) \), \( \varphi_D^{-1}(S_1 \cdot S_2) \), and \( \varphi_D^{-1}(S_1^2) \). Hence, the \( \text{stl-det-local-CD-R}(1) \)-systems of Corollary 5.8 form an effective calculus for rational trace languages. However, a \( \text{stl-det-local-CD-R}(1) \)-system may accept a rational trace language, even if it does not satisfy all the additional restrictions above. Hence, the following problem remains.

**Open Problem 4.** Is there a syntactic characterization for those \( \text{stl-det-local-CD-R}(1) \)-systems that accept rational trace languages by mode \( 1 \) computations?

### 6. Closure properties

Corollary 4.12 shows that the language class \( L_{=1}(\text{stl-det-local-CD-R}(1)) \) is not closed under intersection with regular languages. Here we derive further non-closure properties, but also a number of closure properties for this class.

The **commutative closure** \( \text{com}(L) \) of a language \( L \subseteq \Sigma^* \) is the set of all words that are letter-equivalent to a word from \( L \), that is,

\[
\text{com}(L) = \psi^{-1}(\psi(L)) = \{w \in \Sigma^* \mid \exists u \in L : \psi(w) = \psi(u)\},
\]

where \( \psi : \Sigma^* \rightarrow \mathbb{N}^{[\Sigma]} \) denotes the Parikh mapping (see Section 4). If \( L \) is accepted by a \( \text{stl-det-local-CD-R}(1) \)-system \( M \), then from \( M \) we can construct a finite-state acceptor \( B \) for a regular sublanguage \( E \) of \( L \) that is letter-equivalent to \( L \) (Theorem 4.9). Obviously, the commutative closure \( \text{com}(L) \) of \( L \) coincides with the commutative closure \( \text{com}(E) \) of \( E \). For the dependency relation \( D = \{(a, a) \mid a \in \Sigma\} \), the trace monoid \( M(D) \) presented by \( (\Sigma, D) \) is the free commutative monoid generated by \( \Sigma \). Thus, \( \text{com}(E) = \bigcup_{w \in E} [w]_D \) is simply the rational trace language \( \varphi_D^{-1}(\varphi_D(E)) \). Hence, it follows from Theorem 5.5 that this language is accepted by a \( \text{stl-det-local-CD-R}(1) \)-system \( M' \). In fact, the system \( M' \) can effectively be constructed from the finite-state acceptor \( B \), and therewith from the given \( \text{stl-det-local-CD-R}(1) \)-system \( M \). This yields the following effective closure property.

**Corollary 6.1.** The language class \( L_{=1}(\text{stl-det-local-CD-R}(1)) \) is effectively closed under the operation of taking the commutative closure.

A language \( L \subseteq \Sigma^* \) is called **commutative** if \( \text{com}(L) = L \) holds, that is, if it contains all permutations of all its elements. As each semi-linear language is letter-equivalent to some regular language, it follows that each commutative semi-linear language is the commutative closure of some regular language, and therewith it is a rational trace language. Thus, Theorem 5.5 implies the following result.

**Corollary 6.2.** All commutative semi-linear languages are contained in the language class \( L_{=1}(\text{stl-det-local-CD-R}(1)) \).
Next we consider the closure under Boolean operations.

**Proposition 6.3.**

(a) The language class \( L_{\text{stl-det-local-CD-R}}(1) \) is closed under union.
(b) The language class \( L_{\text{stl-det-local-CD-R}}(1) \) is neither closed under intersection nor under complementation.

**Proof.** (a) Let \( M = ((M_i, \sigma_i)_{i \in I}, I_0) \) and \( M' = ((M'_i, \sigma'_i)_{i \in I'}, I'_0) \) be stl-det-local-CD-R(1)-systems with disjoint sets of indices \( I \) and \( I' \). We define a new stl-det-local-CD-R(1)-system \( \tilde{M} = ((\tilde{M}_i, \tilde{\sigma}_i)_{i \in I}, I_0) \) by taking \( I = I \cup I' \), \( I_0 = I_0 \cup I'_0 \),

\[
\tilde{M}_i = \begin{cases} M_i, & \text{for } i \in I \\ M'_i, & \text{for } i \in I' \end{cases}, \quad \text{and} \quad \tilde{\sigma}_i = \begin{cases} \sigma_i, & \text{for } i \in I \\ \sigma'_i, & \text{for } i \in I' \end{cases}.
\]

Then \( \tilde{M} \) is the disjoint union of the two given systems, and it follows immediately that \( L = L(\tilde{M}) = L_1(M) \cup L_1(M') \). This proves that the class \( L_{\text{stl-det-local-CD-R}}(1) \) is closed under union.

(b) From Proposition 4.4 and Corollary 4.12 we see that this language class is not closed under intersection. Now closure under union and non-closure under intersection imply that this class is not closed under complementation, either. □

We now turn to the product operation. We will show that the language class \( L_{\text{stl-det-local-CD-R}}(1) \) is closed under product, that is, if \( L_1 \) and \( L_2 \) are accepted by stl-det-local-CD-R(1)-systems, then so is the language \( L_1 \cdot L_2 = \{ uv \mid u \in L_1, v \in L_2 \} \).

Remark 4.8 we can even assume that it only has a single accepting component \( M_+ \), and that this component only accepts the empty word. Thus, \( M_1 \) reduces a given input word \( w \in L_1 \) first to the empty word by performing \(|w|\) many cycles, and then it accepts by activating \( M_+ \). Now it would appear that we obtain a stl-det-local-CD-R(1)-system \( M \) for the language \( L_1 \cdot L_2 \) by simply replacing the component \( M_+ \) of \( M_1 \) by the initial components of the system \( M_2 \) for the language \( L_2 \). However, the situation is not that easy as shown by the following example.

