Reversible Ordered Restarting Automata

Friedrich Otto[1](✉), Matthias Wendlandt[2], and Kent Kwee[1]

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

[2] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392, Giessen, Germany
matthias.wendlandt@informatik.uni-giessen.de

Abstract. Stateless deterministic ordered restarting automata characterize the class of regular languages. Here we introduce a notion of *reversibility* for these automata and show that each regular language is accepted by such a reversible stateless deterministic ordered restarting automaton. We study the descriptional complexity of these automata, showing that they are exponentially more succinct than nondeterministic finite-state acceptors. We also look at the case of unary input alphabets.

Keywords: Restarting automaton · Reversibility · Descriptional complexity

1 Introduction

Reversibility is a property that has been investigated for various types of automata. It means that every configuration has a unique successor configuration and a unique predecessor configuration, that is, the automaton considered is forward and backward deterministic. The main motivation for studying this notion is the observation that information is lost in computations that are not reversible. Each Turing machine can be simulated by a reversible Turing machine [1], which shows that reversible Turing machines are just as expressive as general Turing machines. On the other hand, reversible deterministic finite-state acceptors (DFAs) are strictly less expressive than DFAs [13]. The notion of reversibility has also been studied for other types of automata, e.g., for pushdown automata [5] and for queue automata [6].

Here we introduce a notion of *reversibility* for a rather restricted class of restarting automata, the *stateless deterministic ordered restarting automata*. The restarting automaton was introduced in [4] as a formal device to model the linguistic technique of *analysis by reduction*. Since then many variants and extensions of the basic model have been introduced and studied (for an overview, see, e.g., [11]), and several classical families of formal languages, like the regular languages, the deterministic context-free languages, and the context-free languages, have been characterized by certain types of restarting automata.

J. Krivine and J.-B. Stefani (Eds.): RC 2015, LNCS 9138, pp. 60–75, 2015.
DOI: 10.1007/978-3-319-20860-2_4

The *deterministic ordered restarting automaton* (or *det-ORWW-automaton*) was introduced in [10] in the setting of picture languages. A det-ORWW-automaton has a finite-state control, a tape with end markers that initially contains the input, and a window of size three. Based on its state and the content of its window, the automaton can perform one of three types of operations: it may perform a *move-right step*, it may perform a combined *rewrite/restart step*, or it may perform an *accept step* (see Section 2 for details). The nondeterministic variant of the ordered restarting automaton accepts some languages that are not even context-free, but the deterministic variant accepts exactly the regular languages [10]. In fact, each det-ORWW-automaton can be simulated by a det-ORWW-automaton with only a single state [12]. As for the latter, states are essentially useless, they are called *stateless* det-ORWW-automata, or stl-det-ORWW-automata for short. In [12] the descriptional complexity of stl-det-ORWW-automata is investigated, using the size of the working alphabet as a complexity measure. It is shown that there is a double exponential trade-off when changing from a stl-det-ORWW-automaton to an equivalent DFA. Accordingly, we think that the stl-det-ORWW-automaton is a very interesting type of automaton, as it is a simple deterministic device that, nevertheless, yields succinct representations for the regular languages.

Here we introduce a notion of *reversibiliy* for stl-det-ORWW-automata, and we show that each regular language is accepted by a stl-det-ORWW-automaton that is reversible by presenting a transformation that turns a given stl-det-ORWW-automaton into an equivalent stl-det-ORWW-automaton that is reversible. This construction yields an exponential upper bound for the size increase of this transformation, but unfortunately we do not yet have a matching lower bound. Then we investigate the descriptional complexity of a reversible stl-det-ORWW-automaton in relation to the size of an equivalent DFA or NFA. We recall a simulation of stl-det-ORWW-automata by NFAs from [7], which also applies to stl-det-ORWW-automata that are reversible, and by considering a specific class of example languages we show that the resulting trade-off is indeed exponential. For DFAs, the corresponding trade-off is even double exponential. Finally, we consider the problem of turning a unary NFA into an equivalent stl-det-ORWW-automaton that is reversible. Here we obtain a quadratic size increase, but it remains open whether there is a matching lower bound. The paper closes with a short summary and a list of open problems.

2 Stateless Deterministic Ordered Restarting Automata

A *stl-det-ORWW-automaton* is a one-tape machine that is described by a 6-tuple $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, where Σ is a finite input alphabet, Γ is a finite tape alphabet such that $\Sigma \subseteq \Gamma$, the symbols $\rhd, \lhd \notin \Gamma$ serve as markers for the left and right border of the work space, respectively,

$$\delta : (((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \cup \{\rhd\lhd\}) \dashrightarrow \{\mathsf{MVR}\} \cup \Gamma \cup \{\mathsf{Accept}\}$$

is the (partial) *transition function*, and $>$ is a *partial ordering* on Γ. The transition function describes three different types of transition steps:

(1) A *move-right step* has the form $\delta(a_1a_2a_3) = \mathsf{MVR}$, where $a_1 \in \Gamma \cup \{\rhd\}$ and $a_2, a_3 \in \Gamma$. It causes M to shift the window one position to the right.
(2) A *rewrite/restart step* has the form $\delta(a_1a_2a_3) = b$, where $a_1 \in \Gamma \cup \{\rhd\}$, $a_2, b \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$ such that $a_2 > b$ holds. It causes M to replace the symbol a_2 in the middle of its window by the symbol b and to restart, that is, the window is repositioned on the left end of the tape.
(3) An *accept step* has the form $\delta(a_1a_2a_3) = \mathsf{Accept}$, where $a_1 \in \Gamma \cup \{\rhd\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$. It causes M to halt and accept. In addition, we allow an accept step of the form $\delta(\rhd\lhd) = \mathsf{Accept}$.

