Theory of Computing Systems (2021) 65:1033–1068
https://doi.org/10.1007/s00224-020-10028-3

On the Expressive Power of Stateless Ordered Restart-Delete Automata

Friedrich Otto[1]

Accepted: 26 December 2020 / Published online: 13 March 2021

Abstract
Stateless ordered restart-delete automata (stl-ORD-automata) are studied. These are obtained from the stateless ordered restarting automata (stl-ORWW-automata) by introducing an additional restart-delete operation, which, based on the surrounding context, deletes a single letter. While the stl-ORWW-automata accept the regular languages, we show that the swift stl-ORD-automata yield a characterization for the class of context-free languages. Here a stl-ORD-automaton is called *swift* if it can move its window to any position after performing a restart. We also study the descriptional complexity of swift stl-ORD-automata and relate them to limited context restarting automata.

Keywords Restarting automaton · Ordered rewriting · Context-free language · Descriptional complexity · Limited context restarting automaton

1 Introduction

The restarting automaton was introduced in [9] to model the linguistic technique of analysis by reduction (see, e.g., [12]). Despite its linguistic motivation, also many classical families of formal languages have been characterized by various types of restarting automata (for a survey see, e.g., [16]). A particularly simple type of restarting automaton is the *ordered restarting automaton* (ORWW-automaton, for short) that has been introduced in [15] in relation with the processing of languages of rectangular pictures.

An ORWW-automaton consists of a finite-state control, a tape with endmarkers, a read-write window of size three, and a (partial) ordering on its tape alphabet. Based

Some of the results of this paper have been announced at SOFSEM 2020 in Limassol, Cyprus, January 2020. An extended abstract appeared in the proceedings of that conference [20] .

✉ Friedrich Otto
f.otto@uni-kassel.de

[1] Fachbereich Elektrotechnik/Informatik, Universität Kassel, 34109 Kassel, Germany

on the actual state and window contents, the automaton can move its window one position to the right and change its state, or it can replace the letter in the middle of the window by a smaller letter and restart, or it can accept. During a restart, the window is moved back to the left end of the tape, and the finite-state control is reset to the initial state. In [17], it is shown that deterministic ORWW-automata (det-ORWW-automata) don't need states and that they characterize the class of regular languages. On the other hand, the nondeterministic ORWW-automata yield an abstract family of languages that is incomparable to the (deterministic) context-free languages, the Church-Rosser languages, and the growing context-sensitive languages with respect to inclusion [10, 11]. However, stateless nondeterministic ORWW-automata (stl-ORWW-automata) are only as expressive as det-ORWW-automata, that is, they yield just another characterization for the regular languages.

In [18] (see, also [19]), the det-ORWW-automaton was extended to the *deterministic ordered restart-delete automaton* (or det-ORD-automaton, for short). This was achieved by introducing an additional restart-delete operation that allows to delete the symbol from the middle of the window and to restart. The det-ORD-automata do not need states, either, and the class of languages they accept properly includes the class of deterministic context-free languages, while it is contained in the intersection of the (unambiguous) context-free languages and the Church-Rosser languages.

Here we turn to the nondeterministic variant of the ordered restart-delete automaton. As this model extends both, the det-ORD-automaton as well as the ORWW-automaton, it is immediate that the resulting language class is quite large. In particular, this class contains languages that are not even growing context-sensitive [10, 11]. Therefore, we restrict our attention to the stateless variant of these automata, the *stateless ordered restart-delete automaton* (or stl-ORD-automaton, for short). In fact, for technical reasons that will be discussed in Section 3, we concentrate on a restricted variant called *swift* stl-ORD-automaton. A swift stl-ORD-automaton can always perform move-right steps unless its window is already at the right end of the tape. While the stl-ORWW-automaton just accepts the regular languages, the swift stl-ORD-automaton yields a characterization for the context-free languages.

We also study the descriptional complexity of stl-ORD-automata, where we use the size of the underlying tape alphabet as a complexity measure for a stateless ORWW- or ORD-automaton. Based on our constructions we show that the increase in alphabet size that is required when turning a stl-ORD-automaton into a stl-ORWW-automaton for the same (regular) language cannot be bounded from above by any recursive function. That is, we obtain a non-recursive trade-off for this conversion.

Finally, we relate the swift stl-ORD-automata to a new class of limited context restarting automata. A limited context restarting automaton can be seen as a certain type of string rewriting system [2], where the accepted language is defined as the set of all input words that are reducible to the empty word (see, e.g., [1]). Various types of limited context restarting automata have been defined and used to characterize some of the classes of the Chomsky hierarchy [21]. Here we show that the new class of limited context restarting automata that is obtained from the swift stl-ORD-automata

is a proper subclass of the limited context restarting automata of type $\mathcal{R}'_1$. In addition, this new class is incomparable under inclusion to the limited context restarting automata of type $\mathcal{R}'_2$, although it has exactly the same expressive capacity.

This paper is structured as follows. In Section 2, we define the stl-ORD-automaton, we illustrate it by a detailed example, and we establish a kind of normal form result for stl-ORD-automata. In the next section, we prove that each swift stl-ORD-automaton can be simulated by a pushdown automaton (PDA), which implies that all languages accepted by swift stl-ORD-automata are necessarily context-free. Then, in Section 4, we show conversely that each context-free language is accepted by a swift stl-ORD-automaton. This is done by providing a simulation of a PDA by a swift stl-ORD-automaton. In Section 5, we present the aformentioned non-recursive trade-off and in Section 6, we relate the swift stl-ORD-automata to limited context restarting automata. In the concluding section, we summarize our results and describe the hierarchy of language classes obtained through a detailed diagram.

2 The Stateless Ordered Restart-Delete Automaton

An alphabet Σ is a finite set of letters. For all $n \geq 1$, Σ^n is the set of words over Σ of length n, Σ^+ is the set of non-empty words, and $\Sigma^* = \Sigma^+ \cup \{\lambda\}$, where λ denotes the empty word. For $w \in \Sigma^*$, $|w|$ denotes the length of w. A language over Σ is any subset of Σ^*. Of particular interest are the classes REG, DCFL, CFL, CRL, and GCSL of regular, deterministic context-free, context-free, Church-Rosser [13], and growing context-sensitive languages [3, 6]. Furthermore, for any type of automaton X, $\mathcal{L}(\mathsf{X})$ is used to denote the class of (input) languages that are accepted by automata of that type.

Definition 1 A *stateless ordered restart-delete automaton* (stl-ORD-automaton) has a flexible tape with endmarkers and a read/write window of size 3. It is defined by a 6-tuple $M = (\Sigma, \Gamma, \triangleright, \triangleleft, \delta, >)$, where Σ is a finite input alphabet, Γ is a finite tape alphabet such that $\Sigma \subseteq \Gamma$, the symbols $\triangleright, \triangleleft \notin \Gamma$, called *sentinels*, serve as markers for the left and right border of the work space, respectively, $>$ is a *partial ordering* on Γ, and

$$\delta : (((\Gamma \cup \{\triangleright\}) \cdot \Gamma \cdot (\Gamma \cup \{\triangleleft\})) \cup \{\triangleright\triangleleft\}) \to 2^{\Gamma \cup \{\lambda, \mathsf{MVR}\}} \cup \{\mathsf{Accept}\}$$

is the *transition relation* that describes four types of transition steps:

(1) A *move-right step* has the form $\mathsf{MVR} \in \delta(a_1a_2a_3)$, where $a_1 \in \Gamma \cup \{\triangleright\}$ and $a_2, a_3 \in \Gamma$. It causes M to shift the window one position to the right.
(2) A *restart-rewrite step* has the form $b \in \delta(a_1a_2a_3)$, where $a_1 \in \Gamma \cup \{\triangleright\}$, $a_2, b \in \Gamma$, and $a_3 \in \Gamma \cup \{\triangleleft\}$ such that $a_2 > b$ holds. It causes M to replace the symbol

a_2 in the middle of its window by b and to restart, which means that the window is shifted to the left end of the tape so that the first symbol it contains is the left sentinel $\triangleright$. Observe that this operation requires that the newly written letter b is smaller than the letter a_2 being replaced with respect to the partial ordering $>$.

(3) A *restart-delete step* has the form $\lambda \in \delta(a_1a_2a_3)$, where $a_1 \in \Gamma \cup \{\triangleright\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\triangleleft\}$. It causes M to delete the symbol a_2 in the middle of its window and to restart. Through this step, the tape field that contains a_2 is also removed, that is, the length of the tape is reduced.

(4) An *accept step* has the form $\delta(a_1a_2a_3) = \mathsf{Accept}$, where $a_1 \in \Gamma \cup \{\triangleright\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\triangleleft\}$. It causes M to halt and accept. In addition, we allow an accept step of the form $\delta(\triangleright\triangleleft) = \mathsf{Accept}$.

Observe that, for each word $u \in (\Gamma \cup \{\triangleright\}) \cdot \Gamma \cdot (\Gamma \cup \{\triangleleft\})$, either $\delta(u) = \mathsf{Accept}$ or $\delta(u)$ does not contain an accept step. The motivation for this restriction is the observation that the aim of a computation consists in accepting an input word, and hence, once an accept step becomes applicable, it makes no sense to choose a different transition step.

If $\delta(u)$ is undefined for some word u, then M necessarily halts, when its window contains the word u, and we say that M *rejects* in this situation. Furthermore, the letters in $\Gamma \setminus \Sigma$ are called *auxiliary symbols*.

A *configuration* of a stl-ORD-automaton M is a word $\alpha \in \{\triangleright\} \cdot \Gamma^* \cdot \{\triangleleft\}$ in which a factor of length three (or all of α if $|\alpha| \leq 3$) is underlined. Here it is understood that α is the current contents of the tape and that the window contains the underlined factor. A *restarting configuration* has the form $\triangleright\, w \,\triangleleft$, where the prefix of length three is underlined; if $w \in \Sigma^*$, then this restarting configuration is also called an *initial configuration*. A configuration that is reached by an accept step is an *accepting configuration*, denoted by Accept, and a configuration of the form $\alpha_1\underline{a_1a_2a_3}\alpha_2$ such that $\delta(a_1a_2a_3)$ is undefined is a *rejecting configuration*. A *halting configuration* is either an accepting or a rejecting configuration. By $\vdash_M$ we denote the *single-step computation relation* that M induces on its set of configurations. Then $\vdash_M^*$, the reflexive and transitive closure of $\vdash_M$, is the *computation relation* of M.

Any computation of a stl-ORD-automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the window is moved along the tape by MVR steps until a restart-rewrite or a restart-delete step is performed and thus, a new restarting configuration is reached. By $\vdash_M^c$ we denote the execution of a cycle, and $\vdash_M^{c^*}$ is the reflexive transitive closure of $\vdash_M^c$. It is the *reduction relation* that M induces on its set of restarting configurations.

The part of a computation that follows after the last execution of a restart-rewrite or restart-delete operation is called a *tail*. Hence, a tail starts in a restarting configuration, the window is moved along the tape by MVR steps until either an accept step is performed or the automaton gets stuck in a rejecting configuration. Accordingly, we speak of an *accepting tail* or a *rejecting tail*.

A word $w \in \Gamma^*$ is accepted by M, if there exists a computation of M which starts with the restarting configuration $\triangleright\, w \,\triangleleft$ and ends with an accept step. The language

consisting of all words that are accepted by M is denoted by $L_C(M)$. It is called the *characteristic language* of M. The (*input*) *language* $L(M)$ of M is the set of all words $w \in \Sigma^*$ that are accepted by M. Obviously, $L(M) = L_C(M) \cap \Sigma^*$.

As each cycle ends with a rewrite operation, which replaces a symbol a by a symbol b that is strictly smaller than a with respect to the given ordering $>$, or with a delete operation, we see that each computation of M on an input of length n consists of at most $|\Gamma| \cdot n$ many cycles. Each cycle (and each tail) of M can be simulated in linear time by a nondeterministic two-tape Turing machine that stores the prefix α_1 of a configuration $\alpha_1 \underline{a_1 a_2 a_3} \alpha_2$ of M on one of the tapes and the suffix $a_1 a_2 a_3 \alpha_2$ on the other tape. Hence, M can be simulated by a nondeterministic two-tape Turing machine in time $O(n^2)$.

As each det-ORD-automaton can be simulated by a stateless det-ORD-automaton [18], we obtain the following inclusion.

Proposition 1 $\mathcal{L}(\textit{det-ORD}) \subseteq \mathcal{L}(\textit{stl-ORD})$.

The following example illustrates how stl-ORD-automata work.

Example 1 Let $L = \{a^n b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$. It is well-known that L is a context-free language that is not deterministic context-free. Furthermore, this language is not accepted by any ORWW-automaton [10, 11]. However, L is accepted by the stl-ORD-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ that is defined by taking $\Sigma = \{a, b\}$ and $\Gamma = \Sigma \cup \{a_1, e, e_1, f, f_1, f_2\}$, by choosing the partial ordering $>$ such that $a > a_1$, $b > e > e_1$, and $b > f > f_1 > f_2$, and by defining the transition relation δ as follows:

(0) $\delta(xyz) \ni$ MVR for all $x \in \Gamma \cup \{\rhd\}$ and $y, z \in \Gamma$,
(1) $\delta(\rhd\lhd) =$ Accept,
(2) $\delta(ab\lhd) = \{e\}$,
(3) $\delta(bb\lhd) = \{e, f\}$,
(4) $\delta(bbe) \ni e$,
(5) $\delta(abe) \ni e$,
(6) $\delta(bbf) \ni f$,
(7) $\delta(abf) \ni f$,
(8) $\delta(aae) \ni a_1$,
(9) $\delta(\rhd ae) \ni a_1$,
(10) $\delta(a_1 ee) \ni e_1$,
(11) $\delta(a_1 e\lhd) \ni e_1$,
(12) $\delta(aa_1 e_1) \ni \lambda$,
(13) $\delta(\rhd a_1 e_1) \ni \lambda$,
(14) $\delta(ae_1 e) \ni \lambda$,
(15) $\delta(\rhd e_1\lhd) \ni \lambda$,
(16) $\delta(aaf) \ni a_1$,
(17) $\delta(\rhd af) \ni a_1$,
(18) $\delta(a_1 ff) \ni f_1$,
(19) $\delta(f_1 ff) \ni f_2$,
(20) $\delta(f_1 f\lhd) \ni f_2$,
(21) $\delta(a_1 f_1 f_2) \ni \lambda$,
(22) $\delta(aa_1 f_2) \ni \lambda$,
(23) $\delta(\rhd a_1 f_2) \ni \lambda$,
(24) $\delta(af_2 f) \ni \lambda$,
(25) $\delta(\rhd f_2\lhd) \ni \lambda$.

For an input of the form $a^m b^n$, first the factor b^n is rewritten, from right to left, into e^n or into f^n using lines (2) to (7). In the former case, it is then checked whether $m = n$ by alternatingly rewriting the last letter a into a_1, the first letter e into e_1 and then deleting a_1 and e_1 using lines (8) to (15). In the latter case, it is checked whether $n = 2m$ by alternatingly rewriting the last letter a into a_1 and the first factor ff into $f_1 f_2$ and then deleting f_1, a_1, and f_2 using lines (16) to (25). For example, given $w = aabbbb$ as input, M can execute the following computation, where we attach

the number of the line used as an index to the computation relation $\vdash$:

$$
\begin{array}{llllll}
\underline{\rhd aa}bbbbb\lhd & \vdash^{5}_{(0)} & \rhd aabbb\underline{bb\lhd} & \vdash_{(3)} & \underline{\rhd aa}bbbbf\lhd & \vdash^{4}_{(0)} \rhd aabbb\underline{bf\lhd} \\
 & \vdash_{(6)} & \underline{\rhd aa}bbfff\lhd & \vdash^{*} & \underline{\rhd aa}ffff\lhd & \vdash_{(0)} \rhd\underline{aa}fffff\lhd \\
 & \vdash_{(16)} & \underline{\rhd aa_1}ffff\lhd & \vdash^{2}_{(0)} & \rhd a\underline{a_1}fffff\lhd & \vdash_{(18)} \underline{\rhd aa_1}f_1fff\lhd \\
 & \vdash^{3}_{(0)} & \rhd aa_1\underline{f_1}ff\lhd & \vdash_{(19)} & \underline{\rhd aa_1}f_1f_2ff\lhd & \vdash^{2}_{(0)} \rhd aa_1f_1\underline{f_2}ff\lhd \\
 & \vdash_{(21)} & \underline{\rhd aa_1}f_2ff\lhd & \vdash_{(0)} & \rhd a\underline{a_1f_2}ff\lhd & \vdash_{(22)} \underline{\rhd af_2}ff\lhd \\
 & \vdash_{(0)} & \rhd\underline{af_2}f\lhd & \vdash_{(24)} & \underline{\rhd af}f\lhd & \vdash^{*} \underline{\rhd f_2\lhd} \\
 & \vdash_{(25)} & \underline{\rhd\lhd} & \vdash_{(1)} & \mathsf{Accept}. &
\end{array}
$$

It follows that $L(M) = L$.

Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-ORD-automaton. A word $w \in L(M)$ is first rewritten into a word $z \in \Gamma^*$ through a sequence of cycles, and then M executes an accepting tail computation on the tape contents $\rhd z\lhd$. Actually, we can require that all accepting configurations of a stl-ORD-automaton are of a very restricted form.

Proposition 2 *From a given stl-ORD-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, one can construct a stl-ORD-automaton $M' = (\Sigma, \Delta, \rhd, \lhd, \delta', >')$ such that $\Delta = \Gamma \cup \{\#\}$, where $\#$ is a new auxiliary symbol, $\delta'(\rhd\#\lhd) = \mathsf{Accept}$ is the only accept step of M', and $L(M') = L(M) \setminus \{\lambda\}$.*

Proof Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-ORD-automaton and let $\#$ be a new auxiliary symbol. The stl-ORD-automaton M' is constructed by replacing each accept step of M by a rewrite-restart step that produces an occurrence of the symbol $\#$. This symbol is then used as context in delete-restart steps that delete all other letters until the configuration $\underline{\rhd\#\lhd}$ is reached.

Accordingly, we define the stl-ORD-automaton $M' = (\Sigma, \Delta, \rhd, \lhd, \delta', >')$ by taking $\Delta = \Gamma \cup \{\#\}$, by defining $>' = > \cup (\Gamma \times \{\#\})$, and by defining the transition relation δ' as follows, where $a, b, c \in \Gamma$:

$$
\begin{array}{lll}
(1)\ \delta'(abc) = \delta(abc), & & \text{if } \delta(abc) \neq \mathsf{Accept}, \\
(2)\ \delta'(abc) = \{\#\}, & & \text{if } \delta(abc) = \mathsf{Accept}, \\
(3)\ \delta'(ab\lhd) = \delta(ab\lhd), & & \text{if } \delta(ab\lhd) \neq \mathsf{Accept}, \\
(4)\ \delta'(ab\lhd) = \{\#\}, & & \text{if } \delta(ab\lhd) = \mathsf{Accept}, \\
(5)\ \delta'(\rhd bc) = \delta(\rhd bc), & & \text{if } \delta(\rhd bc) \neq \mathsf{Accept}, \\
(6)\ \delta'(\rhd bc) = \{\#\}, & & \text{if } \delta(\rhd bc) = \mathsf{Accept}, \\
(7)\ \delta'(\rhd b\lhd) = \delta(\rhd b\lhd) \setminus \{\lambda\}, & & \text{if } \delta(\rhd b\lhd) \cap \Gamma \neq \emptyset \text{ and} \\
 & & (\lambda \notin \delta(\rhd b\lhd) \text{ or } \delta(\rhd\lhd) \neq \mathsf{Accept}), \\
(8)\ \delta'(\rhd b\lhd) = \{\#\}, & & \text{if } \delta(\rhd b\lhd) = \mathsf{Accept}, \\
(9)\ \delta'(\rhd b\lhd) = (\delta(\rhd b\lhd) \setminus \{\lambda\}) \cup \{\#\}, & & \text{if } \lambda \in \delta(\rhd b\lhd) \text{ and } \delta(\rhd\lhd) = \mathsf{Accept},
\end{array}
$$

$$
\begin{array}{llll}
(10)\ \delta'(\rhd\#b) = \{\mathsf{MVR}\}, & (13)\ \delta'(\#bc) = \{\lambda\}, \\
(11)\ \delta'(ab\#) = \{\lambda\}, & (14)\ \delta'(\#b\lhd) = \{\lambda\}, \\
(12)\ \delta'(\rhd b\#) = \{\lambda\}, & (15)\ \delta'(\rhd\#\lhd) = \mathsf{Accept}.
\end{array}
$$

From the definition of δ', we see immediately that $\lambda \notin L(M')$. Let $w = a_1a_2\cdots a_n \in L(M)$, where $a_1, a_2, \ldots, a_n \in \Sigma$ and $n \geq 1$. Then M has an accepting computation

on input w, that is, w is rewritten into a word $z \in \Gamma^*$, which is then accepted in a tail computation. If $z \neq \lambda$, then by using lines (1), (3), (5), (7) and (9), M' can execute the same cycles as M, in this way rewriting w into z. The accept step of M yields a rewrite step of M' that produces an occurrence of the new auxiliary symbol # (by (2), (4), (6), or (8)), and then M' can rewrite its complete tape contents into the configuration $\underline{\triangleright\#\triangleleft}$ by lines (10) to (14). If $z = \lambda$, then M' produces the tape contents $\triangleright b\triangleleft$, from which $\triangleright z\triangleleft = \triangleright\triangleleft$ is obtained through a delete step of M. In this situation M' accepts by lines (9) and (15). Thus, we see that $L(M) \setminus \{\lambda\} \subseteq L(M')$ holds.

On the other hand, let $w \in L(M')$. Then w is non-empty, and we see from the definition of M' that $\triangleright w\triangleleft \vdash_{M'}^{c^*} \underline{\triangleright\#\triangleleft}$ holds. As w does not contain any occurrence of the special symbol #, M' must introduce an occurrence of this symbol in the course of its accepting computation. Hence, this computation has the form

$$\triangleright w\triangleleft \vdash_{M'}^{c^*} \triangleright z\triangleleft = \triangleright z_1 b z_2\triangleleft \vdash_{M'}^{c} \triangleright z_1\#z_2\triangleleft \vdash_{M'}^{c^*} \underline{\triangleright\#\triangleleft}$$

for some word $z = z_1 b z_2$ ($z_1, z_2 \in \Gamma^*$ and $b \in \Gamma$), where the first occurrence of the symbol # is created by an application of line (2), (4), (6), (8), or (9). This in turn implies that M can execute the accepting computation $\triangleright w\triangleleft \vdash_M^{c^*} \triangleright z\triangleleft \vdash_M^* \mathsf{Accept}$, which shows that $w \in L(M)$. Hence, $L(M') = L(M) \setminus \{\lambda\}$ follows. □

3 Swift Stl-ORD-Automata Only Accept Context-Free Languages

We would like to show that each language accepted by a stl-ORD-automaton is necessarily context-free. To prove this result, we would simulate the accepting computations of a stl-ORD-automaton by a (nondeterministic) PDA. For this simulation, we would use an extension of the simulation that is given in [18] (see also [19]) for simulating a stl-det-ORD-automaton by a PDA. Unfortunately, that approach leads to a serious technical problem (see the paragraph following Example 2 below). As currently we do not see a way to overcome this problem, we restrict our attention to just a subclass of stl-ORD-automata, the *swift-ORD-automata*.

We first describe the data structure that will be used for the simulation.

Definition 2 Let $M = (\Sigma, \Gamma, \triangleright, \triangleleft, \delta, >)$ be a stl-ORD-automaton, and let $w = w_1 w_2 \cdots w_n$, where $n \geq 1$ and $w_1, w_2, \ldots, w_n \in \Gamma$. To encode the computations of M for the word w in a compact way, we introduce a 3-tuple of vectors $T_i = [L_i, W_i, R_i]$ for each letter w_i of w, $1 \leq i \leq |w|$, where

- W_i is a sequence $(x_{i,1}, x_{i,2}, \ldots, x_{i,j_i}, x_{i,j_i+1})$ over $\Gamma \cup \{\lambda\}$ such that $j_i \geq 0$, $w_i = x_{i,1} > x_{i,2} > \cdots > x_{i,j_i}$ and $((x_{i,j_i+1} \in \Gamma$ and $x_{i,j_i} > x_{i,j_i+1})$ or $x_{i,j_i+1} = \lambda)$,
- L_i is a sequence of letters $(y_{i,1}, y_{i,2}, \ldots, y_{i,j_i})$ over $\Gamma \cup \{\triangleright\}$, and
- R_i is a sequence of letters $(z_{i,1}, z_{i,2}, \ldots, z_{i,j_i})$ over $\Gamma \cup \{\triangleleft\}$ such that $\delta(y_{i,r} x_{i,r} z_{i,r}) \ni x_{i,r+1}$ holds for all $r = 1, 2, \ldots, j_i$.

In addition, $y_{1,r} = \triangleright$ for all $r = 1, 2, \ldots, j_1$ and $z_{n,r} = \triangleleft$ for all $r = 1, 2, \ldots, j_n$.

The idea is that W_i encodes the sequence of letters that are produced by M in an accepting computation for a particular field, and L_i and R_i encode the information on the neighboring letters to the left and to the right that are used to perform the corresponding rewrite steps. For example, the triple $(y_{i,1}, x_{i,1}, z_{i,1}) \in (L_i, W_i, R_i)$ means that $x_{i,1}$ is rewritten into $x_{i,2}$ by the transition $x_{i,2} \in \delta(y_{i,1}x_{i,1}z_{i,1})$. We say that $x_{i,1}$ is rewritten into $x_{i,2}$ with *left context* $y_{i,1}$ and *right context* $z_{i,1}$. In particular, if $x_{i,j_i+1} = \lambda$, then the transition $\lambda \in \delta(y_{i,j_i}x_{i,j_i}z_{i,j_i})$ is used to delete the letter x_{i,j_i}. The restrictions on the elements of L_1 and R_n just express the fact that w_1 is the first letter, and so its left neighboring field contains the left sentinel $\triangleright$, while w_n is the last letter, and so its right neigboring field contains the right sentinel $\triangleleft$.

We illustrate this definition by a simple example. As the purpose of this example is simply to explain the dynamics of the simulation, the language accepted by the given automaton is not of importance.

Example 2 Let M be the stl-ORD-automaton on the input alphabet $\Sigma = \{a_1, a_2, a_3, a_4, a_5\}$, the tape alphabet $\Gamma = \Sigma \cup \{b_1, b_2, b_3, b_4, \#\}$, and the ordering $a_i > b_i > \#\,(1 \leq i \leq 4)$, where the transition relation is given by the following table:

$$
\begin{array}{lll}
\delta(\triangleright a_1 a_2) = \{\mathsf{MVR}, b_1\}, & \delta(a_1 a_2 a_3) = \{\mathsf{MVR}\}, & \delta(a_2 a_3 a_4) = \{b_3\}, \\
\delta(a_1 a_2 b_3) = \{b_2\}, & \delta(\triangleright a_1 b_2) = \{\mathsf{MVR}\}, & \delta(a_1 b_2 b_3) = \{\lambda\}, \\
\delta(\triangleright a_1 b_3) = \{\mathsf{MVR}, b_1\}, & \delta(b_1 b_3 a_4) = \{\mathsf{MVR}\}, & \delta(b_3 a_4 a_5) = \{\mathsf{MVR}, b_4\}, \\
\delta(a_4 a_5 \triangleleft) = \{\lambda\}, & \delta(b_3 a_4 \triangleleft) = \{b_4\}, & \delta(b_3 b_4 \triangleleft) = \{\lambda\}, \\
\delta(\triangleright b_1 b_3) = \{\mathsf{MVR}\}, & \delta(b_1 b_3 b_4) = \{\mathsf{MVR}\}, & \delta(b_1 b_3 \triangleleft) = \{\#\}, \\
\delta(\triangleright b_1 \#) = \{\lambda\}, & \delta(\triangleright \# \triangleleft) = \mathsf{Accept}. &
\end{array}
$$

Given the word $w = a_1a_2a_3a_4a_5$ as input, M can execute the following accepting computation:

$$
\begin{array}{llll}
\underline{\triangleright a_1 a_2} a_3 a_4 a_5 \triangleleft & \vdash_M \triangleright \underline{a_1 a_2 a_3} a_4 a_5 \triangleleft & \vdash_M \triangleright a_1 \underline{a_2 a_3 a_4} a_5 \triangleleft & \vdash_M \triangleright \underline{a_1 a_2} b_3 a_4 a_5 \triangleleft \\
& \vdash_M \triangleright \underline{a_1 a_2} b_3 a_4 a_5 \triangleleft & \vdash_M \triangleright \underline{a_1 b_2} b_3 a_4 a_5 \triangleleft & \vdash_M \triangleright \underline{a_1 b_2 b_3} a_4 a_5 \triangleleft \\
& \vdash_M \triangleright \underline{a_1 b_3} a_4 a_5 \triangleleft & \vdash_M \triangleright \underline{b_1 b_3} a_4 a_5 \triangleleft & \vdash_M \triangleright \underline{b_1 b_3 a_4} a_5 \triangleleft \\
& \vdash_M \triangleright b_1 \underline{b_3 a_4 a_5} \triangleleft & \vdash_M \triangleright b_1 \underline{b_3 a_4 a_5} \triangleleft & \vdash_M \triangleright \underline{b_1 b_3 a_4} \triangleleft \\
& \vdash_M \triangleright \underline{b_1 b_3 a_4} \triangleleft & \vdash_M \triangleright b_1 \underline{b_3 a_4} \triangleleft & \vdash_M \triangleright \underline{b_1 b_3 b_4} \triangleleft \\
& \vdash_M \triangleright \underline{b_1 b_3 b_4} \triangleleft & \vdash_M \triangleright b_1 \underline{b_3 b_4} \triangleleft & \vdash_M \triangleright \underline{b_1 b_3} \triangleleft \\
& \vdash_M \triangleright \underline{b_1 b_3} \triangleleft & \vdash_M \triangleright \underline{b_1 \#} \triangleleft & \vdash_M \underline{\triangleright \# \triangleleft} \\
& \vdash_M \mathsf{Accept}. & &
\end{array}
$$

For this computation, we obtain the following sequence of triples:

$$
\left[\begin{array}{ccc|ccc|ccc|ccc|ccc}
L_1 & W_1 & R_1 & L_2 & W_2 & R_2 & L_3 & W_3 & R_3 & L_4 & W_4 & R_4 & L_5 & W_5 & R_5 \\
\triangleright & a_1 & b_3 & a_1 & a_2 & b_3 & a_2 & a_3 & a_4 & b_3 & a_4 & \triangleleft & a_4 & a_5 & \triangleleft \\
\triangleright & b_1 & \# & a_1 & b_2 & b_3 & b_1 & b_3 & \triangleleft & b_3 & b_4 & \triangleleft & & \lambda & \\
 & \lambda & & & \lambda & & & \# & & & \lambda & & & &
\end{array}\right]
$$

These triples do not only record the history of rewrite and delete steps that have been applied to the various input letters, but the sequence of these triples also provides information on the ordering in which these rewrite and delete steps have been executed at neighboring positions. In fact, if $[L_i, W_i, R_i]_{i=1,2,\ldots,n}$ is a sequence of triples that describe an accepting computation of M on input $w = w_1w_2 \cdots w_n$, then

we can extract the complete sequence of rewrite and delete operations of M from this sequence. To see this, we inspect the above sequence.

First, we see that $w = a_1a_2a_3a_4a_5$. Furthermore, from the first triple $[L_1, W_1, R_1] = \begin{bmatrix} \rhd\ a_1\ b_3 \\ \rhd\ b_1\ \# \\ \lambda \end{bmatrix}$, we conclude that M executes the rewrite step $b_1 \in \delta(\rhd a_1 b_3)$ and the delete step $\lambda \in \delta(\rhd b_1 \#)$ at position 1. As this is the first position, we know that the left neighboring field contains the left sentinel $\rhd$, that is, the left contexts of these steps coincide with (the contents of) the left neighboring field. We express this fact by saying that these steps *match left*.

Next, we consider the second triple $[L_2, W_2, R_2] = \begin{bmatrix} a_1\ a_2\ b_3 \\ a_1\ b_2\ b_3 \\ \lambda \end{bmatrix}$. Thus, at position 2, M executes the sequence of rewrite and delete steps $b_2 \in \delta(a_1a_2b_3)$ and $\lambda \in \delta(a_1b_2b_3)$. As initially field 1 contains the letter a_1, we see that the left contexts of these steps coincide with their left neigboring field, that is, they match left.

At position 3, M executes the sequence of rewrite steps $b_3 \in \delta(a_2a_3a_4)$ and $\# \in \delta(b_1b_3\lhd)$. As the initial letter at position 2 is a_2, we see that the first of these rewrite steps matches left. After a_3 has been rewritten into b_3, the right contexts of the rewrite and delete steps at position 2 coincide with (the contents) of their right neighboring field, that is, they *match right*. Thus, all rewrite and delete steps at position 2 match left and right, which means that all their left and right contexts have been verified. In particular, this implies that these steps can be executed *after* a_3 has been rewritten into b_3. As W_2 ends with λ, we can now remove the triple $[L_2, W_2, R_2]$ from the sequence. But then the triple $[L_3, W_3, R_3]$ becomes the right neighbor of $[L_1, W_1, R_1]$, which shows that now the first rewrite at position 1 also matches right. Hence, a_1 can now be rewritten into b_1, and after that, the second rewrite step at position 3 matches left, too. So the second rewrite step at position 3 is executed *after* the first rewrite step at position 1. After b_3 has been rewritten into #, also the final delete step at position 1 matches right, that is, all rewrite and delete steps at position 1 match left and right. As W_1 ends with λ, we can now remove the triple $[L_1, W_1, R_1]$ from the sequence.