**Example 6.4.** Consider the following language \( L_1 \) on \( \Sigma = \{ a, b, c, d \} \), where \( D_1^* \) denotes the Dyck language on \( \{ a, b \} \), \( \hat{D}_1^* \) denotes the Dyck language on \( \{ c, d \} \), and \( \text{sh} \) denotes the shuffle operation:

\[
L_1 = \{ w \in \Sigma^* \mid w \in \text{sh}(D_1^*, \hat{D}_1^*) \text{ such that } \forall x, y, z: w = xyz \text{ and } |x|_c = |xy|_d \implies |x|_a \geq |xy|_b \},
\]

and let \( M_1 = ((M_i, \sigma_i)_{i \in I}, I_0) \) be the stl-det-local-CD-R(1)-system that is specified by \( I = \{ 1, 2, 3, 4 \}, I_0 = \{ 1, 3 \}, \sigma_1 = \{ 2 \}, \sigma_2 = \{ 1, 3, + \}, \sigma_3 = \{ 4 \}, \sigma_4 = \{ 1, 3, + \}, + \), and the R-automata \( M_1, \ldots, M_4, M_+ \) are defined through the following transition functions:

\[
\begin{align*}
M_1: \quad (1) & \quad \delta_1(\epsilon) = \text{MVR}, \\
& \quad (2) \quad \delta_1(\alpha) = \epsilon, \\
M_2: \quad (3) & \quad \delta_2(\epsilon) = \text{MVR}, \\
& \quad (4) \quad \delta_2(x) = \text{MVR} \quad \text{for all } x \in \{ a, b, c, d \}, \\
& \quad (5) \quad \delta_2(b) = \epsilon, \\
M_3: \quad (6) & \quad \delta_3(\epsilon) = \text{MVR}, \\
& \quad (7) \quad \delta_3(c) = \epsilon, \\
M_4: \quad (8) & \quad \delta_4(\epsilon) = \text{MVR}, \\
& \quad (9) \quad \delta_4(x) = \text{MVR} \quad \text{for all } x \in \{ a, c \}, \\
& \quad (10) \quad \delta_4(d) = \epsilon, \\
M_+: \quad (11) & \quad \delta_+(\epsilon) = \text{MVR}, \\
& \quad (12) \quad \delta_+(\$) = \text{Accept}.
\end{align*}
\]

**Claim 1.** \( L_1 = L_{\text{det-local-CD-R}}(M_1) \).

**Proof.** Let \( w \in L_1 \), \( w \neq \epsilon \), be given as input. As \( L_1 \) is contained in the shuffle product of \( D_1^* \) and \( \hat{D}_1^* \), each element of \( L \) starts with an occurrence of a letter \( a \) or \( c \). Accordingly, if \( w = aubv \), where \(|u|_b = 0 \), then the \( M_1 \)-computation starts with component automaton \( M_1 \), that is, it starts by executing the cycles \( w = aubv \vdash M_1 \vdash uv \vdash M_2 \vdash uv \). On the other hand, if \( w = cu'dv' \), where \(|u'|_d = 0 \), then the \( M_1 \)-computation starts with the component automaton \( M_3 \), which executes the cycle...
\[ w = cu'dv' \vdash_{M_k} u'dv'. \] after which automaton \( M_k \) becomes active. Now \( M_k \) can erase the leftmost occurrence of the letter \( d \) only if \( |u'|_b = 0 \), which, however, is satisfied if \( w \in L \). It will then execute the cycle \( u'dv' \vdash_{M_k} u'dv' \).

Now by repeatedly cycling through these two cycles of length two, the word \( w \) will be reduced to the empty word, if it is an element of the language \( L \), and then automaton \( M_+ \) is called, which accepts.

Conversely, we see from the definition of \( M_1 \) that all accepting computations proceed as described above, which implies that only words from the language \( L_1 \) are accepted. It follows that \( L_1(M_1) = L_1 \) holds. \( \square \)

Now let \( L_2 \) be the language \( L_{abc} = \{ w \in \{ a, b, c \}^* \mid |w|_a = |w|_b = |w|_c \} \) from Proposition 4.2. Then \( L_2 \) is accepted by the stl-det-local-CD-R(1)-system

\[ M_2 = (\langle M_1, \sigma_1 \rangle, (M_2, \sigma_2), (M_3, \sigma_2), (M_4, \sigma_2), \{ a, + \}) \]

given in the proof of Proposition 4.2. If we construct a stl-det-local-CD-R(1)-system \( M \) by combining the systems \( M_1 \) and \( M_2 \), replacing each occurrence of \( M_+ \) in the successor sets of \( M_1 \) by \( M_0 \), then the resulting system will certainly accept all words from the product \( L_1 \cdot L_2 \). However, it will also execute the following accepting computation:

\[ acdcbba \vdash_{M_1} cdcdba \vdash_{M_2} dcdba \vdash_{M_3} cdba \vdash_{M_4} cb \vdash_{M_3} c \vdash_{M_1} \in \vdash_{M}^* \text{ Accept.} \]

However, the word \( acdcbba \) does not belong to the product \( L_1 \cdot L_2 \), a contradiction.

The problem in the above example stems from the fact that, in computations of the system \( M_1 \), the component automaton \( M_2 \) reads across occurrences of the symbol \( d \) when looking for the leftmost occurrence of the symbol \( b \). Accordingly, it may delete an occurrence of \( b \) that does not belong to the first factor. Thus, we need to modify the system \( M_1 \) into an equivalent system \( M'_1 \) for which this does not happen. This is the contents of the following technical result.

**Theorem 6.5.** From a stl-det-local-CD-R(1)-system \( M \) one can construct a stl-det-local-CD-R(1)-system \( M' \) such that \( L_1(M') = L_1(M) \cdot \Sigma^* \). For all \( u \in L_1(M) \) and all \( v \in \Sigma^* \), each accepting computation of \( M' \) on input \( w = uv \) will completely delete the word \( u \in L_1(M) \) before it deletes a letter from \( v \).