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

A *configuration* of a stl-det-ORWW-automaton M is a pair of words (α, β), where $|\beta| \geq 3$, and either $\alpha = \lambda$ (the empty word) and $\beta \in \{\rhd\} \cdot \Gamma^+ \cdot \{\lhd\}$ or $\alpha \in \{\rhd\} \cdot \Gamma^*$ and $\beta \in \Gamma \cdot \Gamma^+ \cdot \{\lhd\}$; here $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first three symbols of β. In addition, we admit the configuration $(\lambda, \rhd\lhd)$. A *restarting configuration* has the form $(\lambda, \rhd w \lhd)$ (usually simply written as $\rhd w \lhd$); if $w \in \Sigma^*$, then $(\lambda, \rhd w \lhd)$ is also called an *initial configuration*. Further, we use Accept to denote an *accepting configuration*, which is a configuration that M reaches by an accept step.

A computation of a stl-det-ORWW-automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the head is moved along the tape by MVR steps until a rewrite/restart step is performed and thus, a new restarting configuration is reached. If no further rewrite operation is performed, any computation necessarily finishes in a halting configuration – such a phase is called a *tail*. By $\vdash_M^c$ we denote the execution of a complete cycle, and $\vdash_M^{c^*}$ is the reflexive transitive closure of this relation.

An input $w \in \Sigma^*$ is accepted by M, if the computation of M which starts with the initial configuration $\rhd w \lhd$ ends with an accept step. The language consisting of all words that are accepted by M is denoted by $L(M)$.

Theorem 1. [10,12] $\mathsf{REG} = \mathcal{L}(\mathsf{det\text{-}ORWW}) = \mathcal{L}(\mathsf{stl\text{-}det\text{-}ORWW})$.

3 Reversibility for stl-det-ORWW-Automata

Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton. A combined rewrite/restart step of the form $\delta(abc) = b'$ takes M from a configuration of the form $(\rhd u, abcv\lhd)$ to the restarting configuration $\rhd uab'cv\lhd$. Now it is not at all clear how a *reverse transition function* could be designed that would transform the latter configuration back to the former configuration. Therefore, we consider a different notion of reversibility for our automata, a notion that is more in the spirit of restarting automata.

Definition 2. *A stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ is called* reversible*, if there exists a* reverse transition function

$$\delta^R : ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \dashrightarrow \{\mathsf{MVR}\} \cup \Gamma$$

such that, for all restarting configurations $\rhd w \lhd$ *and* $\rhd w' \lhd$ *that can occur within computations of* M, $\rhd w \lhd \vdash^{c}_{M} \rhd w' \lhd$ *iff* $\rhd w' \lhd \vdash^{c^R}_{M} \rhd w \lhd$*. Here* $\vdash^{c^R}_{M}$ *denotes a cycle that is realized by using the reverse transition function* δ^R*. We describe the above reversible stl-det-ORWW-automaton as* $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, \delta^R, >)$*, and we use the prefix rev- to denote reversible automata.*

Observe that in the definition above, we require that a cycle must be reversible by δ^R only for the case that the corresponding restarting configurations occur in a valid computation of M, that is, there exists an input $x \in \Sigma^*$ such that $\rhd w \lhd$ is reached from the initial configuration $\rhd x \lhd$ of M. This corresponds to the way reversibility is defined for queue automata in [6].

Obviously, rev-stl-det-ORWW-automata can only accept certain regular languages. However, in contrast to the situation for DFAs, they actually accept all regular languages, as we have the following result.

Theorem 3. *For each stl-det-ORWW-automaton* M *working on an alphabet with* n *letters, there exists a rev-stl-det-ORWW-automaton* R *with* $2^{O(n)}$ *letters such that* $L(R) = L(M)$ *holds.*

For deriving this result we need the following normal form result for stl-det-ORWW-automata. Here the *right distance* of a cycle $C : \rhd uabcv \lhd \vdash^{c}_{M} \rhd uab'cv \lhd$ of a stl-det-ORWW-automaton M is defined as $D_r(C) = |v|+1$, where $|v|$ denotes the length of the word v. Thus, $D_r(C)$ is the distance from the window to the right end of the tape at the time of rewriting in cycle C.

Definition 4. *A stl-det-ORWW-automaton* $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ *is said to be in* normal form *if it satisfies the following two conditions:*

1. *In any computation* $(C_0, C_1, C_2, \ldots, C_m)$ *of* M, $|D_r(C_i) - D_r(C_{i-1})| \leq 1$ *holds for all* $i = 1, \ldots, m$.
2. M *only accepts with the right delimiter* $\lhd$ *in its window.*

Lemma 5. *For each stl-det-ORWW-automaton* M *working on an alphabet with* n *letters, there exists an equivalent stl-det-ORWW-automaton* $\hat{M}$ *with an alphabet of size at most* $2(n+1)$ *that is in normal form.*

Proof. From M we obtain an equivalent stl-det-ORWW-automaton $M' = (\Sigma, \Gamma, \rhd, \lhd, \delta', >')$ that only accepts with the right delimiter in its window by using one extra symbol, that is, $|\Gamma| = n+1$ [7]. From M' we construct the stl-det-ORWW-automaton $\hat{M} = (\Sigma, \Delta, \rhd, \lhd, \delta, >)$ as follows:

- $\Delta = \Gamma \cup \{\, \underline{a} \mid a \in \Gamma \,\}$, which implies that $|\Delta| = 2 \cdot |\Gamma| = 2(n+1)$;
- $a > \underline{a}$ for all $a \in \Gamma$, and $\underline{a} > b$ for $a, b \in \Gamma$, if $a >' b$ holds;
- the transition function δ is defined as follows, where $a, b, c, d \in \Gamma$:

$$\delta(\rhd a \lhd) = \delta'(\rhd a \lhd) \text{ for } a \in \Gamma \cup \{\lambda\}, \qquad \delta(\underline{a}b\lhd) = \delta'(ab\lhd),$$

$$\delta(\rhd ab) = \begin{cases} c, \text{ if } \delta'(\rhd ab) = c, \\ \underline{a}, \text{ if } \delta'(\rhd ab) = \mathsf{MVR}, \end{cases} \qquad \delta(\underline{a}bc) = \begin{cases} d, \text{ if } \delta'(abc) = d, \\ \underline{b}, \text{ if } \delta'(abc) = \mathsf{MVR}, \end{cases}$$

$$\delta(\rhd \underline{a}b) = \begin{cases} c, & \text{if } \delta'(\rhd ab) = c, \\ \mathsf{MVR}, & \text{if } \delta'(\rhd ab) = \mathsf{MVR}, \end{cases} \qquad \delta(\underline{a}\,\underline{b}c) = \begin{cases} d, & \text{if } \delta'(abc) = d, \\ \mathsf{MVR}, & \text{if } \delta'(abc) = \mathsf{MVR}, \end{cases}$$

$$\delta(\rhd \underline{a}\,\underline{b}) = \mathsf{MVR}, \qquad \delta(\underline{a}\,\underline{b}\,\underline{c}) = \mathsf{MVR}.$$

The automaton $\hat{M}$ simulates a computation of M' by proceeding as follows. Assume that on input $x = x_1 \dots x_m$, M' will perform the cycle

$$(\lambda, \rhd x \lhd) = (\lambda, \rhd x_1 \dots x_{i-1} x_i x_{i+1} \dots x_m \lhd) \vdash^c_M (\lambda, \rhd x_1 \dots x_{i-1} a x_{i+1} \dots x_m \lhd).$$

Then $\hat{M}$ will first rewrite x_j, $j = 1, \dots, i-1$, into $\underline{x}_j$, and then it will rewrite x_i into a, producing the restarting configuration $(\lambda, \rhd \underline{x}_1 \dots \underline{x}_{i-1} a x_{i+1} \dots x_m \lhd)$. For the next cycle of M', there are three possibilities:

1. M' may rewrite x_{i-1} into a symbol b. Then $\hat{M}$ will rewrite $\underline{x}_{i-1}$ into b.
2. M' may rewrite a into a symbol b. Then so will $\hat{M}$.
3. Finally, M' may rewrite a symbol x_j for some $j \geq i+1$ into a symbol b. Then $\hat{M}$ will replace the symbols $a, x_{i+1}, \dots, x_{j-1}$ from left to right by the symbols $\underline{a}, \underline{x}_{i+1}, \dots, \underline{x}_{j-1}$, and then it will rewrite x_j into b.

Thus, in each cycle $\hat{M}$ either rewrites the first symbol from Γ from the left, or it rewrites the last symbol from $\Delta \setminus \Gamma$ from the left. It now follows easily that $\hat{M}$ is in normal form, and that $L(\hat{M}) = L(M') = L(M)$ holds. □

Now we can give the proof of Theorem 3.

Proof. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_M, >)$ be a stl-det-ORWW-automaton with $n = |\Gamma|$. Without loss of generality we can assume that M only accepts with the right marker $\lhd$ in its window. By Lemma 5 we can construct a stl-det-ORWW-automaton $\hat{M} = (\Sigma, \overline{\Gamma}, \rhd, \lhd, \hat{\delta}, >)$ that is equivalent to M and in normal form. Here $\overline{\Gamma} = \Gamma \cup \underline{\Gamma}$, where $\underline{\Gamma} = \{ \underline{a} \mid a \in \Gamma \}$, and hence, $\hat{M}$ has $2n$ letters.

From $\hat{M}$ we construct a rev-stl-det-ORWW-automaton $R = (\Sigma, \Delta, \rhd, \lhd, \delta, \delta^R, >)$ such that $L(R) = L(\hat{M}) = L(M)$ as follows:

– The tape alphabet Δ contains the input alphabet Σ and all triples of the form (L, W, R), where

  • W is a sequence of letters $W = (w_1, \dots, w_k)$ from $\overline{\Gamma}$ of length $1 \leq k \leq 2n$ such that $w_1 > w_2 > \dots > w_k$,

  • L is a sequence of positive integers $L = (l_1, \dots, l_{k-1})$ of length $k-1$ such that $l_1 \leq l_2 \leq \dots \leq l_{k-1} \leq 2n$, and

  • R is a sequence of positive integers $R = (r_1, \dots, r_{k-1})$ of length $k-1$ such that $r_1 \leq r_2 \leq \dots \leq r_{k-1} \leq 2n$.

  As in [7] the idea is that W encodes the sequence of letters that are produced by $\hat{M}$ in an accepting computation for a particular field, and L and R encode the information on the neighbouring letters to the left and to the right that are used to perform the corresponding rewrite operations. For example, the triple $(l_1, w_1, r_1) \in (L, W, R)$ means that w_1 is rewritten into w_2, while the left neighbouring field contains the l_1-th letter of its sequence W', and the right neighbouring field contains the r_1-th letter of its sequence W''. To simplify the discussion below, we simply interpret a symbol $a \in \Sigma \cup \{\rhd, \lhd\}$ as the triple $(L, W, R) = (\emptyset, (a), \emptyset)$.

Further, in order to ensure that triples in neighbouring fields are consistent with each other, the following notion has been introduced in [7]. For two finite non-decreasing sequences of integers $R' = (r'_1, \dots, r'_k)$ and $L = (\ell_1, \dots, \ell_s)$, where $k, s \geq 0$, we define a multiset $\mathrm{order}(R', L)$ as follows:

$$\mathrm{order}(R', L) = \{\, r'_i + i - 1 \mid i = 1, \dots, k \,\} \cup \{\, \ell_j + j - 1 \mid j = 1, \dots, s \,\}.$$

Now a pair of triples $((L', W', R'), (L, W, R))$ is called *consistent*, if $\mathrm{order}(R', L) = \{1, 2, \dots, k+s\}$, that is, it is the integer interval $[1, k+s]$. This notion of consistency will be of importance in the definition of the transition functions below.