At position 4, M executes the rewrite step $b_4 \in \delta(b_3a_4\lhd)$ and the delete step $\lambda \in \delta(b_3b_4\lhd)$. As this position initially contains the letter a_4, we conclude that the first rewrite step at position 3 matches right. Hence, this rewrite step is executed *before* the rewrite step at position 4. After a_3 has been rewritten into b_3, the steps at position 4 match left.

Finally, at position 5, M only executes the delete step $\lambda \in \delta(a_4a_5\lhd)$, which matches left and right. In particular, we see that this step is executed *before* the first rewrite step at position 4. As W_5 ends with λ, the triple $[L_5, W_5, R_5]$ can be removed, which means that the right sentinel $\lhd$ is now the new right neighbor of position 4. This in turn implies that the rewrite and delete steps at position 4 match right. Thus, these steps can now be executed and the triple $[L_4, W_4, R_4]$ can be removed as well. But then $\lhd$ is the new right neighbor of position 3, and hence, the second rewrite step at position 3 matches right.

Thus, only the triple $[L_3, W_3, R_3]$ remains, and all its rewrite steps have been matched left and right. In fact, from the above considerations we can conclude that the rewrite and delete transitions encoded in the given sequence of triples are executed in the following order:

$$\begin{array}{lll} 1.\ a_3 \to b_3, & 4.\ a_1 \to b_1, & 7.\ b_4 \to \lambda, \\ 2.\ a_2 \to b_2, & 5.\ a_5 \to \lambda, & 8.\ b_3 \to \#, \\ 3.\ b_2 \to \lambda, & 6.\ a_4 \to b_4, & 9.\ b_1 \to \lambda. \end{array}$$

This is actually the same order as the one in the above computation of M. Finally, as $\mathsf{Accept} \in \delta(\rhd\#\lhd)$, we see that the above sequence of triples does indeed describe the accepting computation of M on input $a_1a_2a_3a_4a_5$ *under the assumption that M can always move to the required position by a sequence of move-right steps*.

In the case of stl-ORWW-automata, that is, when no letter can be deleted, the required MVR-steps can easily be inferred from the corresponding sequence of triples considered, and so, by checking the transition relation of M, it can be verified whether they are actually possible. However, for stl-ORD-automata, that is, when delete operations can be used, this is not at all clear, as particular move-right steps may or may not use letters at positions that are at some point deleted. Therefore, in order to turn our idea into a correct simulation of a stl-ORD-automaton by a PDA, we only consider a restricted class of stl-ORD-automata.

Definition 3 A stl-ORD-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ is called *swift* (or simply a *swift-ORD-automaton*) if $\mathsf{MVR} \in \delta(a_1a_2a_3)$ for all $a_1 \in \Gamma \cup \{\rhd\}$ and all $a_2, a_3 \in \Gamma$. By $\mathsf{swift\text{-}ORD}$ we denote the class of all swift-ORD-automata.

From a restarting configuration $\rhd w\lhd$, a swift-ORD-automaton M can move its window to any position on the tape. Thus, a computation of M cannot be blocked by a factor across which M cannot move its window. The stl-ORD-automaton M of Example 1 is actually a swift-ORD-automaton. Furthermore, Proposition 2 carries over to swift-ORD-automata. The aim of this section is the following result.

Theorem 1 $\mathcal{L}(\mathsf{swift\text{-}ORD}) \subseteq \mathsf{CFL}$.

For establishing this result, we proceed as follows. First, motivated by the discussion in Example 2, we define the notion of compatibility for sequences of triples $[L, W, R]$. After illustrating this definition by an example, we prove that, for a given input $w = w_1w_2 \cdots w_n$, there exists a compatible sequence of triples if and only if w is accepted by the swift-ORD-automaton M considered. Then we describe a PDA P that, on input $w = w_1w_2 \cdots w_n$, guesses a corresponding sequence of triples, checks whether this sequence is compatible, and accepts in the affirmative. Together these results imply that $L(M)$ is accepted by the PDA P, which completes the proof of Theorem 1.

Definition 4 Let $M = (\Sigma, \Gamma \cup \{\#\}, \rhd, \lhd, \delta, >)$ be a swift-ORD-automaton for which $\delta(\rhd\#\lhd) = \mathsf{Accept}$ is the only accept step, and let $w = w_1w_2 \cdots w_n$, where

$n \geq 1$ and $w_1, w_2, \ldots, w_n \in \Gamma$. Furthermore, let $\Pi = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1})$ be a sequence of triples as in Definition 2. Here $\Pi_0 = [L_0, W_0, R_0] = [\emptyset, (\triangleright), \emptyset]$, $\Pi_{n+1} = [L_{n+1}, W_{n+1}, R_{n+1}] = [\emptyset, (\triangleleft), \emptyset]$, and, for all $i = 1, 2, \ldots, n$,

$$\Pi_i = [L_i, W_i, R_i] = \begin{bmatrix} a_{i,1} & b_{i,1} & c_{i,1} \\ a_{i,2} & b_{i,2} & c_{i,2} \\ \cdots & \cdots & \cdots \\ a_{i,j_i} & b_{i,j_i} & c_{i,j_i} \\ & x_i & \end{bmatrix},$$

where $j_i \geq 0, a_{i,r}, b_{i,r}, c_{i,r} \in \Gamma$ for all $r = 1, 2, \ldots, j_i, b_{i,1} = w_i$, and $x_i \in \Gamma \cup \{\lambda\}$.

(a) A pair (Π_i, Π_{i+1}) of neighboring triples from the sequence Π is said to be *compatible* if one of the following five conditions is satisfied:

(i) If $i = 0$, then $a_{1,r} = \triangleright$ for all $r = 1, 2, \ldots, j_1$. This means that all rewrite (and delete) transitions at position 1 have left context $\triangleright$. In the affirmative, all left contexts of Π_1 are marked.

(ii) If $i = n$, then $c_{n,r} = \triangleleft$ for all $r = 1, 2, \ldots, j_n$. This means that all rewrite (and delete) transitions at position n have right context $\triangleleft$. In the affirmative, all right contexts of Π_n are marked.

(iii) If $x_i \neq \lambda \neq x_{i+1}$, then all rewrite transitions of Π_i and all rewrite transitions of Π_{i+1} can be put into a linear order in such a way that the respective right contexts of the rewrite transitions of Π_i coincide with the actual letters at position $i + 1$ and the respective left contexts of the rewrite transitions of Π_{i+1} coincide with the actual letters at position i. As in Example 2 we say that the rewrite transitions of Π_i *match right* and the rewrite transitions of Π_{i+1} *match left*. Observe that each rewrite transition at position i $(i + 1)$ changes the letter at that position, which means that the possible left (right) context for the rewrite transitions at position $i + 1$ (i) is changed. The matching right contexts of Π_i and the matching left contexts of Π_{i+1} are marked.

(iv) If $x_i \neq \lambda$ and $x_{i+1} = \lambda$, then there exists a maximal integer $r \in \{0, 1, \ldots, j_i\}$ such that the first r rewrite transitions of Π_i and all j_{i+1} rewrite (and delete) transitions of Π_{i+1} can be put into a linear order in such a way that the respective right contexts of the first r rewrite transitions of Π_i coincide with the actual letters at position $i + 1$ and the respective left contexts of the rewrite (and delete) transitions of Π_{i+1} coincide with the actual letters at position i. We say that the first r rewrite transitions of Π_i *match right* and the rewrite (and delete) transitions of Π_{i+1} *match left*. The matching right contexts of Π_i and the matching left contexts of Π_{i+1} are marked.

(v) If $x_i = \lambda$ and $x_{i+1} \neq \lambda$, then there exists a maximal integer $r \in \{0, 1, \ldots, j_{i+1}\}$ such that the first r rewrite transitions of Π_{i+1} and all j_i rewrite (and delete) transitions of Π_i can be put into a linear order in such a way that the respective right contexts of the j_i rewrite (and delete) transitions of Π_i coincide with the actual letters at position $i + 1$ and

the respective left contexts of the first r rewrite transitions of Π_{i+1} coincide with the actual letters at position i. We say that the rewrite (and delete) transitions of Π_i *match right* and the first r rewrite transitions of Π_{i+1} *match left*. The matching right contexts of Π_i and the matching left contexts of Π_{i+1} are marked.

(vi) If $x_i = \lambda$ and $x_{i+1} = \lambda$, then there are two possibilities.

(1) There exists a maximal integer $r \in \{0, 1, \ldots, j_i\}$ such that the first r rewrite transitions of Π_i match right and all j_{i+1} rewrite (and delete) transitions of Π_{i+1} match left. Again, the matching right contexts of Π_i and the matching left contexts of Π_{i+1} are marked.

(2) There exists a maximal integer $r \in \{0, 1, \ldots, j_{i+1}\}$ such that the first r rewrite transitions of Π_{i+1} match left and all j_i rewrite (and delete) transitions of Π_i match right. The matching right contexts of Π_i and the matching left contexts of Π_{i+1} are marked.

(b) The sequence of triples $\Pi = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1})$ is called *compatible* if it satisfies all of the following conditions:

(1) There exists an index $s \in \{1, 2, \ldots, n\}$ such that $x_s = \#$, while $x_i = \lambda$ for all $i \in \{1, 2, \ldots, n\} \setminus \{s\}$.

(2) Each pair of neighboring triples (Π_i, Π_{i+1}), $i = 0, 1, 2, \ldots, n$, is compatible.

(3) From Π a sequence of *remainder triples* Π' is constructed as follows. Each triple Π_i, $i \in \{1, 2, \ldots, n\} \setminus \{s\}$, for which all left and all right contexts have been marked, is deleted. For each other triple, all rewrite transitions are deleted for which the corresponding left and right contexts have both been marked. The resulting sequence of triples is the sequence Π'. If Π' does not contain any rewrite or delete transitions anymore, then it is compatible; otherwise, it is now checked recursively that the sequence Π' is compatible. For checking the conditions in (a) for Π', the marked contexts are ignored.

We illustrate this definition by taking another look at our example.

Example 2 (cont.) For the automaton M considered above, we have the following sequence of triples $\Pi = (\Pi_0, \Pi_1, \Pi_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6) =$

$$\left[\begin{array}{c} \emptyset \ (\triangleright) \ \emptyset \end{array}\right] \left[\begin{array}{ccc} \triangleright & a_1 & b_3 \\ \triangleright & b_1 & \# \\ & \lambda & \end{array}\right] \left[\begin{array}{ccc} a_1 & a_2 & b_3 \\ a_1 & b_2 & b_3 \\ & \lambda & \end{array}\right] \left[\begin{array}{ccc} a_2 & a_3 & a_4 \\ b_1 & b_3 & \triangleleft \\ & \# & \end{array}\right] \left[\begin{array}{ccc} b_3 & a_4 & \triangleleft \\ b_3 & b_4 & \triangleleft \\ & \lambda & \end{array}\right] \left[\begin{array}{ccc} a_4 & a_5 & \triangleleft \\ & \lambda & \end{array}\right] \left[\begin{array}{c} \emptyset \ (\triangleleft) \ \emptyset \end{array}\right].$$

The pairs (Π_0, Π_1) and (Π_5, Π_6) are obviously compatible by (i) and (ii). Accordingly, the left contexts in Π_1 and the right contexts in Π_5 are marked.

The pair $(\Pi_1, \Pi_2) = \begin{bmatrix} \triangleright\ a_1\ b_3 \\ \triangleright\ b_1\ \# \\ \lambda \end{bmatrix} \begin{bmatrix} a_1\ a_2\ b_3 \\ a_1\ b_2\ b_3 \\ \lambda \end{bmatrix}$ is compatible by case (vi) (1) for $r = 0$. Here the rewrite and the delete steps at position 2 have left context a_1, which is the current letter at position 1.

The pair $(\Pi_2, \Pi_3) = \begin{bmatrix} a_1\ a_2\ b_3 \\ a_1\ b_2\ b_3 \\ \lambda \end{bmatrix} \begin{bmatrix} a_2\ a_3\ a_4 \\ b_1\ b_3\ \triangleleft \\ \# \end{bmatrix}$ is compatible by case (v) for $r = 1$. Here the first rewrite transition at position 3 comes before the rewrite and delete transitions at position 2.

The pair $(\Pi_3, \Pi_4) = \begin{bmatrix} a_2\ a_3\ a_4 \\ b_1\ b_3\ \triangleleft \\ \# \end{bmatrix} \begin{bmatrix} b_3\ a_4\ \triangleleft \\ b_3\ b_4\ \triangleleft \\ \lambda \end{bmatrix}$ is compatible by case (iv) with $r = 1$. Here the first rewrite transition at position 3 comes before the rewrite and delete transitions at position 4.

Finally, the pair $(\Pi_4, \Pi_5) = \begin{bmatrix} b_3\ a_4\ \triangleleft \\ b_3\ b_4\ \triangleleft \\ \lambda \end{bmatrix} \begin{bmatrix} a_4\ a_5\ \triangleleft \\ \lambda \end{bmatrix}$ is compatible by case (vi) (1) with $r = 0$. Here only the delete transition at position 5 is possible.

By marking all the left and right contexts that correspond to the enabled rewrite and delete transitions, we obtain the following variant of the original sequence, where the marked letters are written in boldface:

$$\left[\ \emptyset\ (\triangleright)\ \emptyset\ \right] \begin{bmatrix} \blacktriangleright\ a_1\ b_3 \\ \blacktriangleright\ b_1\ \# \\ \lambda \end{bmatrix} \begin{bmatrix} \mathbf{a_1}\ a_2\ \mathbf{b_3} \\ \mathbf{a_1}\ b_2\ \mathbf{b_3} \\ \lambda \end{bmatrix} \begin{bmatrix} \mathbf{a_2}\ a_3\ \mathbf{a_4} \\ b_1\ b_3\ \triangleleft \\ \# \end{bmatrix} \begin{bmatrix} \mathbf{b_3}\ a_4\ \triangleleft \\ \mathbf{b_3}\ b_4\ \triangleleft \\ \lambda \end{bmatrix} \begin{bmatrix} \mathbf{a_4}\ a_5\ \blacktriangleleft \\ \lambda \end{bmatrix} \left[\ \emptyset\ (\triangleleft)\ \emptyset\ \right].$$

From this sequence we obtain the sequence of remainder triples

$$\Pi' = (\Pi'_0, \Pi'_1, \Pi'_2, \Pi'_3, \Pi'_4) = \left[\ \emptyset\ (\triangleright)\ \emptyset\ \right] \begin{bmatrix} \blacktriangleright\ a_1\ b_3 \\ \blacktriangleright\ b_1\ \# \\ \lambda \end{bmatrix} \begin{bmatrix} b_1\ b_3\ \triangleleft \\ \# \end{bmatrix} \begin{bmatrix} \mathbf{b_3}\ a_4\ \triangleleft \\ \mathbf{b_3}\ b_4\ \triangleleft \\ \lambda \end{bmatrix} \left[\ \emptyset\ (\triangleleft)\ \emptyset\ \right].$$

Continuing recursively, we see that the pairs (Π'_0, Π'_1) and (Π'_3, Π'_4) are compatible. Furthermore, the pair $(\Pi'_1, \Pi'_2) = \begin{bmatrix} \blacktriangleright\ a_1\ b_3 \\ \blacktriangleright\ b_1\ \# \\ \lambda \end{bmatrix} \begin{bmatrix} b_1\ b_3\ \triangleleft \\ \# \end{bmatrix}$ is compatible by case (v) with $r = 1$. Here the first rewrite step of Π'_1 comes first, the rewrite step of Π'_2 comes second, and then comes the delete step of Π'_1.