**Proof.** Let \( M = (\langle M_1, \sigma_1 \rangle) \in \mathbb{F}_1 \) be a stl-det-local-CD-R(1)-system that accepts a language \( L \subseteq \Sigma^* \). We assume that \( M \) is in normal form, and that \( M_+ \) is the only accepting component automaton of \( M \), and that this component only accepts the empty word, that is, \( \delta_+ \) is defined as \( \delta_+ = \{ (c, \text{ MVR}), (\emptyset, \text{ Accept}) \} \) (see Remark 4.8). Further, for each \( i \in I \), we use \( \Sigma_i^{(1)} \) and \( \Sigma_i^{(2)} \) to denote the subalphabets of \( \Sigma \) that correspond to automaton \( M_i \) according to Definition 2.3. Finally we assume that the alphabet \( \Sigma \) is ordered. For simplicity we write \( \Sigma = \{ a_1, \ldots, a_n \} \), and call \( a_i \) the \( i \)-th letter of \( \Sigma \), \( 1 \leq i \leq n \).

The stl-det-local-CD-R(1)-system \( M \) will now be modified into a system \( M' \) that meets the requirements stated above. This system will consist of a (large) number of subsystems each of which is a (slightly) revised version of \( M \). These subsystems will be indexed by the set of \( n \)-tuples

\[ \text{IND} = \{ (i_1, \ldots, i_n) \mid i_1, \ldots, i_n \in \{ 2, 1, 0, d \} \}. \]

Below we will describe the necessary modifications for the various subsystems, but as a first general rule we require that all those component automata of \( M \) are excluded from the subsystem \( M^{(i_1, \ldots, i_n)} \) that attempt to erase a letter \( a_k \) for which \( i_k = 0 \) or \( i_k = d \) holds. With a word \( w \in \Sigma^* \) we associate an index \( \text{IND}(w) = (i_1, \ldots, i_n) \) by taking

\[ i_j = \begin{cases} 2, & \text{if } |w|_{a_j} \geq 2 \\ 1, & \text{if } |w|_{a_j} = 1 \\ 0, & \text{if } |w|_{a_j} = 0 \end{cases} \]

for all \( j = 1, \ldots, n \).

On input \( w \in \Sigma^* \), \( M' \) guesses a tuple \( \text{IND}'(w) = (i_1, \ldots, i_n) \in \{ 2, 1, 0 \}^n \), and then one of the initial component automata \( M_k^{\text{IND}'(w)} \) of subsystem \( M_k^{\text{IND}'(w)} \) is activated. It attempts to erase the leftmost occurrence of a letter \( a_k \) for some \( s \) such that \( i_s \neq 0 \). If \( \text{IND}'(w) \neq \text{IND}(w) \), then at some point in the resulting computation this will be realized, causing \( M' \) to halt and reject (see the detailed description of \( M' \) below).

If \( i_1 = 2 \), then \( M_k^{\text{IND}'(w)} \) transforms the word \( w = w_1 a_1 w_2 \) into the word \( w_1 w_2 \), provided that \( w_1 \in \Sigma_1^{(k)} \). Now \( \text{IND}(w_1 w_2) \) either coincides with \( \text{IND}(w) \) or it is obtained from \( \text{IND}(w) \) by replacing \( i_k \) by the value \( i_k' = 1 \). Accordingly, if \( j \in \delta_k \), then \( j^{(i_1, \ldots, i_{k-1}, 1, i_{k+1}, \ldots, i_n)} \) in \( \sigma_k^{(j)} \) holds.

If \( i_1 = 1 \), then \( M_k \) would transform the word \( w = w_1 a_1 w_2 \) into the word \( w_1 w_2 \), provided that \( w_1 \in \Sigma_1^{(k)} \). Here \( |w_1 w_2|_{a_0} = 0 \), if \( \text{IND}'(w) = \text{IND}(w) \), that is, within this cycle \( M_k \) would delete the last occurrence of the letter \( a_0 \). This, however, may cause problems (see the discussion above), and so we must be very careful when simulating this step. If \( w_1 w_2 \) contains occurrences of letters that do not belong to the subalphabet \( \Sigma_1^{(k)} \), then either the above cycle will not be
completed successfully, if \(w_1\) contains such a letter, or these letters are all contained in \(w_2\), implying that \(a_\varepsilon\) is not the rightmost letter of \(w\). Thus, if the subalphabet

\[
\text{Alph}(\text{IND'}(w)) = \{a_j \in \Sigma \mid j \in \{1, \ldots, n\} \text{ such that } i_j \in \{2, 1\}\}
\]

is not contained in \(\Sigma_1^{(i)} \cup \{a_\varepsilon\}\), then \(M_k^{\text{IND}(w)}\) just simulates the above cycle of \(M_k\), and \(j^{(1, \ldots, 0, \ldots, i_n)} \in \sigma_k^{(1, \ldots, n)}\) for all \(j \in \sigma_k\).

If, however, \(\text{Alph}(\text{IND}(w)) \subseteq \Sigma_1^{(i)} \cup \{a_\varepsilon\}\), then instead of \(M_k^{\text{IND}(w)}\) a component automaton \(M_k^{(i_1, \ldots, i_n)}\) is activated for some \(j \in \sigma_k\). Here the indicator \(d\) in position \(s\) means that the current word \(w\) contains a single occurrence of the letter \(a_i\), but that in the corresponding computation of the system \(M\), this letter has already been erased. Observe that the above property only depends on the value \(\text{IND}(w)\) guessed and on the component automaton \(M_k^{\text{IND}(w)}\), and hence, the set of initial components of \(M'\) can be chosen accordingly.