– The ordering $>$ on Δ is defined by taking $(L, W, R) > (L', W', R')$, if there exist $b \in \overline{\Gamma}$ and $l, r \in \mathbb{N}$ such that $L' = (L, l)$, $W' = (W, b)$, and $R' = (R, r)$.

– For a triple $(L, W, R) = ((l_1, \dots, l_{k-1}), (w_1, \dots, w_k), (r_1, \dots, r_{k-1}))$, we take $\pi((L, W, R)) = w_k$ and $||(L, W, R)|| = k$. The transition function δ is defined as follows, where $A, B, C \in \Delta \cup \{\rhd, \lhd\}$ satisfy the condition that the pair (A, B) and the pair (B, C) are both consistent (see above):

$$\delta(ABC) = \begin{cases} \mathsf{MVR}, & \text{if } \hat{\delta}(\pi(A)\pi(B)\pi(C)) = \mathsf{MVR}, \\ \mathsf{Accept}, & \text{if } \hat{\delta}(\pi(A)\pi(B)\pi(C)) = \mathsf{Accept}, \\ ((L, ||A||), (W, b), (R, ||C||)), & \text{if } B = (L, W, R) \text{ and} \\ & \quad \hat{\delta}(\pi(A)\pi(B)\pi(C)) = b. \end{cases}$$

Thus, instead of replacing the symbol $\pi(B)$ by the symbol b, as $\hat{M}$ does, the automaton R appends the symbol b to the sequence of symbols W at the corresponding position. In addition, it appends the integers $||A||$ and $||C||$ to the lists L and R at this position, as these numbers point to the symbols (within the corresponding lists) that are at this moment contained in the neighbouring positions. Observe that $\delta(ABC)$ is undefined, if any of the pairs (A, B) or (B, C) is not consistent.

– Finally, the reverse transition function δ^R is defined as follows, where it is again required that the pairs (A, B) and (B, C) are consistent:

$$\delta^R(ABC) = \begin{cases} (L, W, R), & \text{if } B = ((L, l), (W, b), (R, r)),\ l = ||A||,\ r = ||C||, \\ & \quad \text{and } \hat{\delta}(\pi(A)\pi((L, W, R))\pi(C)) = b, \\ \mathsf{MVR}, & \text{if } C \neq \lhd \text{ and the above conditions are not met}, \\ \text{undefined}, & \text{if } C = \lhd \text{ and the above conditions are not met}. \end{cases}$$

It remains to verify that R accepts the same language as M, and that R is indeed reversible.

Claim 1. $L(R) = L(M)$.

Proof. Let $w \in \Sigma^*$ such that $w \in L(M) = L(\hat{M})$ holds. Assume that $|w| = m \geq 1$. As $w \in L(\hat{M})$, the computation of $\hat{M}$ on input w is accepting, that is, it consists of a sequence of $s \geq 0$ cycles and an accepting tail. If $s = 0$, then $\hat{M}$ simply scans w from left to right, and it accepts on reaching the symbol $\lhd$. From the definition of δ it follows that R does exactly the same on input w, that is, $w \in L(R)$ holds in this case. If $s \geq 1$, then the accepting computation of $\hat{M}$ on input w looks as follows:

$$(\lambda, \rhd w \lhd) \vdash^c_{\hat{M}} (\lambda, \rhd w_1 \lhd) \vdash^c_{\hat{M}} \cdots \vdash^c_{\hat{M}} (\lambda, \rhd w_s \lhd) \vdash^*_{\hat{M}} (\rhd w'_s, bc \lhd) \vdash_{\hat{M}} \mathsf{Accept},$$

where $w_1, \ldots, w_s \in \overline{\Gamma}^m$ and $w_s = w'_s bc$ for some letters $b, c \in \overline{\Gamma}$.

Let us look at the first cycle $(\lambda, \rhd w \lhd) \vdash^c_{\hat{M}} (\lambda, \rhd w_1 \lhd)$. It consists of $t_1 \geq 0$ move-right steps and a rewrite/restart step that replaces the symbol at position $t_1 + 1$ of w by a smaller symbol from $\overline{\Gamma}$, that is, $w = w'aw''$ for some $w' \in \Sigma^{t_1}$, $a \in \Sigma$, and $w'' \in \Sigma^{m-t_1-1}$, and $w_1 = w'bw''$ for some $b \in \overline{\Gamma}$ such that $a > b$ holds. From the definition of δ we see that, starting from the configuration $(\lambda, \rhd w \lhd)$, the automaton R will execute the following cycle:

$$(\lambda, \rhd w \lhd) = (\lambda, \rhd w'aw'' \lhd) \vdash^c_R (\lambda, \rhd w'Bw'' \lhd),$$

where $B = ((1), (a, b), (1))$. Thus, after simulating the first cycle the tape of R contains all the information on the tape content of $\hat{M}$ plus the information on the rewrite step that was executed during the first cycle. Observe that for all factors AB occurring on the tape of R during this computation, the corresponding pair (A, B) is trivially consistent.

Inductively it can be shown that R simulates the above computation of $\hat{M}$ cycle by cycle, in each rewrite/restart step not only simulating the corresponding rewrite/restart step of $\hat{M}$, but also encoding information on this very step. Hence, R accepts on input w, too, which shows that $L(M)$ is contained in $L(R)$.