Finally, the pair $(\Pi'_2, \Pi'_3) = \begin{bmatrix} b_1\ b_3\ \triangleleft \\ \# \end{bmatrix} \begin{bmatrix} \mathbf{b_3}\ a_4\ \triangleleft \\ \mathbf{b_3}\ b_4\ \triangleleft \\ \lambda \end{bmatrix}$ is compatible by case (iv) with $r = 0$. By marking the left and right contexts of the corresponding rewrite and delete transitions, we obtain the following sequence:

$$\left[\ \emptyset\ (\triangleright)\ \emptyset\ \right] \begin{bmatrix} \blacktriangleright\ a_1\ \mathbf{b_3} \\ \blacktriangleright\ b_1\ \# \\ \lambda \end{bmatrix} \begin{bmatrix} \mathbf{b_1}\ b_3\ \triangleleft \\ \# \end{bmatrix} \begin{bmatrix} \mathbf{b_3}\ a_4\ \blacktriangleleft \\ \mathbf{b_3}\ b_4\ \blacktriangleleft \\ \lambda \end{bmatrix} \left[\ \emptyset\ (\triangleleft)\ \emptyset\ \right].$$

From this sequence we finally get the remainder sequence

$$\Pi'' = (\Pi_0'', \Pi_1'', \Pi_2'') = \left[\begin{array}{ccc} \emptyset & (\rhd) & \emptyset \end{array}\right] \left[\begin{array}{ccc} \mathbf{b_1} & b_3 & \lhd \\ & \# & \end{array}\right] \left[\begin{array}{ccc} \emptyset & (\lhd) & \emptyset \end{array}\right].$$

In this sequence the right context $\lhd$ of Π_1'' is marked and the first line of Π_1'' is deleted. This yields the final sequence $\left[\begin{array}{ccc} \emptyset & (\rhd) & \emptyset \end{array}\right] \left[\begin{array}{ccc} \emptyset & (\#) & \emptyset \end{array}\right] \left[\begin{array}{ccc} \emptyset & (\lhd) & \emptyset \end{array}\right]$, which shows that the original sequence is indeed compatible.

On the other hand, the following sequence is not compatible:

$$\Pi = (\Pi_0, \Pi_1, \Pi_2, \Pi_3, \Pi_4) = \left[\begin{array}{ccc} \emptyset & (\rhd) & \emptyset \end{array}\right] \left[\begin{array}{ccc} \rhd & a_1 & a_2 \\ \rhd & b_1 & \# \\ & \lambda & \end{array}\right] \left[\begin{array}{ccc} a_1 & a_2 & b_3 \\ & b_2 & \end{array}\right] \left[\begin{array}{ccc} \emptyset & (b_3) & \emptyset \end{array}\right] \left[\begin{array}{ccc} \emptyset & (\lhd) & \emptyset \end{array}\right].$$

In fact, the pair (Π_1, Π_2) is not compatible. Indeed, Π_1 tells us that the first rewrite transition at position 1 is $b_1 \in \delta(\rhd a_1 a_2)$, which requires the right context a_2, while Π_2 tells us that the first rewrite transition at position 2 is $b_2 \in \delta(a_1 a_2 b_3)$, which requires the left context a_1. No matter which of these rewrite transitions is applied first, it destroys the required context for the other rewrite transition.

Lemma 1 *Let $M = (\Sigma, \Gamma \cup \{\#\}, \rhd, \lhd, \delta, >)$ be a swift-ORD-automaton as constructed in Proposition 2, and let $w = w_1 w_2 \cdots w_n$, where $n \geq 1$ and $w_1, w_2, \ldots, w_n \in \Gamma \cup \{\#\}$. Then the following statements are equivalent:*

(a) $w \in L_{\mathrm{C}}(M)$.

(b) *There exists a compatible sequence of triples*

$$\Pi = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1}),$$

where $\Pi_0 = [\emptyset, (\rhd), \emptyset]$, $\Pi_{n+1} = [\emptyset, (\lhd), \emptyset]$, and, for all $i = 1, 2, \ldots, n$,

$$\Pi_i = [L_i, W_i, R_i] = \left[\begin{array}{ccc} a_{i,1} & b_{i,1} & c_{i,1} \\ a_{i,2} & b_{i,2} & c_{i,2} \\ \cdots & \cdots & \cdots \\ a_{i,j_i} & b_{i,j_i} & c_{i,j_i} \\ & x_i & \end{array}\right] \text{ such that } b_{i,1} = w_i.$$

Proof Let $M = (\Sigma, \Gamma \cup \{\#\}, \rhd, \lhd, \delta, >)$ be a swift-ORD-automaton as constructed in Proposition 2. Then $\delta(\rhd \# \lhd) = \mathsf{Accept}$ is the only accept step of M, and the letter # can neither be rewritten nor deleted. Let $w = w_1 w_2 \cdots w_n$, where $n \geq 1$ and $w_1, w_2, \ldots, w_n \in \Gamma \cup \{\#\}$.

Claim 1. If $w \in L_{\mathrm{C}}(M)$, then there exists a sequence of triples $\Pi = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1})$ that satisfies all the properties in (b).

Proof If $w \in L_{\mathrm{C}}(M)$, then M has an accepting computation

$$\rhd w \lhd \vdash_M^c \rhd z_1 \lhd \vdash_M^c \cdots \vdash_M^c \rhd z_m \lhd \vdash_M^c \rhd \# \lhd \vdash_M \mathsf{Accept}.$$

By Definition 2, we can associate a triple $\Pi_i = [L_i, W_i, R_i]$ to each letter w_i, $i = 1, 2, \ldots, n$, such that Π_i describes the rewrite and delete steps that are executed at

position i during the above computation. For $i = 1, 2, \ldots, n$, let

$$\Pi_i = [L_i, W_i, R_i] = \begin{bmatrix} a_{i,1} & b_{i,1} & c_{i,1} \\ a_{i,2} & b_{i,2} & c_{i,2} \\ \cdots & \cdots & \cdots \\ a_{i,j_i} & b_{i,j_i} & c_{i,j_i} \\ & x_i & \end{bmatrix},$$

let $\Pi_0 = [\emptyset, (\triangleright), \emptyset]$, and let $\Pi_{n+1} = [\emptyset, (\triangleleft), \emptyset]$. Then $a_{1,j} = \triangleright$ for all $1 \leq j \leq j_1$, $c_{n,j} = \triangleleft$ for all $1 \leq j \leq j_n$, $b_{i,1} = w_i$, $x_i \in \{\#, \lambda\}$, and $b_{i,1} > b_{i,2} > \cdots > b_{i,j_i} > x_i$ for all $1 \leq i \leq n$. Here we extend the partial ordering $>$ by taking $b > \lambda$ for all letters $b \in \Gamma$. Furthermore, as $\delta(\triangleright\#\triangleleft) = \mathsf{Accept}$ is the only accept step of M, there is a unique index $s \in \{1, 2, \ldots, n\}$ such that $x_s = \#$ and $x_i = \lambda$ for all $i \in \{1, 2, \ldots, n\} \setminus \{s\}$.

We consider the sequence of triples $\Pi = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1})$. It remains to prove that this sequence is compatible. From the definition of Π, we see immediately that conditions (i) and (ii) of part (a) and condition (1) of part (b) of Definition 4 are satisfied.

For proving compatibility of Π, we proceed by induction on m, the number of cycles in the above computation. For $m = 1$, there are three cases:

(1) $n = 1$, $w = w_1$, and $\# \in \delta(\triangleright w_1 \triangleleft)$ is the rewrite transition that is used in this cycle. Then $\Pi_1 = \begin{bmatrix} \triangleright & w_1 & \triangleleft \\ & \# & \end{bmatrix}$.

(2) $n = 2$, $w_1 \in \Gamma$, $w_2 = \#$, and $\lambda \in \delta(\triangleright w_1 \#)$ is the delete transition that is used in this cycle. Then $\Pi_1 = \begin{bmatrix} \triangleright & w_1 & \# \\ & \lambda & \end{bmatrix}$ and $\Pi_2 = \begin{bmatrix} \emptyset & (\#) & \emptyset \end{bmatrix}$.

(3) $n = 2$, $w_1 = \#$, $w_2 \in \Gamma$, and $\lambda \in \delta(\# w_2 \triangleleft)$ is the delete transition that is used in this cycle. Then $\Pi_1 = \begin{bmatrix} \emptyset & (\#) & \emptyset \end{bmatrix}$ and $\Pi_2 = \begin{bmatrix} \# & w_2 & \triangleleft \\ & \lambda & \end{bmatrix}$.

In each of these cases it is easily seen that the corresponding sequence Π is compatible.

For the inductive step we scrutinize the first cycle

$$\triangleright w \triangleleft = \underline{\triangleright w_1 w_2} w_3 \cdots w_n \triangleleft \vdash_M^c \triangleright z_1 \triangleleft$$

of the above computation. In this cycle a rewrite or delete transition $b \in \delta(w_{i-1} w_i w_{i+1})$ is executed, where $b \in \Gamma \cup \{\#, \lambda\}$ and $i \in \{1, 2, \ldots, n\}$. Thus, the triples Π_{i-1}, Π_i, and Π_{i+1} look as follows, where $w_0 = \triangleright$ and $w_{n+1} = \triangleleft$:

$$\Pi_{i-1} = \begin{bmatrix} a_{i-1,1} & w_{i-1} & c_{i-1,1} \\ L'_{i-1} & W'_{i-1} & R'_{i-1} \end{bmatrix}, \Pi_i = \begin{bmatrix} w_{i-1} & w_i & w_{i+1} \\ a_{i,2} & b & c_{i,2} \\ L'_i & W'_i & R'_i \end{bmatrix},$$

and

$$\Pi_{i+1} = \begin{bmatrix} a_{i+1,1} & w_{i+1} & c_{i+1,1} \\ L'_{i+1} & W'_{i+1} & R'_{i+1} \end{bmatrix}.$$

Here $L'_{i-1}, W'_{i-1}, R'_{i-1}, L'_i, W'_i, R'_i, L'_{i+1}, W'_{i+1}, R'_{i+1}$ denote the remaining parts of the corresponding sequences.

We now contruct a new sequence Π'. For all $j \in \{0, 1, 2, \ldots, n+1\} \setminus \{i\}$, we take $\Pi'_j = \Pi_j$. Furthermore, we take $\Pi'_i = \begin{bmatrix} a_{i,2} & b & c_{i,2} \\ L'_i & W'_i & R'_i \end{bmatrix}$, if $b \neq \lambda$. In this way, we obtain the sequence $\Pi' = (\Pi'_0, \Pi'_1, \ldots, \Pi'_{i-1}, \Pi'_i, \Pi'_{i+1}, \ldots, \Pi'_n, \Pi'_{n+1})$, if $b \neq \lambda$, or $\Pi' = (\Pi'_0, \Pi'_1 \ldots, \Pi'_{i-1}, \Pi'_{i+1} \ldots, \Pi'_n, \Pi'_{n+1})$, if $b = \lambda$. In either case, the sequence Π' corresponds to the computation

$$\rhd z_1 \lhd = \rhd w_1 \cdots w_{i-1} b w_{i+1} \cdots w_n \lhd \vdash_M^c \cdots \vdash_M^c \rhd z_m \lhd \vdash_M^c \rhd \# \lhd \vdash_M \textsf{Accept}$$

of M. As this computation consists of only $m-1$ cycles. the induction hypothesis yields that the sequence Π' is compatible. Compatibility of Π now follows easily, which completes the proof of Claim 1. □

Claim 2. If there exists a sequence of triples $\Pi = (\Pi_0, \Pi_1, \ldots, \Pi_n, \Pi_{n+1})$ that satisfies all the properties in (b), then $w \in L_C(M)$.

Proof Let $\Pi = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1})$ be a sequence of triples that satisfies all the properties in (b). We proceed by induction on the overall number $R = \sum_{i=1}^{n} j_i$ of rewrite and delete steps encoded in this sequence of triples. If $R = 1$, then either a single rewrite step or a single delete step is described by the given sequence. In the former case, we have $n = 1$ and compatibility of Π implies that $\Pi_1 = \begin{bmatrix} \rhd & w_1 & \lhd \\ & \# & \end{bmatrix}$. Hence, $\underline{\rhd w_1 \lhd} \vdash_M^c \underline{\rhd \# \lhd} \vdash_M \textsf{Accept}$ is an accepting computation of M. In the latter case, we have $n = 2$, and compatibility of Π implies that $\Pi_1 = \begin{bmatrix} \rhd & w_1 & \# \\ & \lambda & \end{bmatrix}$ and $\Pi_2 = \begin{bmatrix} \emptyset & (\#) & \emptyset \end{bmatrix}$ or $\Pi_1 = \begin{bmatrix} \emptyset & (\#) & \emptyset \end{bmatrix}$ and $\Pi_2 = \begin{bmatrix} \# & w_2 & \lhd \\ & \lambda & \end{bmatrix}$. Hence, we have the accepting computation $\underline{\rhd w_1 \#} \lhd \vdash_M^c \underline{\rhd \# \lhd} \vdash \textsf{Accept}$ or $\underline{\rhd \# w_2} \lhd \vdash_M^c \underline{\rhd \# \lhd} \vdash \textsf{Accept}$.

Let us assume that $R \geq 2$. We claim that there exists an index $s \in \{1, 2, \ldots, n\}$ such that $a_{s,1} = w_{s-1}$ and $c_{s,1} = w_{s+1}$. This means that the transition $b_{s,2} \in \delta(a_{s,1} w_s c_{s,1})$, which is the first rewrite (or delete) transition described by Π_s, matches left and right. From the fact that Π is compatible, we see that $a_{1,1} = \rhd = w_0$. Thus, if $c_{1,1} = w_2$, we can take $s = 1$. If $c_{1,1} \neq w_2$, then compatibility of the pair (Π_1, Π_2) implies that $a_{2,1} = w_1$. Thus, if $c_{2,1} = w_3$, we can take $s = 2$; otherwise, we can repeat this argument for $i = 3, 4, \ldots, n$. As by compatibility, the rewrite and delete transitions of Π_n match right, we see that $c_{n,1} = \lhd = w_{n+1}$. Hence, there exists an index $s \in \{1, 2, \ldots, n\}$ that meets the stated properties.

Now we consider the initial configuration

$$\begin{aligned} &\underline{\rhd w_1 w_2} w_3 \cdots w_{s-2} w_{s-1} w_s w_{s+1} w_{s+2} \cdots w_n \\ &= \underline{\rhd w_1 w_2} w_3 \cdots w_{s-2} a_{s,1} w_s c_{s,1} w_{s+2} \cdots w_n \lhd \end{aligned}$$

of the swift-ORD-automaton M. As M is swift, it can make MVR-steps until its window contains the factor $a_{s,1} w_s c_{s,1}$, and it can then execute the rewrite (or delete)

step $b_{s,2} \in \delta(a_{s,1}w_sc_{s,1})$. Thus, M can execute the cycle

$$\begin{array}{ll} & \underline{\rhd w_1 w_2} w_3 \cdots w_{s-2} a_{s,1} w_s c_{s,1} w_{s+2} \cdots w_n \lhd \\ \vdash_M^c & \underline{\rhd w_1 w_2} w_3 \cdots w_{s-2} a_{s,1} b_{s,2} c_{s,1} w_{s+2} \cdots w_n \lhd \\ = & \underline{\rhd w_1 w_2} w_3 \cdots w_{s-2} w_{s-1} b_{s,2} w_{s+1} w_{s+2} \cdots w_n \lhd . \end{array}$$

If $j_s = 1$ and $b_{s,2} = \lambda$, then we define a new sequence of triples $\Pi' = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_{s-1}, \Pi_{s+1}, \ldots, \Pi_{n+1})$ by deleting the triple Π_s. Otherwise, we define a triple Π'_s by removing the first entries from each of the columns of the triple Π_s, that is,

$$\Pi'_s = [L'_s, W'_s, R'_s] = \begin{bmatrix} a_{s,2} & b_{s,2} & c_{s,2} \\ \cdots & \cdots & \cdots \\ a_{s,j_s} & b_{s,j_s} & c_{s,j_s} \\ & x_s & \end{bmatrix}.$$

In this case we take $\Pi' = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_{s-1}, \Pi'_s, \Pi_{s+1}, \ldots, \Pi_n, \Pi_{n+1})$. In either case, Π' is now a sequence of triples that has all the properties required in (b). As this sequence only contains $R - 1$ rewrite and delete steps, we can conclude by induction that M has an accepting computation that begins with the restarting configuration $\underline{\rhd w_1 w_2} w_3 \cdots w_{s-1} b_{s,2} w_{s+1} \cdots w_n \lhd$, that is,

$$\underline{\rhd w_1 w_2} w_3 \cdots w_{s-1} b_{s,2} w_{s+1} \cdots w_n \lhd \vdash_M^* \textsf{Accept}.$$

Thus, by combining the above cycle with this accepting computation, we obtain an accepting computation of M that begins with the restarting configuration $\underline{\rhd w_1 w_2} w_3 \cdots w_{s-1} w_s w_{s+1} \cdots w_n \lhd$. Hence, $w \in L_C(M)$. □

This completes the proof of Lemma 1. □

Let $M = (\Sigma, \Gamma \cup \{\#\}, \rhd, \lhd, \delta, >)$ be a swift-ORD-automaton as constructed in Proposition 2. To prove that the language $L(M)$ is context-free, we now describe a PDA P that, given a word $w = w_1 w_2 \cdots w_n \in \Gamma^n$ ($n \geq 1$) as input, guesses a corresponding sequence of triples $(\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1})$, checks whether this sequence is compatible, and accepts in the affirmative. Actually, the language $L(P)$ accepted by P will be $L(P) = L_C(M) \cdot \{\lhd\}$, but as the class of context-free languages is closed under right quotients by a single letter, this suffices to prove that $L_C(M)$ is a context-free language. Since $L(M) = L_C(M) \cap \Sigma^*$, it then follows that $L(M)$ is context-free, too.