The component \(M_k^{(i_1, \ldots, i_n)}\) is obtained from \(M_j\) by applying the following modifications, where we assume that

\[
\Sigma_2^{(i)} = \{a_i\}.
\]

We see from the general rule above that \(i_i \in \{2, 1\}\) holds. Let \(D(i_1, \ldots, i_n) = \{j \in \{1, \ldots, n\} \mid i_j = d\}\). Then \(M_k^{(i_1, \ldots, i_n)}\) consists of \(|D(i_1, \ldots, i_n)| + 1\) many subcomponents. For each \(l \in D(i_1, \ldots, i_n)\), there is a component \(M_k^{(i_1, \ldots, i_n)}\) that deletes an occurrence of the letter \(a_i\) if it is the first letter of the tape inscription, that is, the corresponding transition function \(\delta\) is defined by \(\delta(p, e) = \text{MVR}\) and \(\delta(a_i) = e\). Further, the only successor system of \(M_k^{(i_1, \ldots, i_n)}\) is the system \(M_k^{(i_1, \ldots, 0, \ldots, i_n)}\), where \(i_i = d\) is replaced by \(i_i' = 0\), indicating that the last occurrence of the letter \(a_i\) has now been erased. Finally there is a subcomponent \(M_k^{(i_1, \ldots, 0, \ldots, i_n)}\) that simulates the actual behaviour of \(M_j\). Here we have to distinguish two cases.

If \(i_i = 2\), then \(M_k^{(i_1, 0, \ldots, d, \ldots, i_n)}\) simply deletes the first occurrence of the letter \(a_i\), and in doing so it may move across all letters from

\[
(\Sigma_1^{(i)} \cap \text{Alph}(i_1, \ldots, i_n)) \cup \{a_\mu \mid \mu \in D(i_1, \ldots, i_n)\},
\]

that is, it may move across all letters in \(a_i \in \Sigma_1^{(i)}\) for which the indicator \(i_i\) is 1 or 2, and across all letters \(a_\mu\) for which \(i_i = d\). Further, \(p^{(i_1, \ldots, 2, \ldots, d, \ldots, i_n)}\), \(p^{(i_1, \ldots, 1, \ldots, d, \ldots, i_n)} \in \sigma_k^{(1)}\) for all \(p \in \sigma_j\), where the index 2 or 1 is in position \(r\).

If \(i_i = 1\), then the behaviour is similar, if \(\text{Alph}(i_1, \ldots, i_n)\) is not contained in \(\Sigma_1^{(i)} \cup \{a_i\}\). In that case \(p^{(i_1, \ldots, 0, \ldots, d, \ldots, i_n)} \in \sigma_k^{(1)}\) for all \(p \in \sigma_j\), where the index 0 is in position \(r\).

Finally, if \(i_i = 1\) and \(\text{Alph}(i_1, \ldots, i_n) \subseteq \Sigma_1^{(i)} \cup \{a_i\}\), then instead of \(M_k^{(i_1, \ldots, d, \ldots, i_n)}\) a component automaton \(M_k^{(i_1, \ldots, d, \ldots, i_n)}\) is activated for some \(p \in \sigma_j\). Here the additional indicator \(d\) in position \(r\) means that the current word contains a single occurrence of the letter \(a_i\), but that in the corresponding computation of the system \(M\), this letter has already been erased. Observe that the above property only depends on \((i_1, \ldots, d, \ldots, i_n)\) and on the component automaton \(M_j\), and hence, the set of successor components of \(M_k^{\text{IND}(w)}\) can be chosen accordingly.

Finally the accepting component \(M_k^{+}\) is only called from a component \(M_q^{(i_1, \ldots, i_n)}\) for which all but one of the indicators \(i_1, \ldots, i_n\) are 0, the only non-zero indicator \(i_v\) is 1 or \(d\), and \(M_k^{(i_1, \ldots, i_n)}\) deletes an occurrence of the letter \(a_i\). The component \(M_k^{+}\) simply accepts all words from \(\Sigma^*\).

The above modifications are now applied to all component automata \(M_k^{(i_1, \ldots, i_n)}\), where \((i_1, \ldots, i_n) \in \text{IND}^*\) and \(j \in I\).

This completes the description of the system \(M'\). Concerning the behaviour of this system we observe the following:

1. For each word \(w \in L_1(M)\) and each \(v \in \Sigma^*\), \(M'\) has an accepting mode \(= 1\) computation on input \(uv\), that is, \(L_1(M) \cdot \Sigma^* \subseteq L_1(M')\).
2. Each accepting computation of \(M'\) consists of two phases. In the first phase a prefix \(u\) of the given input \(w = uv\) is read simulating \(M\), and then the prefix \(v\) is simply accepted by \(M'\).
3. If during a computation of \(M'\) a component automaton \(M_k^{(i_1, \ldots, i_n)}\) is activated such that, for some \(j\), \(i_j \in \{2, 1, d\}\), but no symbol \(a_\mu\) is on the tape, then \(i_j\) will never be set to 0, and accordingly, this computation fails.
4. If during the first phase of a computation of \(M'\) on input \(w = uv\) a component automaton \(M_k^{(i_1, \ldots, i_n)}\) is activated such that, for some \(j, i_j = 0\), but there are still occurrences of the symbol \(a_\mu\) on the tape, then these occurrences will not be erased within the first phase of this computation, and accordingly, these letters must occur within the suffix \(v\) thus, during a computation of \(M'\), if \(w = uv\) is the given input, then in an accepting computation the correct value for \(\text{IND}(u)\) must be guessed.
5. Each time the last occurrence of a letter \(a_i\) is erased from \(u\), it is ensured that this occurrence is not the last letter of \(u\) or that it is the first letter currently on the tape. Thus, the very last letter of \(u\) can only be erased when it has become the very first letter on the tape, that is, when the rest of the word \(u\) has already been erased completely.
From (2) to (4) it follows that $\mathcal{M}'$ can only accept words from the language $L_{=1}(\mathcal{M}) \cdot \Sigma^*$, which together with (1) implies that $L_{=1}(\mathcal{M}) \cdot \Sigma^* = L_{=1}(\mathcal{M}')$ holds. From (5) it follows that on input a word of the form $w = uv$, where $u \in L$ and $v \in \Sigma^*$, $\mathcal{M}'$ has a computation that erases the prefix $u$ completely and then calls the final component automaton $M'_{\ast}$ without scanning any prefix of $v$. Conversely, if $\mathcal{M}'$ has a computation that, starting with input $uv$, $u \in \Sigma^*$, erases the prefix $u$ completely and then calls the final component automaton $M'_{\ast}$, then $u \in L$, and during this computation $\mathcal{M}'$ does not scan any prefix of $v$. This completes the proof of Theorem 6.5. \(\square\)