Now assume conversely that $w \in \Sigma^*$ is accepted by R. As $w \in L(R)$, the computation of R on input w is accepting, that is, it consists of a sequence of $s \geq 0$ cycles and an accepting tail. If $s = 0$, then it follows from the definition of δ that R simply scans w from left to right and accepts on reaching the right delimiter $\lhd$. However, this means that on input w, $\hat{M}$ does exactly the same, that is, $w \in L(M)$ also holds in this case. Finally, if $s \geq 1$, then the computation of R on input w looks as follows:

$$(\lambda, \rhd w \lhd) \vdash^c_R (\lambda, \rhd W_1 \lhd) \vdash^c_R \cdots \vdash^c_R (\lambda, \rhd W_s \lhd) \vdash^*_R (\rhd W'_s, BC \lhd) \vdash_R \mathsf{Accept},$$

where $W_1, \ldots, W_s \in \Delta^m$ and $W_s = W'_s BC$ for some letters $B, C \in \Delta$. Interpreting π as a morphism from Δ^* to $\overline{\Gamma}^*$, it follows that the computation of $\hat{M}$ on input w looks as follows:

$$\begin{aligned}(\lambda, \rhd w \lhd) &\vdash^c_{\hat{M}} (\lambda, \rhd \pi(W_1) \lhd) \vdash^c_{\hat{M}} \cdots \\ &\vdash^c_{\hat{M}} (\lambda, \rhd \pi(W_s) \lhd) \vdash^*_{\hat{M}} (\rhd \pi(W'_s), \pi(B)\pi(C) \lhd) \vdash_{\hat{M}} \mathsf{Accept},\end{aligned}$$

which means that $w \in L(M)$. Here the consistency of each pair (A, B) that corresponds to a factor AB of the tape contents of R implies that the corresponding steps of $\hat{M}$ are indeed possible. It follows that $L(R) = L(M)$ holds. $\square$

We now complete the proof of Theorem 3 by establishing the following claim.

Claim 2. The stl-det-ORWW-automaton R is reversible.

Proof. Let $w, z \in \Delta^m$ such that $(\lambda, \rhd w \lhd) \vdash_R^c (\lambda, \rhd z \lhd)$ holds. Then $w = uAv$ and $z = uBv$ for some $u \in \Delta^r$, $A, B \in \Delta$, and $v \in \Delta^{m-r-1}$, that is, $u = U_1 \dots U_r$, $V = V_1 \dots V_{m-r-1}$ for some $U_1, \dots, U_r, V_1, \dots, V_{m-r-1} \in \Delta$, and

$$\begin{aligned}(\lambda, \rhd w \lhd) &\vdash_R^r (\rhd U_1 \dots U_{r-1}, U_r A V_1 \dots V_{m-r-1} \lhd)\\ &\vdash_R (\lambda, \rhd U_1 \dots U_r B V_1 \dots V_{m-r-1} \lhd).\end{aligned}$$

From the definition of δ we can conclude the following properties:

1. The pairs $(\rhd, U_1), (U_1, U_2), \dots, (U_{r-1}, U_r), (U_r, A), (A, V_1)$ are all consistent, as the factors $\rhd U_1$, $U_1U_2, \dots, U_{r-1}U_r, U_rA$, and AV_1 are all scanned by R during this cycle.
2. $\delta(\rhd U_1U_2) = \delta(U_1U_2U_3) = \dots = \delta(U_{r-1}U_rA) = \mathsf{MVR}$, and so $\mathsf{MVR} = \hat{\delta}(\rhd\pi(U_1)\pi(U_2)) = \hat{\delta}(\pi(U_1)\pi(U_2)\pi(U_3)) = \dots = \hat{\delta}(\pi(U_{r-1})\pi(U_r)\pi(A))$.
3. $\delta(U_rAV_1) = B$, and so $\hat{\delta}(\pi(U_r)\pi(A)\pi(V_1)) = \pi(B)$, where $A = (L, W, R)$ and $B = ((L, ||U_r||), (W, b), (R, ||V_1||))$.

It follows immediately that also the pairs (U_r, B) and (B, V_1) are consistent. Now we apply the reverse transition function δ^R starting with the configuration $(\lambda, \rhd z \lhd) = (\lambda, \rhd U_1 \dots U_r B V_1 \dots V_{m-r-1} \lhd)$. It looks for the first position from the left where a rewrite can be 'undone.' Obviously, if the factor U_rBV_1 is reached, then $B = ((L, ||U_r||), (W, b), (R, ||V_1||))$ is rewritten into $A = (L, W, R)$, which yields the cycle

$$(\lambda, \rhd z \lhd) \vdash_R^{c^R} (\rhd U_1 \dots U_{r-1}, U_r A V_1 \dots V_{m-r-1} \lhd) = (\lambda, \rhd w \lhd).$$

So we must argue that there is no factor $U_{i-1}U_iU_{i+1}$, $1 \leq i \leq r$, such that δ^R would rewrite the letter U_i. Assume to the contrary that such an index exists, that is, $U_i = ((L', l'), (W', b'), (R', r'))$ such that $||U_{i-1}|| = l'$, $||U_{i+1}|| = r'$, and $\hat{\delta}(\pi(U_{i-1})\pi((L', W', R'))\pi(U_{i+1})) = b'$ for some $i \leq r$. Hence, starting from the configuration $(\lambda, \rhd \pi(U_1) \dots \pi(U_{i-1})\pi((L', W', R'))\pi(U_{i+1}) \dots \pi(V_{m-r-1} \lhd)$, $\hat{M}$ would rewrite the letter $\pi((L', W', R'))$ into the letter b'. As $\hat{M}$ is in normal form, its next rewrite would occur at position $i-1$, i, or $i+1$, which means that R, which simulates $\hat{M}$ step by step, would also perform a rewrite at one of these positions when starting from the configuration $(\lambda, \rhd w \lhd)$. Thus, it follows that $i = r$, that is, we have $U_r = (((L', l'), (W', b'), (R', r'))$, $||U_{r-1}|| = l'$, and $||B|| = r'$. However, as the right sequence (R', r') of U_r ends with $r' = ||B||$, while the left sequence $(L, ||U_r||)$ of B ends with the number $||U_r||$, we see that the pair $((R', r'), (L, ||U_r||))$ is not consistent, which contradicts our observation above. Thus, when using the reverse transition function δ^R, the above cycle is indeed inverted. □

This completes the proof of Theorem 3. □

Hence, we obtain the following characterization.