Accordingly, the input alphabet of P will be $\Sigma_1 = \Sigma \cup \{\lhd\}$. The pushdown alphabet Δ will contain the bottom marker $\rhd$ and all triples $[L, W, R]$ describing sequences of rewrite and delete steps of M. In addition, Δ will contain variants of $[L, W, R]$ in which some of the symbols in L and R have been marked to indicate that the corresponding rewrite and delete steps have already been verified to match left, respectively, to match right. The behavior of P is described by the following algorithm, in which the left sentinel $\rhd$ will be represented by the triple $[\emptyset\ (\rhd)\ \emptyset]$, and the right sentinel $\lhd$ will be represented by the triple $[\emptyset\ (\lhd)\ \emptyset]$.

Algorithm 1 Program for the PDA P.

Input: *A word* $w = w_1 w_2 \cdots w_n \lhd$, *where* $n \geq 0$ *and* $w_1, w_2, \ldots, w_n \in \Gamma$.
Output: **Accept**, *if* $w \in L_C(M)$.
Procedure:
begin (1) stack := $[\emptyset\ (\rhd)\ \emptyset]$; (∗ *The pushdown is initialized* ∗)
(2) *read the first input symbol* x;
(3) **if** $x = \lhd$ **then begin if** $\lambda \in L(M)$ **then Accept end**
else
(4) **begin**
(5) *guess a triple* $\Pi = [L, W, R] = \begin{bmatrix} \rhd & x & c_1 \\ \rhd & b_2 & c_2 \\ \rhd & \cdots & \cdots \\ \rhd & b_j & c_j \\ & b_{j+1} & \end{bmatrix}$ *such that* $b_2 \in \delta(\rhd x c_1)$ *and* $b_{i+1} \in \delta(\rhd b_i c_i)$, $i = 1, 2, \ldots, j$;
(6) *mark all symbols* $\rhd$ *in the left column of* Π;
(∗*All rewrite and delete transitions in* Π *match left* ∗)
(7) stack := push(stack, Π); (∗ *Push* Π *onto the pushdown* ∗)
(8) **repeat** *read the next input symbol* y;
(9) **begin if** $y = \lhd$ **then**
begin *choose* $\Pi = [\emptyset\ (\lhd)\ \emptyset]$; $j = 0$; $b_1 = \lhd$ **end**
(10) **else** *guess a triple* $\Pi = [L, W, R] = \begin{bmatrix} a_1 & y & c_1 \\ a_2 & b_2 & c_2 \\ \cdots & \cdots & \cdots \\ a_j & b_j & c_j \\ & b_{j+1} & \end{bmatrix}$ *such that* $b_2 \in \delta(a_1 y c_1)$ *and* $b_{i+1} \in \delta(a_i b_i c_i)$, $i = 1, 2, \ldots, j$;
(11) *let* $\Pi' = [L', W', R'] = \begin{bmatrix} a'_1 & b'_1 & c'_1 \\ a'_2 & b'_2 & c'_2 \\ \cdots & \cdots & \cdots \\ a'_r & b'_r & c'_r \\ & b'_{r+1} & \end{bmatrix} =$ top(stack);
(∗ Π' *is the topmost triple on the pushdown* ∗)
Loop:
(12) *ignoring already marked contexts,*
left mark all the rewrites of Π *that match left with* Π' *and*
(13) *right mark all the rewrites of* Π' *that match right with* Π;
(∗ *As many of the still right unmarked rewrite* (*and delete*) *transitions of* Π' *as possible and as many of the still left unmarked rewrite* (*and delete*) *transitions of* Π *as possible are put into a linear order such that the respective right* (*left*) *contexts of the transitions of* Π' (Π) *coincide with the actual letters in* Π (Π'). *In* Π' (Π) *the right* (*left*) *contexts of the corresponding transitions are marked.* ∗)
(14) **if** $b'_{r+1} \neq \lambda \neq b_{j+1}$ **then**
(15) **begin if** *all rewrites of* Π *are left marked*

```
                    and all rewrites of Π′ are right marked then
                    stack := push(stack, Π) (* Push Π onto the pushdown *)
                    else Halt (* The pair (Π′, Π) is not compatible *)
                end;
(16) if b′_{r+1} ≠ λ = b_{j+1} then
  (17) begin if all rewrites of Π are left marked then
                    stack := push(stack, Π) (* Push Π onto the pushdown *)
                    else Halt (* The pair (Π′, Π) is not compatible *)
                end;
(18) if b′_{r+1} = λ ≠ b_{j+1} then
  (19) begin if all rewrites of Π′ are right marked then
                    begin stack := pop(stack);
                      (* Π′ is popped from the pushdown *)
                      Π′ := top(stack);
                      (* Π′ is the topmost triple on the pushdown *)
                      goto Loop
                    end
                    else Halt (* The pair (Π′, Π) is not compatible *)
                end;
(20) if b′_{r+1} = λ = b_{j+1} then
  (21) begin if all rewrites of Π′ are right marked then
                    begin stack := pop(stack);
                      (* Π′ is popped from the pushdown *)
                      Π′ := top(stack);
                      (* Π′ is the topmost triple on the pushdown *)
                      goto Loop
                    end
    (22) else if all rewrites of Π are left marked then
                      stack := push(stack, Π);
                      (* Push Π onto the pushdown *)
                      else Halt (* The pair (Π′, Π) is not compatible *)
                end
  end (* of the repeat-loop *)
(23) until y = ◁; (* The final input symbol ◁ has been read *)
(24) if stack contains the sequence [∅ (▷) ∅]Π′[∅ (◁) ∅] and
  all rewrites of Π′ are right marked and b′_{r+1} = #
  then Accept and Halt
end
end.
```

To illustrate the way in which the PDA in Algorithm 1 works, we return to Example 2.

Example 2 (cont.) The computation of the PDA P for simulating the swift-ORD-automaton M is described in the table in Fig. 1. Here the input processed is $w = a_1a_2a_3a_4a_5\triangleleft$, and the table contains the various steps of the computation of the PDA

Step No.	Stack Contents	Input Read	Triple Guessed
(1.1)	$\left[\emptyset\ (\triangleright)\ \emptyset\right]$	a_1	$\begin{bmatrix} \triangleright & a_1 & b_3 \\ \triangleright & b_1 & \# \\ & \lambda & \end{bmatrix}$
(1.2)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \blacktriangleright & a_1 & b_3 \\ \blacktriangleright & b_1 & \# \\ & \lambda & \end{bmatrix}$	-	-
(2.1)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \blacktriangleright & a_1 & b_3 \\ \blacktriangleright & b_1 & \# \\ & \lambda & \end{bmatrix}$	a_2	$\begin{bmatrix} a_1 & a_2 & b_3 \\ a_1 & b_2 & b_3 \\ & \lambda & \end{bmatrix}$
(2.2)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \blacktriangleright & a_1 & b_3 \\ \blacktriangleright & b_1 & \# \\ & \lambda & \end{bmatrix} \begin{bmatrix} \mathbf{a}_1 & a_2 & b_3 \\ \mathbf{a}_1 & b_2 & b_3 \\ & \lambda & \end{bmatrix}$	-	-
(3.1)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \blacktriangleright & a_1 & b_3 \\ \blacktriangleright & b_1 & \# \\ & \lambda & \end{bmatrix} \begin{bmatrix} \mathbf{a}_1 & a_2 & b_3 \\ \mathbf{a}_1 & b_2 & b_3 \\ & \lambda & \end{bmatrix}$	a_3	$\begin{bmatrix} a_2 & a_3 & a_4 \\ b_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix}$
(3.2)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \blacktriangleright & a_1 & b_3 \\ \blacktriangleright & b_1 & \# \\ & \lambda & \end{bmatrix}$	a_3	$\begin{bmatrix} \mathbf{a}_2 & a_3 & a_4 \\ b_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix}$
(3.3)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & a_4 \\ \mathbf{b}_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix}$	-	-
(4.1)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & a_4 \\ \mathbf{b}_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix}$	a_4	$\begin{bmatrix} b_3 & a_4 & \triangleleft \\ b_3 & b_4 & \triangleleft \\ & \lambda & \end{bmatrix}$
(4.2)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & \mathbf{a}_4 \\ \mathbf{b}_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix} \begin{bmatrix} \mathbf{b}_3 & a_4 & \triangleleft \\ \mathbf{b}_3 & b_4 & \triangleleft \\ & \lambda & \end{bmatrix}$	-	-
(5.1)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & \mathbf{a}_4 \\ \mathbf{b}_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix} \begin{bmatrix} \mathbf{b}_3 & a_4 & \triangleleft \\ \mathbf{b}_3 & b_4 & \triangleleft \\ & \lambda & \end{bmatrix}$	a_5	$\begin{bmatrix} a_4 & a_5 & \triangleleft \\ & \lambda & \end{bmatrix}$
(5.2)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & \mathbf{a}_4 \\ \mathbf{b}_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix} \begin{bmatrix} \mathbf{b}_3 & a_4 & \triangleleft \\ \mathbf{b}_3 & b_4 & \triangleleft \\ & \lambda & \end{bmatrix} \begin{bmatrix} \mathbf{a}_4 & a_5 & \triangleleft \\ & \lambda & \end{bmatrix}$	-	-
(6.1)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & \mathbf{a}_4 \\ \mathbf{b}_1 & b_3 & \triangleleft \\ & \# & \end{bmatrix} \begin{bmatrix} \mathbf{b}_3 & a_4 & \triangleleft \\ \mathbf{b}_3 & b_4 & \triangleleft \\ & \lambda & \end{bmatrix} \begin{bmatrix} \mathbf{a}_4 & a_5 & \triangleleft \\ & \lambda & \end{bmatrix}$	$\triangleleft$	$\left[\emptyset\ (\triangleleft)\ \emptyset\right]$
(6.2)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & \mathbf{a}_4 \\ \mathbf{b}_1 & b_3 & \blacktriangleleft \\ & \# & \end{bmatrix}$	$\triangleleft$	$\left[\emptyset\ (\triangleleft)\ \emptyset\right]$
(6.3)	$\left[\emptyset\ (\triangleright)\ \emptyset\right] \begin{bmatrix} \mathbf{a}_2 & a_3 & \mathbf{a}_4 \\ \mathbf{b}_1 & b_3 & \blacktriangleleft \\ & \# & \end{bmatrix} \left[\emptyset\ (\triangleleft)\ \emptyset\right]$	-	-

Fig. 1 Simulating the stl-ORD-automaton of Example 2 by a PDA

simulating M for this input. In the table those tape symbols in the left or right column of a triple that have been marked are written in boldface. Each time an input symbol is read, a corresponding triple $\Pi = [L, W, R]$ is guessed, the rewrite steps of which are neither left nor right marked. Then the triple $\Pi' = [L', W', R']$ on the top of the pushdown is compared to Π, some entries in R' and in L are marked, and depending on the situation (see (14) to (22)), Π' is possibly popped from the pushdown, some more triples may be popped, and then Π is pushed onto the pushdown. As the simulation ends with the pushdown containing the sequence

$$\left[\, \emptyset \;\; (\triangleright) \;\; \emptyset \,\right] \begin{bmatrix} \blacktriangleright \; a_1 \;\; \mathbf{b_3} \\ \blacktriangleright \; b_1 \;\; \blacktriangleleft \\ \# \end{bmatrix} \left[\, \emptyset \;\; (\triangleleft) \;\; \emptyset \,\right],$$

the PDA accepts in line (24).

It remains to prove that $L(P) = L(M) \cdot \{\triangleleft\}$. For doing so, we establish the following technical lemma.

Lemma 2 *The PDA P of Algorithm 1 accepts on input $w = w_1 w_2 \cdots w_n \triangleleft$ ($n \geq 1$, $w_1, w_2, \ldots, w_n \in \Gamma$) if and only if there exists a sequence of triples for $w_1 w_2 \cdots w_n$ that satisfies all the properties of Lemma 1 (b).*

Proof Let $w = w_1 w_2 \cdots w_n$, where $n \geq 1$ and $w_1, w_2, \ldots, w_n \in \Gamma$. Furthermore, let $\Pi = (\Pi_0, \Pi_1, \ldots, \Pi_n, \Pi_{n+1})$ be a sequence of triples, where

$$\Pi_i = [L_i, W_i, R_i] = \begin{bmatrix} a_{i,1} & b_{i,1} & c_{i,1} \\ a_{i,2} & b_{i,2} & c_{i,2} \\ \cdots & \cdots & \cdots \\ a_{i,j_i} & b_{i,j_i} & c_{i,j_i} \\ & x_i & \end{bmatrix}$$

such that $b_{i,1} = w_i$ for all $i = 1, 2, \ldots, n$, let $\Pi_0 = [\emptyset, (\triangleright), \emptyset]$, and $\Pi_{n+1} = [\emptyset, (\triangleleft), \emptyset]$. Comparing the various conditions in Definition 4 to the tests in Algorithm 1, we see that

- condition (a) (i) is guaranteed by line (5),
- condition (a) (ii) is guaranteed by the choice of Π_{n+1} and the tests in lines (19) and (21) on triples that are popped from the pushdown and by the test in line (24),
- condition (a) (iii) corresponds to lines (14) and (15),
- condition (a) (iv) corresponds to lines (16) and (17),
- condition (a) (v) corresponds to lines (18) and (19),
- condition (a) (vi) corresponds to lines (20) to (22), and
- condition (b) (1) is checked by the test in line (24).

Finally, the recursion in the definition of compatibility (see condition (b) (3)) is resolved inductively by the PDA P. In fact, for each triple Π_i, all rewrite (and delete) steps are verified to match left before this triple is pushed onto the pushdown (see lines (15), (17), and (22)), and all rewrite and delete steps are verified to match right before this triple is popped from the pushdown (see lines (19) and (21)). It follows

that the PDA P accepts on input $w = w_1 w_2 \cdots w_n \triangleleft$ if and only if the sequence of triples guessed for $w_1 w_2 \cdots w_n$ satisfies all the properties of Lemma 1 (b). This completes the proof of Lemma 2. □

Now we can present the proof of the main result of this section.

Proof (of Theorem 1.) Let $M = (\Sigma, \Gamma, \triangleright, \triangleleft, \delta, >)$ be a swift stl-ORD-automaton. By Proposition 2, we can construct a swift stl-ORD-automaton $M' = (\Sigma, \Gamma \cup \{\#\}, \triangleright, \triangleleft, \delta', >')$ from M such that $\delta'(\triangleright\#\triangleleft) = \mathsf{Accept}$ is the only accept step of M' and $L(M') = L(M) \setminus \{\lambda\}$. Lemma 1 now implies that, for each $n \geq 1$ and each word $w = w_1 w_2 \cdots w_n \in \Gamma^n$, $w \in L_{\mathrm{C}}(M')$ if and only if there exists a compatible sequence of triples

$$\Pi = (\Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_n, \Pi_{n+1}),$$

where $\Pi_0 = [\emptyset, (\triangleright), \emptyset]$, $\Pi_{n+1} = [\emptyset, (\triangleleft), \emptyset]$, and, for all $i = 1, 2, \ldots, n$,

$$\Pi_i = [L_i, W_i, R_i] = \begin{bmatrix} a_{i,1} & b_{i,1} & c_{i,1} \\ a_{i,2} & b_{i,2} & c_{i,2} \\ \cdots & \cdots & \cdots \\ a_{i,j_i} & b_{i,j_i} & c_{i,j_i} \\ & x_i & \end{bmatrix} \text{ such that } b_{i,1} = w_i.$$

Let P be the PDA that is defined by Algorithm 1 for the swift stl-ORD-automaton M'. Then by Lemma 2, P accepts on input $w = w_1 w_2 \cdots w_n \triangleleft$ if and only if there exists a sequence of triples for $w_1 w_2 \cdots w_n$ that satisfies all the properties above. It follows that $L(P) = L_{\mathrm{C}}(M') \cdot \{\triangleleft\}$, which shows that $L_{\mathrm{C}}(M') \cdot \{\triangleleft\}$ is a context-free language. Hence, $L_{\mathrm{C}}(M')$ is a context-free language, and therewith, $L(M)$ is a context-free language, too, as $L(M') = L_{\mathrm{C}}(M') \cap \Sigma^*$. Finally, this implies that $L(M')$ is a context-free language, as $L(M') = L(M) \setminus \{\lambda\}$. This completes the proof of Theorem 1. □

4 All Context-Free Languages are Accepted by Swift Stl-ORD-Automata

Here we establish the converse of Theorem 1, showing that each context-free language is accepted by a swift stl-ORD-automaton.