Now let $\mathcal{M}_1$ be a stl-det-local-CD-R(1)-system accepting a language $L_1$, let $\mathcal{M}_2$ be a stl-det-local-CD-R(1)-system accepting a language $L_2$, and let $\mathcal{M}_1'$ be the system obtained from $\mathcal{M}_1$ according to Theorem 6.5. If we now replace every occurrence of $M'_{\ast}$ in the set of initial components and in the sets of successor components of $\mathcal{M}_1'$ by the initial components of the system $\mathcal{M}_2$, then we obtain a stl-det-local-CD-R(1)-system $\mathcal{M}$ for the language $L_1 \cdot L_2$. Hence, we have the following closure property.

**Corollary 6.6.** The language class $\mathcal{L}_{=1}(\text{stl-det-local-CD-R}(1))$ is closed under product.

As a consequence of the construction above the following results are now immediate.

**Corollary 6.7.** The language class $\mathcal{L}_{=1}(\text{stl-det-local-CD-R}(1))$ is closed under Kleene-star and Kleene-plus.

For showing that the class $\mathcal{L}_{=1}(\text{stl-det-local-CD-R}(1))$ is not closed under morphisms, we need a variant of the language $L_{ab} = \{ w \in \{a, b\}^* \mid |w|_a = |w|_b \}$.

In analogy to Proposition 4.2 it can be shown that $L_{ab}$ is accepted by a CD-system of stateless deterministic $R$-automata working in mode $= 1$. Now consider the morphism $\varphi : \{a, b\}^* \to \{a, b\}^*$ that is induced by $a \mapsto ab$ and $b \mapsto b$, and let $L'_{ab}$ denote the language $\varphi(L_{ab})$. It is easily seen that $w \in L'_{ab}$ if and only if $|w|_b = 2 \cdot |w|_a$, and each occurrence of a letter $a$ in $w$ is immediately followed by an occurrence of the letter $b$.

**Lemma 6.8.** The language $L'_{ab}$ is not accepted by any stl-det-local-CD-R(1)-system working in mode $= 1$.

**Proof.** Assume that $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ is a stl-det-local-CD-R(1)-system in normal form satisfying $L_{=1}(\mathcal{M}) = L'_{ab}$. If $i_0 \in I$ such that $M_{i_0}$ executes accept instructions, then we see from Proposition 2.4 that $S(M_{i_0}) = \{e\}$ must hold. Thus, any given non-empty word $w \in L'_{ab}$ must first be reduced to the empty word by executing $|w|$ many cycles, and then a component $M_{i_0}$ is called which accepts.

Consider the words $w = b^n(ab)^n \in L'_{ab}$ and $w' = b^n ba(ab)^{n-1} \notin L'_{ab}$, where $n > 0$ is sufficiently large. The system $\mathcal{M}$ has an accepting mode $= 1$ computation for input $w$. In this computation, if an occurrence of the letter $a$ before the prefix $b^n$ has been deleted completely, that is, if this accepting computation can be written as

\[ w = b^n(ab)^n \gamma_{M_1} \cdots \gamma_{M_j} b^{n-j}(ab)^n \gamma_{M_{ij}} b^{n-j} ba(ab)^{n-1} \vdash_{\mathcal{M}} \text{Accept}, \]

then $\mathcal{M}$ will also perform the following computation:

\[ w' = b^n ba(ab)^{n-1} \gamma_{M_1} \cdots \gamma_{M_j} b^{n-j} ba(ab)^{n-1} \gamma_{M_{ij}} b^{n-j} ba(ab)^{n-1} \vdash_{\mathcal{M}} \text{Accept}. \]

Thus, $\mathcal{M}$ will also accept the word $w'$, a contradiction.

Hence, in an accepting mode $= 1$ computation of $\mathcal{M}$ on input $w$, no occurrence of the letter $a$ before the prefix $b^n$ has been deleted completely. If $n$ is sufficiently large, then this accepting computation can be written as

\[ w = b^n(ab)^n \gamma_{M_k} b^{n-k}(ab)^n \gamma_{M_l} b^{n-k-l-1}(ab)^n \gamma_{M_m} b^{n-k-l-2}(ab)^n \gamma_{M_1} b^{n-k-l-2}(ab)^n \vdash_{\mathcal{M}} \text{Accept}, \]

for some index $i \in I$ and some numbers $k, l \leq |I|$. But then $\mathcal{M}$ would also perform the following accepting computation:

\[ b^{n+i+1}(ab)^n \gamma_{M_k} b^{n-i+1}(ab)^n \gamma_{M_l} b^{n-k+i+1}(ab)^n \gamma_{M_m} b^{n-k+l}(ab)^n \gamma_{M_1} b^{n-k-1}(ab)^n \gamma_{M_1} b^{n-k-1}(ab)^n \vdash_{\mathcal{M}} \text{Accept}. \]

As $b^{n+i+1}(ab)^n \notin L'_{ab}$, this is again a contradiction. It follows that $L'_{ab}$ is not accepted by any stl-det-local-CD-R(1)-system working in mode $= 1$. \(\square\)

As $L_{ab} \in \mathcal{L}_{=1}(\text{stl-det-local-CD-R}(1))$, Lemma 6.8 has the following consequence.