Corollary 6. $\mathsf{REG} = \mathcal{L}(\mathsf{rev\text{-}stl\text{-}det\text{-}ORWW})$.

4 Descriptional Complexity

We are interested in the descriptional complexity of rev-stl-det-ORWW-automata and its relation to that of DFAs and NFAs, where we use the number of states as a measure for the size of a DFA or NFA, and we use the number of symbols in its tape alphabet as the complexity measure for a stl-det-ORWW-automaton.

Theorem 7. *For each DFA $A = (Q, \Sigma, q_0, F, \varphi)$, there exists a rev-stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, \delta^R, >)$ such that $L(M) = L(A)$ and $|\Gamma| = |Q| \cdot (|\Sigma| + 1)$.*

Proof. Let $A = (Q, \Sigma, q_0, F, \varphi)$ be a DFA that accepts a language $L \subseteq \Sigma^*$. We take $\Gamma = \Sigma \cup (Q \times \Sigma)$, define $a > (q, a)$ for all $a \in \Sigma$ and all $q \in Q$, and define the transition functions δ and δ^R as follows, where $a, b, c \in \Sigma$ and $p, q, q' \in Q$:

$$\begin{array}{llll}
\delta(\rhd\lhd) & = \mathsf{Accept}, \text{ if } \lambda \in L(A), & \delta(\rhd(q,a)b) & = \mathsf{MVR}, \\
\delta(\rhd a\lhd) & = \mathsf{Accept}, \text{ if } a \in L(A), & \delta(\rhd(q,a)(p,b)) & = \mathsf{MVR}, \\
\delta(\rhd ab) & = (q,a), \ \text{ if } \varphi(q_0,a) = q, & \delta((p,a)(q,b)(q',c)) & = \mathsf{MVR}, \\
\delta((q,a)bc) & = (p,b), \ \text{ if } \varphi(q,b) = p, & \delta((p,a)(q,b)c) & = \mathsf{MVR}, \\
\delta((q,a)b\lhd) & = (p,b), \ \text{ if } \varphi(q,b) = p, & \delta((q,a)(p,b)\lhd) & = \mathsf{Accept}, \text{ if } p \in F, \\
 & & & \\
\delta^R(\rhd(q,a)b) & = a, \qquad \text{ if } \varphi(q_0,a) = q, & \delta^R(\rhd(p,a)(q,b)) & = \mathsf{MVR}, \\
\delta^R((p,a)(q,b)c) & = b, \qquad \text{ if } \varphi(p,b) = q, & \delta^R((p,a)(q,b)(q',c)) & = \mathsf{MVR}. \\
\delta^R((p,a)(q,b)\lhd) & = b, \qquad \text{ if } \varphi(p,b) = q, & &
\end{array}$$

Thus, given $w = a_1 \ldots a_n$ as input, where $n \geq 2$ and $a_1, \ldots, a_n \in \Sigma$, M rewrites w from left to right into the word $(q_1, a_1) \ldots (q_{n-1}, a_{n-1})(q_n, a_n)$, where $q_i = \varphi(q_0, a_1 \ldots a_i)$, $1 \leq i \leq n$, and this word is then accepted in a tail computation if $q_n \in F$, that is, M accepts on input w iff $\varphi(q_0, a_1 \ldots a_n) = \varphi(\varphi(q_0, a_1 \ldots a_{n-1}), a_n) = \varphi(q_{n-1}, a_n) = q_n \in F$, that is, iff A accepts on input w. Hence, we see that $L(M) = L$ holds.

From the definition of δ^R it follows immediately that by δ^R, a restarting configuration of the form $\rhd(q_1, a_1) \ldots (q_{i-1}, a_{i-1})(q_i, a_i)a_{i+1} \ldots a_n\lhd$ is transformed back into the restarting configuration $\rhd(q_1, a_1) \ldots (q_{i-1}, a_{i-1})a_i a_{i+1} \ldots a_n\lhd$. Thus, M is indeed reversible in the sense of the above definition. $\square$

Hence, each DFA can be simulated by a rev-stl-det-ORWW-automaton of about the same size and, correspondingly, each NFA of size n can therefore be simulated by a rev-stl-det-ORWW-automaton of size $O(2^n)$. Unfortunately, we do not yet have a corresponding lower bound.

Next we turn to the converse transformation. In [7] the following result is shown, which, of course, also holds for stl-det-ORWW-automata that are reversible.

Theorem 8. [7] *For each stl-det-ORWW-automaton M with n letters, there exists an NFA A with $2^{O(n)}$ states such that $L(A) = L(M)$ holds.*

In particular, it follows that, for each (reversible) stl-det-ORWW-automaton M with n letters, there exists an equivalent DFA B with $2^{2^{O(n)}}$ states. We will now prove that these upper size bounds for turning a rev-stl-det-ORWW-automaton into an equivalent NFA (DFA) are sharp (up to the O-notation). For this, we consider a collection of example language B_n ($n \geq 3$) that are slight variations of languages considered in [14].

Let $\Sigma = \{0, 1, \#, \$\}$. For $n \geq 3$, let B_n be the following regular language:

$$B_n = \{\, v_1\#v_2\#\ldots\#v_m\$u \mid m \geq 1,\ v_1, \ldots, v_m, u \in \{0,1\}^n,\ \exists i : v_i = u \,\}.$$

Using standard techniques the following results can be shown on B_n.

Lemma 9. (a) *Every NFA for B_n has at least 2^n states.*
(b) *Every DFA for B_n has at least 2^{2^n} states.*

Now the following technical result yields the intended lower bounds.