Theorem 2 *CFL* $\subseteq \mathcal{L}$(swift-ORD).

Let $L \subseteq \Sigma^*$ be a context-free language. Then there exists a context-free grammar $G = (V, \Sigma, S, P)$ in quadratic Greibach normal form for the language $L \setminus \{\lambda\}$ [23],

that is, each production $(B \to r) \in P$ satisfies the restriction that $B \in V$ and $r \in \Sigma \cdot (V^2 \cup V \cup \{\lambda\})$. From this grammar, a PDA $A = (Q, \Sigma, \Delta_A, q, S, \delta_A)$ can be obtained such that L is the language $N(A)$ that is accepted by A with empty pushdown. The PDA A is defined by taking $Q = \{q\}$, $\Delta_A = V$, and $(q, \alpha^R) \in \delta_A(q, a, B)$ iff $(B \to a\alpha) \in P$, where $a \in \Sigma$, $B \in V$, and $\alpha \in (V^2 \cup V \cup \{\lambda\})$ (see, e.g., [8]). Here α^R denotes the *reversal* (or *mirror image*) of the word α. Thus, A has a single state only, it does not execute any λ-transitions, and in each step, it replaces the topmost symbol on its pushdown by a word of length at most two. In our encoding below, the bottom (top) of the pushdown will always be on the left (right).

For proving that the language L is accepted by a swift stl-ORD-automaton we now proceed as follows. First, we present a construction of a swift stl-ORD-automaton M that simulates the PDA A. Essentially, M works as the automaton in Example 1, that is, it chooses the transition of A that is to be applied next, it marks the letters that are to be rewritten, and then it replaces (or deletes) the corresponding letters. Here the simulation of A is performed in a non-length-increasing fashion using an appropriate encoding of the pushdown. After giving the construction we illustrate it through a simple example, and then we prove that $L(M) = L$ through a sequence of four lemmas.

Definition 5 Let (q, x, α) be a configuration of the PDA A, where $x \in \Sigma^*$ is the still unread suffix of the given input and $\alpha \in V^+$ is the contents of the pushdown. This configuration will be encoded as $[b_1][b_2] \cdots [b_m][q, \alpha_2]x$, where $\alpha = b_1 b_2 \cdots b_m \alpha_2$ for some $\alpha_2 \in V^2 \cup V$ and either $m \geq 1$ and $b_1, b_2, \ldots, b_m \in V$ or $m = 0$.

The swift-ORD-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ is now defined as follows:

– The tape alphabet Γ contains the input alphabet Σ, two disjoint copies of Σ, encodings $[x]$ of the nonterminals $x \in V$, encodings $[q, \alpha]$, $[q, \alpha]''$, $[q, \alpha]_a$ $(a \in \Sigma)$ for all $\alpha \in V \cup V^2$, and the symbol $[q, \lambda]$, that is,

$$\begin{aligned}\Gamma = \Sigma \cup \{a', a'' \mid a \in \Sigma\} \cup \{[x] \mid x \in V\} \cup \{[q, \lambda]\} \cup \\ \{[q, \alpha], [q, \alpha]'' \mid \alpha \in V \cup V^2\} \cup \{[q, \alpha]_a \mid \alpha \in V \cup V^2, a \in \Sigma\}.\end{aligned}$$

– The partial ordering $>$ on Γ is defined through

$$\begin{aligned}a > a' > a'' > [q, xy]'' > [q, xy] > [q, xy]_b \\ > [x] > [q, x]'' > [q, x] > [q, x]_c > [q, \lambda],\end{aligned}$$

for all $a, b, c \in \Sigma$ and $x, y \in V$.

– The transition relation δ is defined through the following table, where $a, b \in \Sigma$, $c \in \Sigma \cup \{\triangleleft\}$, $x, y \in V$, $\gamma \in V \cup V^2$, and $X \in \{\triangleright\} \cup \{\,[x] \mid x \in V\,\}$:

(0)	$\delta(UYZ)$	$\ni$ MVR	for all $U \in \Gamma \cup \{\triangleright\}$ and $Y, Z \in \Gamma$,
(1)	$\delta(\triangleright a \triangleleft)$	$=$ Accept	for all $a \in L \cap (\Sigma \cup \{\lambda\})$,
(2)	$\delta(\triangleright ab)$	$\ni$ $[q, \gamma]$,	if $\delta_A(q, a, S) \ni (q, \gamma)$ and $\gamma \neq \lambda$,
(3)	$\delta([q, x]ac)$	$\ni$ a',	if $\delta_A(q, a, x) \neq \emptyset$,
(4)	$\delta([q, yx]ac)$	$\ni$ a',	if $\delta_A(q, a, x) \neq \emptyset$,
(5)	$\delta(X[q, x]a')$	$\ni$ $[q, x]_a$,	if $\delta_A(q, a, x) \neq \emptyset$,
(6)	$\delta(X[q, yx]a')$	$\ni$ $[q, yx]_a$,	if $\delta_A(q, a, x) \neq \emptyset$,
(7)	$\delta([q, x]_a a'c)$	$\ni$ $[q, \gamma]''$,	if $\delta_A(q, a, x) \ni (q, \gamma), \gamma \neq \lambda$,
(8)	$\delta([q, yx]_a a'c)$	$\ni$ $[q, \gamma]''$,	if $\delta_A(q, a, x) \ni (q, \gamma), \gamma \neq \lambda$,
(9)	$\delta(X[q, yx]_a[q, \gamma]'')$	$\ni$ $[y]$,	if $\delta_A(q, a, x) \ni (q, \gamma), \gamma \neq \lambda$,
(10)	$\delta(X[q, x]_a[q, \gamma]'')$	$\ni$ λ,	if $\delta_A(q, a, x) \ni (q, \gamma), \gamma \neq \lambda$,
(11)	$\delta(X[q, \gamma]''c)$	$\ni$ $[q, \gamma]$,	
(12)	$\delta([q, x]_a a'c)$	$\ni$ a'',	if $\delta_A(q, a, x) \ni (q, \lambda)$,
(13)	$\delta([q, yx]_a a'c)$	$\ni$ a'',	if $\delta_A(q, a, x) \ni (q, \lambda)$,
(14)	$\delta(X[q, yx]_a a'')$	$\ni$ $[q, y]$,	if $\delta_A(q, a, x) \ni (q, \lambda)$,
(15)	$\delta([q, y]a''c)$	$\ni$ λ,	
(16)	$\delta([y][q, x]_a a'')$	$\ni$ λ,	if $\delta_A(q, a, x) \ni (q, \lambda)$,
(17)	$\delta(X[y]a'')$	$\ni$ $[q, y]$,	
(18)	$\delta(\triangleright[q, x]_a a'')$	$\ni$ $[q, \lambda]$,	if $\delta_A(q, a, x) \ni (q, \lambda)$,
(19)	$\delta([q, \lambda]a''c)$	$\ni$ λ,	
(20)	$\delta(\triangleright[q, \lambda]\triangleleft)$	$=$ Accept.	

In order to illustrate this construction, we consider a simple example.

Example 3 Let $A = (\{q\}, \{a, b\}, \{S, B, C\}, q, S, \delta_A)$, where δ_A only contains the following transitions:

$$\begin{aligned} \delta_A(q, a, S) &= \{(q, BC), (q, B)\}, \\ \delta_A(q, b, B) &= \{(q, \lambda)\}, \\ \delta_A(q, a, C) &= \{(q, BC), (q, B)\}. \end{aligned}$$

Then $N(A)$ is the language $\{\,a^n b^n \mid n \geq 1\,\}$, and, for example, A can execute the following accepting computation:

$$\begin{aligned} (q, aaabbb, S) &\vdash_A (q, aabbb, BC) \vdash_A (q, abbb, BBC) \vdash_A (q, bbb, BBB) \\ &\vdash_A (q, bb, BB) \vdash_A (q, b, B) \vdash_A (q, \lambda, \lambda). \end{aligned}$$

This computation is now simulated by the corresponding swift-ORD-automaton M as follows, where the indices refer to the lines in the definition of the transition relation

δ of M:

$$
\begin{array}{lll}
\underline{\rhd aa}abbb\lhd & \vdash_{(2)} & \underline{\rhd}[q, BC]aabbb\lhd \\
 & \vdash_{(4)} & \underline{\rhd[q, BC]a'}abbb\lhd \\
 & \vdash_{(0)} & \underline{\rhd[q, BC]_a a'}abbb\lhd \\
 & \vdash_{(9)} & \underline{\rhd[B][q, BC]''}abbb\lhd \\
 & \vdash_{(11)} & \underline{\rhd[B][q, BC]}abbb\lhd \\
 & \vdash_{(4)} & \underline{\rhd[B][q, BC]a'}bbb\lhd \\
 & \vdash_{(6)} & \underline{\rhd[B][q, BC]_a a'}bbb\lhd \\
 & \vdash_{(8)} & \underline{\rhd[B][q, BC]_a[q, B]''}bbb\lhd \\
 & \vdash_{(9)} & \underline{\rhd[B][B][q, B]''}bbb\lhd \\
 & \vdash_{(11)} & \underline{\rhd[B][B][q, B]}bbb\lhd \\
 & \vdash_{(3)} & \underline{\rhd[B][B][q, B]b'}bb\lhd \\
 & \vdash_{(5)} & \underline{\rhd[B][B][q, B]_b b'}bb\lhd \\
 & \vdash_{(12)} & \underline{\rhd[B][B][q, B]_b b''}bb\lhd \\
 & \vdash_{(16)} & \underline{\rhd[B][B]b''}bb\lhd \\
 & \vdash_{(17)} & \underline{\rhd[B][q, B]b''}bb\lhd \\
 & \vdash_{(15)} & \underline{\rhd[B][q, B]}bb\lhd \\
 & \vdash_{(3)} & \underline{\rhd[B][q, B]b'}b\lhd \\
 & \vdash_{(5)} & \underline{\rhd[B][q, B]_b b'}b\lhd \\
 & \vdash_{(12)} & \underline{\rhd[B][q, B]_b b''}b\lhd \\
 & \vdash_{(16)} & \underline{\rhd[B]b''}b\lhd \\
 & \vdash_{(0)} & \rhd\underline{[q, B]b''}b\lhd \\
 & \vdash_{(0)} & \rhd\underline{[q, B]}b\lhd \\
 & \vdash_{(5)} & \underline{\rhd[q, B]_b b'}\lhd \\
 & \vdash_{(12)} & \underline{\rhd[q, B]_b b''}\lhd \\
 & \vdash_{(0)} & \rhd\underline{[q, \lambda]b''}\lhd \\
 & \vdash_{(20)} & \text{Accept.}
\end{array}
\qquad
\begin{array}{lll}
 & \vdash_{(0)} & \rhd[q, BC]aabbb\lhd \\
 & \vdash_{(6)} & \underline{\rhd[q, BC]_a a'}abbb\lhd \\
 & \vdash_{(8)} & \underline{\rhd[q, BC]_a[q, BC]''}abbb\lhd \\
 & \vdash_{(0)} & \rhd\underline{[B][q, BC]''}abbb\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[q, BC]}abbb\lhd \\
 & \vdash_{(0)} & \rhd[B]\underline{[q, BC]a'}bbb\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[q, BC]_a a'}bbb\lhd \\
 & \vdash_{(0)} & \rhd\underline{[B][q, BC]_a[q, B]''}bbb\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[B][q, B]''}bbb\lhd \\
 & \vdash^3_{(0)} & \rhd[B]\underline{[B][q, B]}bbb\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[B][q, B]b'}bb\lhd \\
 & \vdash^3_{(0)} & \rhd[B]\underline{[B][q, B]_b b'}bb\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[B][q, B]_b b''}bb\lhd \\
 & \vdash_{(0)} & \rhd\underline{[B][B]b''}bb\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[q, B]b''}bb\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[q, B]}bb\lhd \\
 & \vdash_{(0)} & \rhd\underline{[B][q, B]b'}b\lhd \\
 & \vdash^2_{(0)} & \rhd[B]\underline{[q, B]_b b'}b\lhd \\
 & \vdash_{(0)} & \rhd\underline{[B][q, B]_b b''}b\lhd \\
 & \vdash_{(17)} & \rhd\underline{[q, B]b''}b\lhd \\
 & \vdash_{(15)} & \rhd\underline{[q, B]}b\lhd \\
 & \vdash_{(3)} & \rhd\underline{[q, B]b'}\lhd \\
 & \vdash_{(0)} & \rhd\underline{[q, B]_b b'}\lhd \\
 & \vdash_{(18)} & \underline{\rhd[q, \lambda]b''}\lhd \\
 & \vdash_{(19)} & \underline{\rhd[q, \lambda]}\lhd
\end{array}
$$

We must prove that $L(M) = N(A)$. Let

$$C = (q, w, x_1x_2 \ldots x_{m-2}x_{m-1}x_m)$$

be a configuration of the PDA A, where $w \in \Sigma^+$ and $x_1, x_2, \ldots, x_m \in V$ for some $m \geq 2$. Then the restarting configurations

$$C_1 = \underline{\rhd[x_1][x_2]}[x_3] \cdots [x_{m-2}][x_{m-1}][q, x_m]w\lhd$$

and

$$C_2 = \underline{\rhd[x_1][x_2]}[x_3] \cdots [x_{m-2}][q, x_{m-1}x_m]w\lhd$$

of M are *representations* of the configuration C of A. Conversely, if

$$C_3 = \underline{\rhd[x_1][x_2]}[x_3] \cdots [x_m][q, \alpha]w\lhd$$

is a restarting configuration of M for some $x_1, x_2, \ldots, x_m \in V$, $\alpha \in V \cup V^2$, and $w \in \Sigma^+$, then $C' = (q, w, x_1x_2 \cdots x_m\alpha)$ is the corresponding configuration of A. Thus, for each configuration of the PDA A, in which the height of the pushdown is at least two, we have two restarting configurations of M that represent it.

In order to prove that $N(A)$ is contained in the language $L(M)$, we first establish the following technical result.

Lemma 3 *Let $C = (q, w_1, \alpha_1) \vdash_A (q, w_2, \alpha_2) = C'$, where $w_1, w_2 \in \Sigma^+$ and $\alpha_1, \alpha_2 \in V^+$, and let C_1 be a restarting configuration of M that represents the configuration C. Then $C_1 \vdash_M^{c^*} C_2$ for some restarting configuration C_2 that represents the configuration C'.*

Proof As $C = (q, w_1, \alpha_1) \vdash_A (q, w_2, \alpha_2) = C'$, we have $w_1 = aw_2$ for some $a \in \Sigma$, $\alpha_1 = x_1x_2 \cdots x_mx$, and $\alpha_2 = x_1x_2 \cdots x_m\gamma$ for some $m \geq 0$, $x_1, x_2 \ldots, x_m, x \in V$, and $\gamma \in V \cup V^2 \cup \{\lambda\}$ such that $(q, \gamma) \in \delta_A(q, a, x)$ and $x_1x_2 \cdots x_m\gamma \neq \lambda$. The restarting configuration C_1 represents the configuration C, and so $C_1 = \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_mx]aw_2\lhd$ or $C_1 = \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][x_m][q, x]aw_2\lhd$.

We first assume that $\gamma \neq \lambda$. Then M can proceed as follows starting from the configuration C_1:

$$
\begin{aligned}
C_1 &= \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_mx]aw_2\lhd \\
&\vdash_M^{c^*} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_mx]_a a'w_2\lhd && \text{(by lines (4) and (6))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_mx]_a[q, \gamma]''w_2\lhd && \text{(by line (8))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][x_m][q, \gamma]''w_2\lhd && \text{(by line (9))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][x_m][q, \gamma]w_2\lhd && \text{(by line (11))}
\end{aligned}
$$

or

$$
\begin{aligned}
C_1 &= \underline{\rhd[x_1][x_2]} \cdots [x_m][q, x]aw_2\lhd \\
&\vdash_M^{c^*} \underline{\rhd[x_1][x_2]} \cdots [x_m][q, x]_a a'w_2\lhd && \text{(by lines (3) and (5))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_m][q, x]_a[q, \gamma]''w_2\lhd && \text{(by line (7))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_m][q, \gamma]''w_2\lhd && \text{(by line (10))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_m][q, \gamma]w_2\lhd && \text{(by line (11)),}
\end{aligned}
$$

where, in either case, the restarting configuration $\underline{\rhd[x_1][x_2]} \cdots [x_m][q, \gamma]w_2\lhd$ represents the configuration $C' = (q, w_2, \alpha_2)$ of A.