**Corollary 6.9.** The language class $\mathcal{L}_{=1}(\text{stl-det-local-CD-R}(1))$ is not closed under $\varepsilon$-free morphisms.
Concerning inverse morphisms we have the following preliminary result.

**Proposition 6.10.** The language class $L_{\omega 1}(stl-det-local-CD-R(1))$ is closed under inverse projections.

**Proof.** Let $\mathcal{M} = (\langle M_i, \sigma_i \rangle)_{i \in I}, I_0)$ be a stl-det-local-CD-R(1)-system in strong normal form (see Remark 4.8) on $\Sigma$ accepting a language $L \subseteq \Sigma^*$ in mode $= 1$, let $\Gamma$ be an alphabet that is disjoint from $\Sigma$, and let $\Pr : (\Sigma \cup \Gamma)^* \to \Sigma^*$ be the projection that is induced by $a \mapsto a$ for all $a \in \Sigma$ and $b \mapsto \varepsilon$ for all $b \in \Gamma$. By $L'$ we denote the language

$$L' = \Pr^{-1}(L) = \{ w \in (\Sigma \cup \Gamma)^* \mid \Pr(w) \in L \}.$$  

From $\mathcal{M}$ we construct a stl-det-local-CD-R(1)-system $\mathcal{M}'$ for $L'$ as follows:

$$\mathcal{M}' = \langle \langle D_1, \sigma_{D_1} \rangle, \langle D_2, \sigma_{D_2} \rangle, \langle M_i, \sigma_i \rangle_{i \in I}, I_0 \cup \{ D_1 \} \rangle.$$  

The R-automata $D_1$ and $D_2$ are defined as follows:

**$D_1$:**

1. $\delta_{D_1}(\varepsilon) = \text{MVR},$
2. $\delta_{D_1}(a) = \text{MVR}$ for all $a \in \Sigma$,
3. $\delta_{D_1}(b) = \varepsilon$ for all $b \in \Gamma$,

**$D_2$:**

4. $\delta_{D_2}(a) = \text{MVR},$
5. $\delta_{D_2}(a) = \text{MVR}$ for all $a \in \Sigma$,
6. $\delta_{D_2}(b) = \varepsilon$ for all $b \in \Gamma$,

and $\sigma_{D_1} = \{ D_2 \} \cup I_0$ and $\sigma_{D_2} = \{ D_1 \} \cup I_0$.

Given an input $w \in (\Sigma \cup \Gamma)^*$, $\mathcal{M}'$ uses the component automata $D_1$ and $D_2$ to delete all occurrences of symbols from $\Gamma$, and then it checks whether the word obtained is accepted by $\mathcal{M}$. It follows that $L_{\omega 1}(\mathcal{M}') = L'$. \(\square\)

However, the following general closure property is still open.

**Open Problem 5.** In the language class $L_{\omega 1}(stl-det-local-CD-R(1))$ closed under inverse morphisms?

The application of an inverse projection $\Pr^{-1}$ to a language $L \subseteq \Sigma^*$ results in the shuffle of $L$ with the free monoid $\Gamma^*$, where $\Gamma$ is the set of letters mapped by $\Pr$ to $\varepsilon$. In fact, it can be shown that the language class $L_{\omega 1}(stl-det-local-CD-R(1))$ is closed under disjoint shuffle, that is, if $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Gamma^*$ are languages in $L_{\omega 1}(stl-det-local-CD-R(1))$, where $\Sigma \cap \Gamma = \emptyset$, then the shuffle of $L_1$ and $L_2$ is also in this language class.

**Open Problem 6.** Derive further closure and non-closure results for the language class $L_{\omega 1}(stl-det-local-CD-R(1))$. In particular, is this class closed under reversal?

Let $\Sigma$ be a finite alphabet, and let $\Sigma = \{ a \mid a \in \Sigma \}$ be a copy of $\Sigma$ such that $\Sigma \cap \Sigma = \emptyset$. By $\tilde{\Sigma} : \Sigma^* \to \Sigma^*$ we denote the morphism that replaces each letter $a \in \Sigma$ by its copy $\tilde{a}$. Then the language $L_{\Sigma} := \{ \text{sh}(w, \tilde{w}) \mid w \in \Sigma^* \}$ is called the twin shuffle language over $\Sigma$. These twin shuffle languages are quite expressive as shown by the following classical result.

**Proposition 6.11.** (See [22].) For each recursively enumerable language $L \subseteq \Sigma_1^*$, there exist an alphabet $\Sigma$ containing $\Sigma_T$ and a regular language $R \subseteq (\Sigma \cup \tilde{\Sigma})^*$ such that $L = \Pr_{\Sigma_T}(L_{\Sigma} \cap R)$.

Observe that the twin shuffle language $L_{\Sigma}$ is actually a rational trace language. Indeed, consider the dependency relation $D_{\Sigma}$ on $\Sigma \cup \tilde{\Sigma}$ that is defined by $D_{\Sigma} := \{ (a, b, \tilde{a}, \tilde{b}) \mid a, b \in \Sigma \}$, and let $R_{\Sigma} := \{ a\tilde{a} \mid a \in \Sigma \}$. Then $R_{\Sigma}$ is a regular language over $\Sigma \cup \tilde{\Sigma}$, and $[R_{\Sigma}]_{D_{\Sigma}} = L_{\Sigma}$. Hence, there exists a stl-det-local-CD-R(1)-system $\mathcal{M}_{\Sigma}$ satisfying $L_{\omega 1}(\mathcal{M}_{\Sigma}) = L_{\Sigma}$. Accordingly, we obtain the following consequence.