If $\gamma = \lambda$, then M can proceed as follows starting from the configuration C_1:

$$
\begin{aligned}
C_1 &= \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_mx]aw_2\lhd \\
&\vdash_M^{c^*} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_mx]_a a'w_2\lhd && \text{(by lines (4) and (6))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_mx]_a a''w_2\lhd && \text{(by line (13))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_m]a''w_2\lhd && \text{(by line (14))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_m]w_2\lhd && \text{(by line (15)),}
\end{aligned}
$$

or

$$
\begin{aligned}
C_1 &= \underline{\rhd[x_1][x_2]} \cdots [x_m][q, x]aw_2\lhd \\
&\vdash_M^{c^*} \underline{\rhd[x_1][x_2]} \cdots [x_m][q, x]_a a'w_2\lhd && \text{(by lines (3) and (5))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_m][q, x]_a a''w_2\lhd && \text{(by line (12))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [x_m]a''w_2\lhd && \text{(by line (16))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [q, x_m]a''w_2\lhd && \text{(by line (17))} \\
&\vdash_M^{c} \underline{\rhd[x_1][x_2]} \cdots [q, x_m]w_2\lhd && \text{(by line (15)),}
\end{aligned}
$$

where the restarting configuration $\underline{\rhd[x_1][x_2]} \cdots [x_{m-1}][q, x_m]w_2\lhd$ represents the configuration $C' = (q, w_2, \alpha_2)$ of A. This proves Lemma 3. □

Now we are prepared to prove the following inclusion.

Lemma 4 $N(A) \subseteq L(M)$.

Proof Let $w \in N(A)$. If $|w| \leq 1$, then $w \in L \cap (\Sigma \cup \{\lambda\})$, and hence, M can accept on input w by line (1). So let us assume that $w = a_1 a_2 \cdots a_n$ for some $n \geq 2$ and $a_1, a_2 \ldots, a_n \in \Sigma$. As $w \in N(A)$, A has an accepting computation for input w:

$$\begin{array}{rll} (q, w, S) & = & (q, a_1 a_2 a_3 \cdots a_n, S) \\ & \vdash_A & (q, a_2 a_3 \cdots a_n, \gamma_1) \quad \text{(using } (q, \gamma_1) \in \delta_A(q, a_1, S) \text{ for some } \gamma_1 \neq \lambda) \\ & \vdash_A^* & (q, a_n, \gamma_n) \quad \text{(for some } \gamma_n \in V) \\ & \vdash_A & (q, \lambda, \lambda) \quad \text{(as } A \text{ accepts with empty pushdown).} \end{array}$$

Now M has a computation of the following form:

$$\begin{array}{ll} \underline{\rhd a_1 a_2} a_3 \cdots a_n \lhd & \\ \vdash_M^c \underline{\rhd [q, \gamma_1] a_2} a_3 \cdots a_n \lhd & \text{(by line (2))} \\ \vdash_M^{c^*} \underline{\rhd [q, \gamma_n] a_n} \lhd & \text{(by using Lemma 3 repeatedly)} \\ \vdash_M^{c^*} \underline{\rhd [q, \gamma_n]_{a_n} a_n''} \lhd & \text{(by lines (3), (5), and (12))} \\ \vdash_M^c \underline{\rhd [q, \lambda] a_n''} \lhd & \text{(by line (18))} \\ \vdash_M^c \underline{\rhd [q, \lambda]} \lhd & \text{(by line (19))} \\ \vdash_M \underline{\mathsf{Accept}} & \text{(by line (20)),} \end{array}$$

as the restarting configuration $\underline{\rhd [q, \gamma_1] a_2} a_3 \cdots a_n \lhd$ represents the configuration $(q, a_2 a_3 \cdots a_n, \gamma_1)$ of A and the restarting configuration $\underline{\rhd [q, \gamma_n] a_n} \lhd$ represents the configuration (q, a_n, γ_n) of A. Thus, $N(A) \subseteq L(M)$ follows. □

It remains to prove the converse inclusion. We first derive the following technical result.

Lemma 5 *Let C_1 be a restarting configuration of M such that $C_1 \vdash_M^* \mathsf{Accept}$. If C_1 represents a configuration (q, w, α) of the PDA A for some $w \in \Sigma^*$ of length $|w| \geq 2$ and $\alpha \in V^+$, then there exists a restarting configuration C_2 of M such that $C_1 \vdash_M^{c^+} C_2 \vdash_M^* \mathsf{Accept}$ and C_2 represents an immediate successor configuration of (q, w, α).*

Proof Let $w = a_1 a_2 \cdots a_n$ and $\alpha = x_1 x_2 \cdots x_m$. Then

$$C_1 = \underline{\rhd [x_1][x_2]}[x_3] \cdots [x_{m-2}][q, x_{m-1} x_m] a_1 a_2 \cdots a_n \lhd$$

or

$$C_1 = \underline{\rhd [x_1][x_2]}[x_3] \cdots [x_{m-1}][q, x_m] a_1 a_2 \cdots a_n \lhd .$$

From the form of the transition relation δ, we can conclude that the accepting computation of M that starts with the configuration C_1 begins with a sequence of cycles that finally reaches the restarting configuration $C_f = \rhd [q, \lambda] \lhd$, in which M simply accepts. Thus, during this sequence of cycles, one of the symbols on the tape is rewritten into the symbol $[q, \lambda]$, while all other symbols are eventually deleted. We now analyze the possible rewrite and delete steps that M can apply starting from C_1.

The only rewrite transition that is applicable to the tape contents of the configuration C_1 is from line (3) or (4), and after that a rewrite by line (5) or (6) must follow. Hence, $C_1 \vdash_M^{c^2} D_1$, where

$$D_1 = \underline{\rhd[x_1][x_2]}[x_3]\cdots[x_{m-2}][q, x_{m-1}x_m]_{a_1}a_1'a_2\cdots a_n\lhd$$

or

$$D_1 = \underline{\rhd[x_1][x_2]}[x_3]\cdots[x_{m-1}][q, x_m]_{a_1}a_1'a_2\cdots a_n \lhd .$$

As this sequence of two cycles is an initial part of an accepting computation of M, it follows that $\delta_A(q, a_1, x_m) \neq \emptyset$, and that the next rewrite step is executed based on a transition $(q, \gamma) \in \delta_A(q, a_1, x_m)$ (see lines (7), (8), (12), and (13)).

There are two cases:

(i) If line (7) or (8) is used next, then $\gamma \neq \lambda$, and the subsequent rewrite is executed based on line (9) or (10), that is, $D_1 \vdash_M^{c^2} D_2$, where

$$D_2 = \underline{\rhd[x_1][x_2]}[x_3]\cdots[x_{m-2}][x_{m-1}][q, \gamma]''a_2\cdots a_n\lhd,$$

and then

$$D_2 \vdash_M^c C_2 = \underline{\rhd[x_1][x_2]}[x_3]\cdots[x_{m-2}][x_{m-1}][q, \gamma]a_2\cdots a_n\lhd$$

by using line (11). Now the restarting configuration C_2 represents the configuration $(q, a_2a_3\cdots a_n, x_1x_2\cdots x_{m-1}\gamma)$ of A, and

$$(q, w, \alpha) = (q, a_1a_2\cdots a_n, x_1\cdots x_{m-1}x_m) \vdash_A (q, a_2a_3\cdots a_n, x_1\cdots x_{m-1}\gamma).$$

(ii) If line (12) or (13) is used next, then $\gamma = \lambda$, and the subsequent rewrite is executed by applying line (14) or (16), that is, $D_1 \vdash_M^{c^2} D_2$, where

$$D_2 = \underline{\rhd[x_1][x_2]}[x_3]\cdots[x_{m-2}][q, x_{m-1}]a_1''a_2\cdots a_n\lhd$$

or

$$D_2 = \underline{\rhd[x_1][x_2]}[x_3]\cdots[x_{m-1}]a_1''a_2\cdots a_n \lhd .$$

In the latter case, another cycle follows in which a rewrite step is executed by line (17) that yields the configuration

$$\underline{\rhd[x_1][x_2]}[x_3]\cdots[q, x_{m-1}]a_1''a_2\cdots a_n \lhd .$$

Finally, line (15) is used to delete the symbol a_1'', which yields the configuration

$$C_2 = \underline{\rhd[x_1][x_2]}[x_3]\cdots[x_{m-2}][q, x_{m-1}]a_2\cdots a_n \lhd .$$

In this case, the restarting configuration C_2 represents the configuration $(q, a_2a_3\cdots a_n, x_1x_2\cdots x_{m-1})$ of A, and

$$(q, w, \alpha) = (q, a_1a_2\cdots a_n, x_1\cdots x_{m-1}x_m) \vdash_A (q, a_2a_3\cdots a_n, x_1\cdots x_{m-1}).$$

This completes the proof of Lemma 5. □

Based on Lemma 5, we can now prove the following inclusion.

Lemma 6 $L(M) \subseteq N(A)$.

Proof Let $w \in L(M)$, that is, $\rhd w \lhd \vdash_M^* \mathsf{Accept}$. If $|w| \leq 1$, then w is accepted by line (1). This, however, means that $w \in L = N(A)$.

We now consider the case that $|w| \geq 2$. The only rewrite transitions that can be applied to the tape contents $\rhd a_1 a_2 \cdots a_n \lhd$ are those of line (2). Hence, the accepting computation of M for w has the following structure:

$$\underline{\rhd a_1 a_2} a_3 \cdots a_n \lhd \vdash_M^c C_1 = \underline{\rhd [q, \gamma_1] a_2} a_3 \cdots a_n \lhd \vdash_M^{c^+} \underline{\rhd [q, \lambda] \lhd} \vdash_M \mathsf{Accept}$$

for a pair $(q, \gamma_1) \in \delta_A(q, a_1, S)$ such that $\gamma_1 \neq \lambda$. Thus, on input w, the PDA A can execute the transition

$$(q, w, S) = (q, a_1 a_2 a_3 \cdots a_n, S) \vdash_A (q, a_2 a_3 \cdots a_n, \gamma_1),$$

and C_1 represents the configuration $(q, a_2 a_3 \cdots a_n, \gamma_1)$ of A. If $|a_2 a_3 \cdots a_n| = n - 1 \geq 2$, then we can apply Lemma 5, which implies that there exists a restarting configuration C_2 of M such that $C_1 \vdash_M^{c^+} C_2 \vdash_M^* \mathsf{Accept}$ and C_2 represents an immediate successor configuration $(q, a_3 a_4 \cdots a_n, \gamma_2)$ of the configuration $(q, a_2 a_3 \cdots a_n, \gamma_1)$ of A. Proceeding inductively, we obtain a sequence of restarting configurations $C_3, C_4, \ldots, C_{n-1}$ such that

$$C_2 \vdash_M^{c^+} C_3 \vdash_M^{c^+} \cdots \vdash_M^{c^+} C_{n-1} \vdash_M^* \mathsf{Accept},$$

the PDA A can execute the sequence of transitions

$$(q, a_3 a_4 \cdots a_n, \gamma_2) \vdash_A (q, a_4 a_5 \cdots a_n, \gamma_3) \vdash_A \cdots \vdash_A (q, a_n, \gamma_{n-1}),$$

and the restarting configuration C_i of M represents the configuration $(q, a_{i+1} a_{i+2} \cdots a_n, \gamma_i)$ of A for all $i = 2, 3, \ldots, n-1$.

As C_{n-1} represents the configuration (q, a_n, γ_{n-1}) of A, we have $C_{n-1} = \rhd W[q, \alpha] a_n \lhd$ for some $W \in \{[x] \mid x \in V\}^*$ and $\alpha \in V \cup V^2$ such that $\gamma_{n-1} = \varphi(W[q, \alpha])$, where $\varphi : (\{[x] \mid x \in V\} \cup \{[q, \alpha] \mid \alpha \in V \cup V^2\})^* \to V^*$ denotes the morphism that is defined by $[x] \mapsto x$ and $[q, \alpha] \mapsto \alpha$. From δ we see that the accepting computation of M starting from C_{n-1} has the form

$$C_{n-1} = \underline{\rhd W[q, \alpha]} a_n \lhd \vdash_M^{c^+} \underline{\rhd [q, \lambda] \lhd} \vdash_M \mathsf{Accept}.$$

Furthermore, as there is only a single input letter left in C_{n-1}, we can conclude that $|\gamma_{n-1}| = 1$, that is, $W = \lambda$ and $\alpha = \gamma_{n-1} \in V$. From line (18), which is the only one that produces an occurrence of the symbol $[q, \lambda]$, we see that $(q, \lambda) \in \delta_A(q, a_n, \gamma_{n-1})$. Thus, A can execute the accepting computation

$$\begin{aligned} (q, w, S) &= (q, a_1 a_2 \cdots a_n, S) \vdash_A (q, a_2 a_3 \cdots a_n, \gamma_1) \vdash_A (q, a_3 a_4 \cdots a_n, \gamma_2) \\ &\vdash_A^* (q, a_n, \gamma_{n-1}) \vdash_A (q, \lambda, \lambda), \end{aligned}$$

which proves that $w \in N(A) = L$. This completes the proof of Lemma 6. □

Together Lemmas 4 and 6 show that $L(M) = N(A) = L$, which proves Theorem 2. From Theorems 1 and 2 we obtain the following characterization.

Corollary 1 $\mathcal{L}(\text{swift-ORD}) = \text{CFL}$.

5 Descriptional Complexity

Here we take a look at the descriptional complexity of stl-ORD-automata, relating it to the descriptional complexity of stl-ORWW-automata. As both these types of automata are stateless, we cannot possibly take the number of states as a measure for their descriptional complexity, as is done for finite-state acceptors (see, e.g., [7]). Instead, for both these types of automata, we take the size of the tape alphabet as complexity measure. This is reasonable, as the number of transitions and the size of the description of an automaton of one of these types are polynomially related to this measure. Here we have the following result, showing that for some regular languages, swift-ORD-automata yield a much more concise description than corresponding stl-ORWW-automata.

Theorem 3 *The trade-off between swift-ORD-automata and stl-ORWW-automata is non-recursive.*

Thus, the size increase that occurs when turning a swift-ORD-automaton that accepts a regular language into a stl-ORWW-automaton for the same language cannot be bounded from above by any recursive function.

Proof According to an old result by Meyer and Fischer [14], the trade-off for turning a context-free grammar into an equivalent NFA is non-recursive. Now let G be a given context-free grammar that generates a regular language. From G we can construct an equivalent grammar G_1 that is in quadratic Greibach normal form. The grammar G_1 is in general much larger than the grammar G, but the size increase is bounded from above by an exponential function (see [23]). Thus, $\text{size}(G_1) \leq c^{\text{size}(G)}$, where we use $\text{size}(G)$ (or $\text{size}(A)$) to denote the size of a grammar G or an automaton A. From the latter grammar, we immediately obtain a PDA A that accepts the language $L(G) = L(G_1)$ by empty pushdown. Then $\text{size}(A) = \text{size}(G_1) \leq c^{\text{size}(G)}$. Following the construction given in the proof of Theorem 2, we obtain a swift-ORD-automaton M for the language $L(G)$. As is easily seen from this construction, the number of letters n in the tape alphabet of M is bounded from above by an exponential function in the size of the PDA A. Thus, $\text{size}(M) \leq c^{\text{size}(A)} \leq c^{c^{\text{size}(G)}}$. Now assume that f is a recursive function such that, for each swift-ORD-automaton P, there exists an equivalent stl-ORWW-automaton P' such that the number of letters in the tape alphabet of P' is bounded from above by the value $f(\text{size}(P))$. Then there exists a stl-ORWW-automaton M' for the language $L(M) = L(G)$ such that $\text{size}(M') \leq f(\text{size}(M)) \leq f(c^{c^{\text{size}(G)}})$. Finally, it is known from [10] that from M' we can construct an NFA B for the language $L(G)$ that is of size $2^{O(\text{size}(M'))}$. Thus, we see that $\text{size}(B) \leq 2^{O(f(c^{c^{\text{size}(G)}}))}$, which is a recursive bound for the conversion

of the context-free grammar G into the equivalent NFA B. As this contradicts the aforementioned result of Meyer and Fischer, there is no recursive bound for the conversion of a swift-ORD-automaton into an equivalent stl-ORWW-automaton, which completes the proof. □

As the swift-ORD-automaton is just a restricted variant of the stl-ORD-automaton, Theorem 3 also holds for stl-ORD-automata in general.

6 Limited Context Restarting Automata

After a restart a swift-ORD-automaton can move its window to any position before executing a restart-rewrite, a restart-delete, or an accept step. Accordingly, this automaton can be seen as a type of rewriting system on its tape alphabet. And indeed, restarting automata of this form have been studied before.

In [4], the so-called *clearing restarting automaton* was introduced, which deletes symbols depending only on the context of a fixed size around the factor to be deleted. Not surprisingly, clearing restarting automata are quite limited in their expressive power. They accept all regular languages and some languages that are not context-free, but they do not even accept all context-free languages. Accordingly, they were extended to the so-called *Δ-clearing restarting automata* that can use a marker symbol Δ in their rewrite transitions. These automata only accept languages that are growing context-sensitive [21], but they accept all context-free languages [5]. However, it is still open whether or not there is a growing context-sensitive language that is not accepted by any Δ-clearing restarting automaton.

In [1], *limited context restarting automata* were defined as an extension of the clearing restarting automaton. Also these automata apply rewrite steps only based on context information, but their rewrite instructions are more general. Following [21], these automata can be defined as follows.

Definition 6 A *limited context restarting automaton* (an lc-R-automaton, for short) M is defined through a triple $M = (\Sigma, \Gamma, I)$, where Σ is an input alphabet, Γ is a working alphabet containing Σ, and I is a finite set of *instructions* of the form $(u \,|\, x \to y \,|\, v)$. Here $x, y \in \Gamma^*$ such that $g(x) > g(y)$ for some weight function $g : \Gamma^* \to \mathbb{N}$, $u \in \{\lambda, \rhd\} \cdot \Gamma^*$, and $v \in \Gamma^* \cdot \{\lambda, \lhd\}$. Again the symbols $\rhd$ and $\lhd$ are used as left and right sentinels, which are not elements of Γ.

The lc-R-automaton $M = (\Sigma, \Gamma, I)$ induces a reduction relation $\vdash_M^c$ on Γ^* as follows: for each $w, z \in \Gamma^*$, $w \vdash_M^c z$, if there exist words $w_1, w_2 \in \Gamma^*$ and an instruction $(u \,|\, x \to y \,|\, v) \in I$ such that $w = w_1 x w_2$, $z = w_1 y w_2$, u is a suffix of $\rhd w_1$, and v is a prefix of $w_2 \lhd$. Thus, the factor x is rewritten into y, if it appears within the context uxv. By $\vdash_M^{c^*}$ we denote the reflexive and transitive closure of $\vdash_M^c$. The language accepted by the lc-R-automaton M is $L(M) = \{\, w \in \Sigma^* \mid w \vdash_M^{c^*} \lambda \,\}$.

An lc-R-automaton M accepts exactly the set of input words which can be reduced to λ. Thus, λ is in $L(M)$ for each lc-R-automaton M. Accordingly, if L is a language that does not contain λ as an element, then L is not accepted by any lc-R-automaton.

In order to overcome this problem, we consider equality of languages only up to the empty word, that is, we say that two languages L and L' on Σ are equal, denoted as $L \doteq L'$, if $L \cap \Sigma^+ = L' \cap \Sigma^+$.

In [21], many different types of limited context restarting automata have been introduced and their expressive power has been studied. Here we are only interested in the following two types. We say that an lc-R-automaton $M = (\Sigma, \Gamma, I)$ is of type

- $\mathcal{R}'_1$, if I only contains instructions of the form $(u \,|\, x \to y \,|\, v)$, where $|y| \leq 1$;
- $\mathcal{R}'_2$, if I only contains instructions of the form $(u \,|\, x \to y \,|\, v)$, where $|y| \leq 1$, $u \in \{\lambda, \rhd\}$, and $v \in \{\lambda, \lhd\}$.

For any $\mathcal{R} \in \{\mathcal{R}'_1, \mathcal{R}'_2\}$, we will refer to lc-R-automata of type $\mathcal{R}$ as lc-R[$\mathcal{R}$]-automata. In [21], the following characterizations were obtained.

Theorem 4 (a) $\mathcal{L}(\text{lc-R}[\mathcal{R}'_1]) = \text{GCSL}$. (b) $\mathcal{L}(\text{lc-R}[\mathcal{R}'_2]) = \text{CFL}$.

A clearing restarting automaton can be described as a limited context restarting automaton $M = (\Sigma, \Sigma, I)$ of type $\mathcal{R}'_1$ such that each instruction $(u \,|\, x \to y \,|\, v) \in I$ satisfies the restrictions $|u| = |v| \geq 1$ and $y = \lambda$. Furthermore, a Δ-clearing restarting automaton can be described as a limited context restarting automaton $M = (\Sigma, \Sigma \cup \{\Delta\}, I)$ of type $\mathcal{R}'_1$ such that each instruction $(u \,|\, x \to y \,|\, v) \in I$ satisfies the restrictions $|u| = |v| \geq 1$ and $y \in \{\lambda, \Delta\}$ [4].

Here we introduce still another type of limited context restarting automata in order to describe swift-ORD-automata. A limited context restarting automaton $M = (\Sigma, \Gamma, I)$ is of *type* $\mathcal{R}_{\text{ORD}}$ if each instruction $(u \,|\, x \to y \,|\, v) \in I$ satisfies the restrictions $|u| = |x| = |v| = 1$ and $|y| \leq 1$. Observe that the weight function associated with M induces a partial ordering $>$ on Γ such that $x > y$ holds for each instruction $(u \,|\, x \to y \,|\, v) \in I$ for which $|y| = 1$. Thus, $\mathcal{R}_{\text{ORD}}$ is obtained from the type $\mathcal{R}'_1$ by restricting the left context, the right context, and the lefthand side of the rewrite instruction to be of length one. Limited context restarting automata of type $\mathcal{R}_{\text{ORD}}$ will be denoted as lc-R[$\mathcal{R}_{\text{ORD}}$]-automata.

Lemma 7 $\mathcal{L}(\text{lc-R}[\mathcal{R}_{\text{ORD}}]) \subseteq \mathcal{L}(\text{swift-ORD})$.

Proof Let $M = (\Sigma, \Gamma, I)$ be an lc-R[$\mathcal{R}_{\text{ORD}}$]-automaton. We define an ordering on Γ by taking $A > B$ if $g(A) > g(B)$, where $g : \Gamma^* \to \mathbb{N}$ is the weight function for M. Now we construct a swift-ORD-automaton $M' = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ by defining the transition relation δ as follows:

$$\begin{array}{lll} (1)\ \delta(abc) \ni \text{MVR} & & \text{for all } a \in \Gamma \cup \{\rhd\} \text{ and } b, c \in \Gamma, \\ (2)\ \delta(abc) \ni y & & \text{if } (a \,|\, b \to y \,|\, c) \in I, \\ (3)\ \delta(\rhd\lhd) = \text{Accept}. & & \end{array}$$

Then $L(M') = \{\, w \in \Sigma^* \mid w \vdash_M^{c^*} \lambda \,\} = L(M)$, which proves the above inclusion. □

Actually, we even have the following result.

Theorem 5 $\mathcal{L}(\text{lc-R}[\mathcal{R}_{\text{ORD}}]) = \mathcal{L}(\text{swift-ORD})$.

Proof Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a swift-ORD. By Proposition 2, we may assume that there is a special non-input symbol $\# \in \Gamma$ such that $\delta(\rhd\#\lhd) = \text{Accept}$ is the only accept step of M. Now we define an lc-R$[\mathcal{R}_{\text{ORD}}]$-automaton $S = (\Sigma, \Gamma, I)$ by taking

$$I = \{ (a \mid b \to d \mid c) \mid d \in \delta(abc) \} \cup \{\rhd \mid \# \to \lambda \mid \lhd\}.$$

Then $L(S) = \{ w \in \Sigma^* \mid w \vdash_S^{c^*} \lambda \} \doteq L(M)$. Thus, we also have $\mathcal{L}(\text{swift-ORD}) \subseteq \mathcal{L}(\text{lc-R}[\mathcal{R}_{\text{ORD}}])$, which, together with Lemma 7, yields the intended equality. □

Hence, we conclude the following from Corollary 1.

Corollary 2 $\mathcal{L}(\text{lc-R}[\mathcal{R}_{\text{ORD}}]) = \text{CFL}$.

Thus, the limited context restarting automata of type $\mathcal{R}_{\text{ORD}}$ are just as expressive as those of type $\mathcal{R}'_2$. However, type $\mathcal{R}_{\text{ORD}}$ is a restricted variant of type $\mathcal{R}'_1$ that is incomparable to type $\mathcal{R}'_2$. Indeed, an automaton of type $\mathcal{R}_{\text{ORD}}$ admits arbitrary left and right contexts of length one, while the only non-empty contexts that an automaton of type $\mathcal{R}'_2$ admits are the sentinels $\rhd$ and $\lhd$. On the other hand, an automaton of type $\mathcal{R}'_2$ may rewrite factors of arbitrary positive length, while an automaton of type $\mathcal{R}_{\text{ORD}}$ can rewrite factors of length one only. Hence, Corollary 2 nicely complements the known results on limited context restarting automata presented in [21].

7 Conclusion

By introducing an additional restart-delete operation, which can just delete a single letter at a time, we have extended the stl-ORWW-automaton to the stl-ORD-automaton. As we have shown, this extension has surprizingly strong consequences. While the stl-ORWW-automaton just accepts the regular languages, we have seen that the stl-ORD-automaton accepts all context-free languages. In fact, by restricting the stl-ORD-automaton to its swift variant, we obtained a new characterization for the class of context-free languages. From the given constructions, we additionally derived the fact that stl-ORD-automata describe some regular languages in a much more succinct way than even stl-ORWW-automata. In fact, the size increase (measured in the cardinality of the underlying tape alphabets) that is required for turning a stl-ORD-automaton into an equivalent stl-ORWW-automaton can in general not be bounded from above by any recursive function.

Furthermore, we noticed that swift-ORD-automata can be interpreted as a new class of limited context restarting automata that is a proper subclass of the limited context restarting automata of type $\mathcal{R}'_1$. This new class is incomparable under inclusion to the limited context restarting automata of type $\mathcal{R}'_2$, although it has exactly the same expressive capacity.

Unfortunately, it still remains open whether stl-ORD-automata that are not swift can accept any languages that are not context-free. The diagram in Fig. 2 presents the inclusion and non-inclusion results between the classes of languages that are

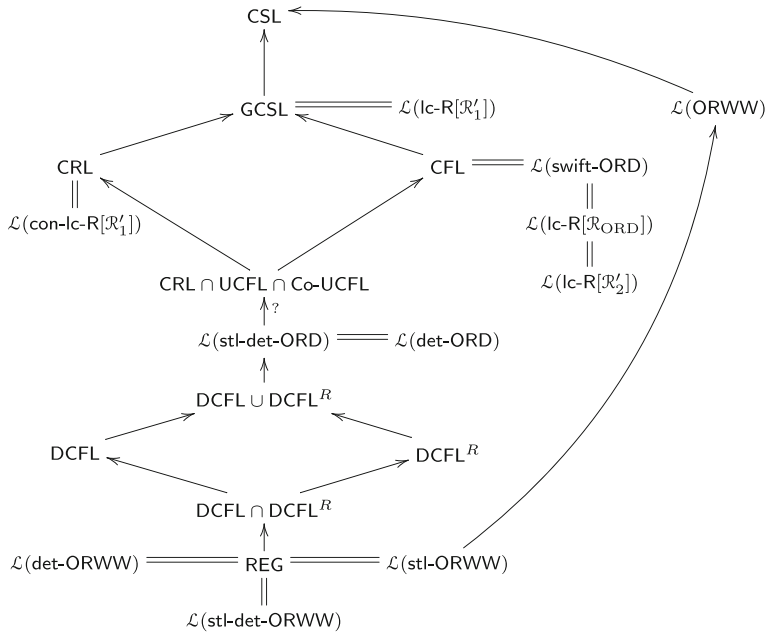

Fig. 2 Hierarchy of language classes that are accepted by the various types of ordered restarting automata. An arrow indicates a proper inclusion, while an arrow with a question mark indicates an inclusion that is not known to be proper. If two classes that are not connected by a sequence of arrows, then they are incomparable under inclusion

accepted by the various types of ordered restarting automata, the classes of the Chomsky hierarchy, and some related classes of limited context restarting automata. Here $\mathsf{DCFL}^R = \{ L^R \mid L \in \mathsf{DCFL} \}$ is the class of languages that are reversals of deterministic context-free languages, UCFL is the class of *unambiguous* context-free languages, and Co-UCFL is the class of languages the complements of which are unambiguous context-free. Finally, $\mathcal{L}(\mathsf{con\text{-}lc\text{-}R}[\mathcal{R}'_1])$ is the class of languages which are accepted by limited context restarting automata that are of the *confluent* version of type $\mathcal{R}'_1$ (see [21]). The results on det-ORWW-automata can be found in [22], those on stl-ORWW- and ORWW-automata are taken from [10, 11], the results on det-ORD-automata are from [18, 19], and those on limited context-restarting automata are from [21].

Acknowledgements The author thanks two anonymous reviewers for their very detailed comments on an earlier version of this paper and their suggestions, which helped greatly to improve the presentation.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Basovník, S., Mráz, F.: Learning limited context restarting automata by genetic algorithms. In: Dassow, J., Truthe, B. (eds.) Theorietag 2011, pp. 1–4. Otto-von-Guericke-Universität, Magdeburg (2011)
2. Book, R., Otto, F.: String-Rewriting Systems. Springer, New York (1993)
3. Buntrock, G., Otto, F.: Growing context-sensitive languages and Church-Rosser languages. Inform. Comput. **141**, 1–36 (1998)
4. Černo, P., Mráz, F.: Clearing restarting automata. Fund. Inform. **104**, 17–54 (2010)
5. Černo, P., Mráz, F.: Δ-clearing restarting automata and CFL. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS 6795, pp. 153–164. Springer, Berlin (2011)
6. Dahlhaus, E., Warmuth, M.: Membership for growing context-sensitive grammars is polynomial. J. Comput. Syst. Sci. **33**, 456–472 (1986)
7. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata—a survey. Inform. Comput. **209**, 456–470 (2011)
8. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
9. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Restarting automata. In: Reichel, H. (ed.) FCT'95, Proceedings. LNCS 965, pp. 283–292. Springer, Heidelberg (1995)
10. Kwee, K., Otto, F.: On the effects of nondeterminism on ordered restarting automata. In: Freivalds, R., Engels, G., Catania, B. (eds.) SOFSEM 2016, Proceedings. LNCS 9587, pp. 369–380. Springer, Heidelberg (2016)
11. Kwee, K., Otto, F.: Nondeterministic ordered restarting automata. Int. J. Found. Comput. Sci. **29**, 663–685 (2018)
12. Lopatková, M., Plátek, M., Sgall, P.: Towards a formal model for functional generative description: Analysis by reduction and restarting automata. Prague Bull. Math. Linguist. **87**, 7–26 (2007)
13. McNaughton, R., Narendran, P., Otto, F.: Church-Rosser Thue systems and formal languages. J. ACM **35**, 324–344 (1988)
14. Meyer, A., Fischer, M.: Economy of description by automata, grammars, and formal systems. In: IEEE Symposium on Switching and Automata Theory (SWAT 1971), pp. 188–191. IEEE Press (1971)
15. Mráz, F., Otto, F.: Ordered restarting automata for picture languages. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Min Tjoa, A. (eds.) SOFSEM 2014, Proceedings. LNCS 8327, pp. 431–442. Springer, Heidelberg (2014)
16. Otto, F.: Restarting automata. In: Ésik, Z., Martín-Vide, C., Mitrana, V. (eds.) Recent Advances in Formal Languages and Applications, Studies in Computational Intelligence, vol. 25, pp. 269–303. Springer, Berlin (2006)
17. Otto, F.: On the descriptional complexity of deterministic ordered restarting automata. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) DCFS 2014, Proceedings. LNCS 8614, pp. 318–329. Springer, Heidelberg (2014)
18. Otto, F.: On deterministic ordered restart-delete automata. In: Hoshi, M., Seki, S. (eds.) DLT 2018, Proceedings. LNCS 11088, pp. 529–540. Springer, Heidelberg (2018)
19. Otto, F.: On deterministic ordered restart-delete automata. Theor. Comput. Sci. **795**, 257–274 (2019)
20. Otto, F.: A characterization of the context-free languages by stateless ordered restart-delete automata. In: Chatzigeorgiou, A., Dondi, R., Herodotou, H., Kapoutsis, C., Manolopoulos, Y., Papadopoulos, G., Sikora, F. (eds.) SOFSEM 2020, Proceedings. LNCS 12011, pp. 39–50. Springer, Heidelberg (2020)
21. Otto, F., Černo, P., Mráz, F.: On the classes of languages accepted by limited context restarting automata. RAIRO - Inf. Theor. Appl. **48**, 61–84 (2014)

22. Otto, F., Kwee, K.: On the descriptional complexity of stateless deterministic ordered restarting automata. Inform. Comput. **259**, 277–302 (2018)
23. Rosenkrantz, D.J.: Matrix equations and normal forms for context-free grammars. J. ACM **14**, 501–507 (1967)