**Corollary 6.12.** For each recursively enumerable language $L \subseteq \Sigma_1^*$, there are an alphabet $\Sigma$ containing $\Sigma_T$, a language $L_1 \in L_{\omega 1}(stl-det-local-CD-R(1))$, and a regular language $R \subseteq (\Sigma \cup \tilde{\Sigma})^*$ such that $L = \Pr_{\Sigma_T}(L_1 \cap R)$.

Thus, the closure of the language class $L_{\omega 1}(stl-det-local-CD-R(1))$ under intersection with regular sets and projections already yields all recursively enumerable languages.
7. Decision problems

Each cycle of a deterministic restarting automaton can be simulated in linear time by a Turing machine. As each cycle is strictly length-reducing, it follows that a stl-det-local-CD-R(1)-system can be simulated by a nondeterministic Turing machine in quadratic time using linear space. In fact, a stl-det-local-CD-R(1)-system can be simulated by a nondeterministic shrinking RRWW-automaton, which yields the following result (see [11] and [17]).

Proposition 7.1. \( L = \{(\text{stl-det-local-CD-R}(1)) \subseteq \text{NTIME}(n^2) \cap \text{DSPACE}(n) \}, \) that is, the membership problem for the language \( L = \{(\text{stl-det-local-CD-R}(1), M) \} \) of a stl-det-local-CD-R(1)-system \( M \) can be solved nondeterministically in quadratic time and deterministically in linear space.

Theorem 4.9 yields an effective construction of a finite-state acceptor \( B \) from a stl-det-local-CD-R(1)-system \( M \) such that the language \( E = L(B) \) is a subset of the language \( L = \{(\text{stl-det-local-CD-R}(1), M) \} \) that is letter-equivalent to \( L \). Hence, \( E \) is non-empty if and only if \( L \) is non-empty, and \( E \) is infinite if and only if \( L \) is infinite. As the emptiness problem and the finiteness problem are decidable for finite-state acceptors, this immediately yields the following decidability results.

Proposition 7.2. The following decision problems are effectively decidable:

Instance: A stl-det-local-CD-R(1)-system \( M \).

Question 1: Is the language \( L = \{(\text{stl-det-local-CD-R}(1), M) \} \) empty?

Question 2: Is the language \( L = \{(\text{stl-det-local-CD-R}(1), M) \} \) finite?

Thus, the emptiness and the finiteness problem are effectively decidable for stl-det-local-CD-R(1)-systems. On the other hand, it is undecidable in general whether a rational trace language is recognizable (see, e.g., [9]). As a rational subset \( S \) of a trace monoid \( M(D) \) is recognizable if and only if \( \psi(S) \) is a regular language, it follows from Corollary 5.8 that it is undecidable in general whether a given stl-det-local-CD-R(1)-system accepts a regular language, that is, the following decision problem is undecidable in general.

Proposition 7.3. The following decision problem is undecidable in general:

Instance: A stl-det-local-CD-R(1)-system \( M \).

Question: Is the language \( L = \{(\text{stl-det-local-CD-R}(1), M) \} \) regular?

Finally we consider the inclusion problem and the equivalence problem for stl-det-local-CD-R(1)-systems. We will see that these problems are also undecidable. For doing so we need the following notion.

A rational transducer is defined as \( T = (Q, \Sigma, \Delta, q_0, F, E) \), where \( Q \) is a finite set of internal states, \( \Sigma \) is a finite input alphabet, \( \Delta \) is a finite output alphabet, \( q_0 \in Q \) is the initial state, \( F \subseteq Q \) is a set of final states, and \( E \subseteq Q \times \Sigma^* \times \Delta^* \times Q \) is a finite set of transitions.

If \( e = (p_1, u_1, v_1, q_1)(p_2, u_2, v_2, q_2) \cdots (p_n, u_n, v_n, q_n) \in E^* \) is a sequence of transitions, then its label is the pair \( (\ell(e) = (u_1u_2 \cdots u_n, v_1v_2 \cdots v_n)) \subseteq \Sigma^* \times \Delta^* \). By \( \ell(e) \) we denote the first component \( u_1u_2 \cdots u_n \subseteq \Sigma^* \), and by \( \epsilon(e) \) we denote the second component \( v_1v_2 \cdots v_n \subseteq \Delta^* \). The sequence \( e \) above is called a path from \( p_1 \) to \( q_n \), if \( p_{i+1} = q_i \) for all \( i = 1 \ldots , n \). It is called successful if \( p_1 \) is the initial state \( q_0 \), and if \( q_n \) is a final state. By \( A(p, q) \) we denote the set of all paths from \( p \in Q \) to \( q \in Q \), and we define \( A(p, Q') = \bigcup_{q \in Q} A(p, q) \) for all subsets \( Q' \subseteq Q \). Finally, \( T(p, Q') = (\ell(e) \mid e \in A(p, q)) \) and \( T(p, Q') = (\epsilon(e) \mid e \in A(p, Q')) \). Thus, \( A(q_0, F) \) is the set of all successful paths, and \( T(q_0, F) \) is the set of labels of all successful paths. Then \( \text{Rel}(T) = T(q_0, F) \) is called the relation defined by \( T \). For \( u \subseteq \Sigma^* \) and \( v \subseteq \Delta^* \), \( T(u) = \{ v \subseteq \Delta^* \mid u \cdot v \subseteq T(q_0, F) \} \), and \( T^{-1}(v) = \{ u \subseteq \Sigma^* \mid u \cdot v \subseteq T(q_0, F) \} \). Obviously, the domain of \( \text{Rel}(T) \) is the language \( L(T) = \{ u \subseteq \Sigma^* \mid T(u) = \emptyset \} \), which is the set of all input words for which \( T \) has an accepting computation.

As shown in Theorem 6.1 of [3] the relations defined by rational transducers are just the so-called rational relations, that is, the rational subsets of the monoid \( \Sigma^* \times \Delta^* \). According to [10] Theorem 6.3 we have the following undecidability result.

Proposition 7.4. The following version of the universality problem for rational transducers is undecidable in general:

Instance: A rational transducer \( T = (Q, \{\alpha, b\}, \{c\}, q_0, F, E) \).

Question: Is the relation \( \text{Rel}(T) = \{\alpha, b\}^* \times \{c\}^* \) hold?

The language \( I = \text{sh}([\alpha, b]^* \cdot [c]^*) \) is the rational trace language that is obtained from the regular language \( R = [\alpha, b]^* \cdot [c]^* \) and the dependency relation \( D = \{ (\alpha, a), (b, b), (c, c), (a, b), (b, a) \} \) on the alphabet \( \Sigma = \{\alpha, b, c\} \). Hence, there exists a stl-det-local-CD-R(1)-system \( M \) such that \( L = \{(\text{stl-det-local-CD-R}(1), M) \} \) by Theorem 5.5.

Now let \( T = (Q, \{\alpha, b\}, \{c\}, q_0, F, E) \) be a rational transducer. By introducing an intermediate state \( p_t \) for each transition of the form \( t = (p, u, v, q) \) and by replacing \( t \) by the two transitions \( t_1 = (p, u, v, p_t) \) and \( t_0 = (p_t, q, v, q) \) we obtain
a transducer that, in each step, either consumes part of its input or produces an output. Next we split each transition of the form $t_i = (p, u, v, p)$, where $|u| > 1$, into $|u|$ many transitions, each of which just consumes a single letter, and we split each transition of the form $t_i = (p_1, \varepsilon, v, q)$, where $|v| > 1$, into $|v|$ many transitions that each produce just a single letter. The resulting transducer $T'$ can now be viewed as a nondeterministic finite-state acceptor $A'$ with $\varepsilon$-transitions on the alphabet $\Sigma = \{a, b, c\}$ by interpreting each transition of the form $(p, x, \varepsilon, p')$ or $(p, \varepsilon, x, p')$ as a transition $(p, x, p')$.

It follows immediately from the above construction that the language $L' = L(A')$ accepted by $A'$ has the following properties:

1. $L' \subseteq \bigcup_{(u, v) \in \text{Rel}(T)} \text{sh}(u, v)$, and
2. for all $(u, v) \in \text{Rel}(T)$, there exists a word $w \in L'$ such that $\text{Pr}^{1}(w) = u$ and $\text{Pr}^{c}(w) = v$. Here $\text{Pr}^{a,b} : \Sigma^* \rightarrow \{a, b\}^*$ denotes the projection onto $\{a, b\}^*$, and $\text{Pr}^{c} : \Sigma^* \rightarrow \{c\}^*$ denotes the projection onto $\{c\}^*$.

From $A'$ we can construct a deterministic finite-state acceptor $A$ for the language $L'$, and from $A$ we obtain a stl-det-local-CD-R(1)-system $\mathcal{M}$ such that $L_{1} = \bigcup_{w \in L'} \text{sh}(\{a, b\}^*, \{c\}^*)$ by the construction given in the proof of Theorem 5.5. Now we have the following chain of equivalences:

$$L_{1}(\mathcal{M}) = L_{1}(\hat{\mathcal{M}}) \text{ iff } \bigcup_{w \in L'} \text{sh}(\{a, b\}^*, \{c\}^*) \text{ iff } \text{Rel}(T) = \{a, b\}^* \times \{c\}^*.$$ 

As the system $\mathcal{M}$ is effectively constructed from the given transducer $T$, Proposition 7.4 yields the following undecidability results.

**Proposition 7.5.** The following problems are undecidable in general:

**Instance:** Two stl-det-local-CD-R(1)-systems $\mathcal{M}_1$ and $\mathcal{M}_2$.

**Question 1:** Is $L_{1}(\mathcal{M}_1)$ contained in $L_{1}(\mathcal{M}_2)$?

**Question 2:** Are $\mathcal{M}_1$ and $\mathcal{M}_2$ equivalent, that is, does $L_{1}(\mathcal{M}_1) = L_{1}(\mathcal{M}_2)$ hold?

8. **Concluding remarks**

We have seen that the stateless deterministic $R$-automata induce an infinite hierarchy of language classes based on the size of their windows, and we have related this hierarchy to the classical language families of regular and (deterministic) context-free languages. In [13] stateless variants of deterministic $\mathcal{R}\mathcal{R}$-automata have been introduced and studied. It remains to investigate the influence of the size of the read/write window on the expressive power of these automata. This also holds for the nondeterministic variants of stateless $R$- and $\mathcal{R}\mathcal{R}$-automata.

We have then seen that the stl-det-local-CD-R(1)-systems accept a subclass of all semi-linear languages that contains all rational trace languages, but that this subclass is incomparable to the (deterministic) linear languages and context-free languages with respect to inclusion. However, it remains open whether this language class can be characterized through more, traditional, means. In closure or non-closure of the language class $\mathcal{L}$ wrt. operations like inverse morphisms or reversal are still open.

Further, it remains to determine the trade-off between the size of a stl-det-local-CD-R(1)-system on the one hand and the size of a (deterministic or nondeterministic) finite-state acceptor for the same language on the other hand. Also it remains to study the exact degree of complexity for those decision problems that we have shown to be solvable for stl-det-local-CD-R(1)-systems. Finally, one could also study CD-systems of nondeterministic stateless CD-R(1)-systems. Are they more expressive than their locally deterministic counterparts considered in this paper?

**References**