Proposition 10. *The language B_n is accepted by a rev-stl-det-ORWW-automaton that has a tape alphabet of size $O(n)$.*

Proof. It has already been observed in [12] that the language B_n is accepted by a stl-det-ORWW-automaton that only uses $O(n)$ letters, but here we have to show that this also holds for a stl-det-ORWW-automaton that is reversible.

The rev-stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ for B_n will work in n phases. Let $w = v_1\#\ldots\#v_m\$u$ be given as input, where $m \geq 1$ and $v_1, \ldots, v_m, u \in \{0,1\}^n$, and let $v_j = v_{j,1}\ldots v_{j,n}$, $1 \leq j \leq m$, and let $u = u_1 \ldots u_n$. In phase i, M will shift the information about the letter u_i to the left until this information reaches the letter $v_{1,i}$. While doing so, it compares this letter to the letter $v_{j,i}$ for all $j = 2, \ldots, m$, storing the results of these comparisons by replacing the symbol $v_{j,i}$ by some appropriate auxiliary symbol. Finally, after phase n has been completed, M moves across the current tape content and checks whether there is a syllable v_j all of its letters have been matched successfully. Now we describe the automaton M in some detail.

First we define the tape alphabet Γ as

$$\begin{aligned}\Gamma = \Sigma \cup \{\, &[*, s, a, i, b] \mid 1 \leq i \leq n, a \in \Sigma, b \in \{0,1\}, s \in \{+,-\} \,\} \cup \\ \{\, &[b, i], [a, i, b], [s, a, i, b] \mid 1 \leq i \leq n, a \in \Sigma, b \in \{0,1\}, s \in \{+,-\} \,\} \cup \\ \{\, &\underline{[b, i]}, \underline{[a, i, b]}, \underline{[s, a, i, b]} \mid 1 \leq i \leq n-1, a \in \Sigma, b \in \{0,1\}, s \in \{+,-\} \,\},\end{aligned}$$

that is, Γ contains $68n - 22 \in O(n)$ letters. Next we define the partial order on Γ as follows, where $a, b_1, b_2, b_3 \in \{0,1\}$ and $s \in \{+,-\}$:

$$a > [a, i] > [a, i, b_1] > [s, a, i, b_2] > [*, s, a, i, b_3] \text{ for all } 1 \leq i \leq n,$$

$$[\$, i, b_1] > \underline{[\$, i, b_1]} > [\$, i+1, b_2] \text{ for all } 1 \leq i \leq n-1,$$

$$[\#, i, b_1] > \underline{[\#, i, b_1]} > [\#, i+1, b_2] \text{ for all } 1 \leq i \leq n-1,$$

$$[s, a, i, b_1] > \underline{[a, i]} > \underline{[a, i, b_1]} > \underline{[s, a, i, b_2]} > [a, i+1, b_3] \text{ for all } 1 \leq i \leq n-1.$$

Finally, we define the transition functions δ and δ^R, dividing this description into n phases as mentioned above.

1. M moves its read/write window to the letter u_1, rewrites u_1 into $[u_1, 1]$, and moves the information on u_1 to the left. Here the following transitions are used, where $a_1, a_2, a_3, b \in \{0, 1\}$ and $s \in \{+, -\}$:

$$\begin{array}{rclrcll}
\delta(\rhd a_1 a_2) &=& \mathsf{MVR}, & \delta(a_1 a_2 [\$, 1, b]) &=& [a_2, 1, b], & \\
\delta(a_1 a_2 a_3) &=& \mathsf{MVR}, & \delta(a_1 a_2 [a_3, 1, b]) &=& [a_2, 1, b], & \\
\delta(a_1 a_2 \#) &=& \mathsf{MVR}, & \delta(\# a_1 [a_2, 1, b]) &=& [+, a_1, 1, b], & \text{if } a_1 = b, \\
\delta(a_1 \# a_2) &=& \mathsf{MVR}, & \delta(\# a_1 [a_2, 1, b]) &=& [-, a_1, 1, b], & \text{if } a_1 \neq b, \\
\delta(\# a_1 a_2) &=& \mathsf{MVR}, & \delta(a_1 \# [s, a_2, 1, b]) &=& [\#, 1, b], & \\
\delta(a_1 a_2 \$) &=& \mathsf{MVR}, & \delta(a_1 a_2 [\#, 1, b]) &=& [a_2, 1, b], & \\
\delta(a_1 \$ a_2) &=& \mathsf{MVR}, & \delta(\rhd a_1 [a_2, 1, b]) &=& [*, +, a_1, 1, b], & \text{if } a_1 = b, \\
\delta(\$ b a_2) &=& [b, 1], & \delta(\rhd a_1 [a_2, 1, b]) &=& [*, -, a_1, 1, b], & \text{if } a_1 \neq b. \\
\delta(a_1 \$ [b, 1]) &=& [\$, 1, b], & & & &
\end{array}$$

It is easily seen that these steps can be reversed by defining the reverse transition function δ^R accordingly.

2. In the following $n-1$ phases M does the same with the remaining letters of u. In each of these phases M first marks all letters previously rewritten by an underline until it reaches the next symbol of u. Here we skip the transitions for these rewrite steps, continuing with those transitions that are used when u_i is encountered. Here $a_1, a_2, a_3, b, b_1 \in \{0, 1\}$ and $s_1, s_2 \in \{+, -\}$:

$$\begin{array}{rcll}
\delta(\underline{[a_1, i-1]} b a_2) &=& [b, i] & \text{for } 2 \leq i \leq n-1, \\
\delta(\underline{[a_1, n-1]} b \lhd) &=& [b, n], & \\
\delta(\underline{[\$, 1, b_1][b_1, 1]}[b, 2]) &=& [b_1, 2, b], & \\
\delta(\underline{[a_1, i-1, b_1][b_1, i-1]}[b, i]) &=& [b_1, i, b] & \text{for } 3 \leq i \leq n, \\
\delta(\underline{[a_1, i-1, b_1][\$, i-1, b_1]}[a_2, i, b]) &=& [\$, i, b] & \text{for } 2 \leq i \leq n, \\
\delta(\underline{[\$, i-1, b_1][a_1, i-1, b_1]}[a_2, i, b]) &=& [a_1, i, b] & \text{for } 3 \leq i \leq n, \\
\delta(\underline{[a_1, i-1, b_1][a_2, i-1, b_1]}[a_3, i, b]) &=& [a_2, i, b] & \text{for } 2 \leq i \leq n, \\
\delta(\underline{[a_1, i-1, b_1][a_2, i-1, b_1]}[\$, i, b]) &=& [a_2, i, b] & \text{for } 2 \leq i \leq n, \\
\delta(\underline{[+, a_1, i-1, b_1][b, i-1, b_1]}[a_3, i, b]) &=& [+, b, i, b] & \text{for } 2 \leq i \leq n, \\
\delta(\underline{[+, a_1, i-1, b_1][a_2, i-1, b_1]}[a_3, i, b]) &=& [-, a_2, i, b] & \text{for } 2 \leq i \leq n \\
 & & & \text{and } a_2 \neq b, \\
\delta(\underline{[-, a_1, i-1, b_1][a_2, i-1, b_1]}[a_3, i, b]) &=& [-, a_2, i, b] & \text{for } 2 \leq i \leq n, \\
\delta(\underline{[\#, i-1, b_1][s_1, a_2, i-1, b_1]}[s_2, a_3, i, b]) &=& [s_1, a_2, i, b] & \text{for } 2 \leq i \leq n, \\
\delta(\underline{[a_1, i-1, b_1][\#, i-1, b_1]}[s_1, a_2, i, b]) &=& [\#, i, b] & \text{for } 2 \leq i \leq n, \\
\delta(\underline{[a_1, i-1, b_1][a_2, i-1, b_1]}[\#, i, b]) &=& [a_2, i, b] & \text{for } 2 \leq i \leq n-1, \\
\delta(\underline{[+, a_1, n-1, b_1][b, n-1, b_1]}[\#, n, b]) &=& [+, b, n, b], & \\
\delta(\underline{[+, a_1, n-1, b_1][a_2, n-1, b_1]}[\#, n, b]) &=& [-, a_2, n, b] & \text{for } a_2 \neq b, \\
\delta(\underline{[-, a_1, n-1, b_1][a_2, n-1, b_1]}[\#, n, b]) &=& [-, a_2, n, b], & \\
\delta(\underline{[s_1, a_1, i-1, b_1][s_2, a_2, i-1, b_1]}[s_3, a_3, i, b]) &=& [s_2, a_2, i, b] & \text{for } 3 \leq i \leq n, \\
\delta(\underline{[*, +, a_1, n-1, b_1][b, n-1, b_1]}[\#, n, b]) &=& [*, +, b, n, b], & \\
\delta(\underline{[*, +, a_1, n-1, b_1][a_2, n-1, b_1]}[\#, n, b]) &=& [*, -, a_2, n, b] & \text{for } a_2 \neq b, \\
\delta(\underline{[*, -, a_1, n-1, b_1][a_2, n-1, b_1]}[\#, n, b]) &=& [*, -, a_2, n, b], & \\
\delta(\underline{[*, +, a_1, i-1, b_1][b, i-1, b_1]}[a_2, i, b]) &=& [*, +, b, i, b] & \text{for } 2 \leq i \leq n-1, \\
\delta(\underline{[*, +, a_1, i-1, b_1][a_2, i-1, b_1]}[a_2, i, b]) &=& [*, -, a_2, i, b] & \text{for } 2 \leq i \leq n-1 \\
 & & & \text{and } a_2 \neq b, \\
\delta(\underline{[*, -, a_1, i-1, b_1][a_2, i-1, b_1]}[a_2, i, b]) &=& [*, -, a_2, i, b] & \text{for } 2 \leq i \leq n-1.
\end{array}$$

From the information stored within the letters that are used to replace the letters $v_{1,1}$ to $v_{1,n}$ it is easily seen that also these transitions can be reversed by defining δ^R accordingly.

3. Finally M checks whether the final tape contents contains a factor of the form $[*,+,a_1,n,b][*,+,a_2,n,b][\#,n,b]$, $[+,a_1,n,b][+,a_2,n,b][\#,n,b]$, $[*,+,a_1,n,b][*,+,a_2,n,b][\$,n,b]$, or $[+,a_1,n,b][+,a_2,n,b][\$,n,b]$, and it accepts in the affirmative.

It remains to argue that $L(M) = B_n$ holds. From the construction it is rather straightforward to see that M accepts all words from the language B_n. Hence, it remains to show that M does not accept any other words.

So let $w \in \Sigma^*$ be a given input word that M accepts. We must show that w meets all of the following properties:

(a) $w = v_1\#v_2\#\ldots\#v_m\$u$, where $m \geq 1$ and $v_1,\ldots,v_m,u \in \{0,1\}^*$.
(b) $|u| = n$.
(c) $|v_1| = \ldots = |v_m| = n$.
(d) There exists an index $i \in \{1,\ldots,m\}$ such that $v_i = u$ holds.

In each phase i, the rewriting process is initialised by rewriting the letter u_i into the symbol $[u_i,i]$. The symbol $[u_1,1]$ can only be rewritten if it is immediately to the right of the symbol \$, and, for $2 \leq i \leq n$, the symbol $[u_i,i]$ is produced only immediately to the right to a symbol $\underline{[u_{i-1},i-1]}$. Finally, $[u_n,n]$ can only be written immediately to the left of the symbol $\lhd$. This ensures property (b). In addition, the MVR steps of the initial phase make sure that (a) holds. The rules for the comparison mark exactly one letter of v_i in each phase, which ensures property (c), and the final scan only accepts if there is a syllable v_i that coincides with u, which proves (d). Thus, $L(M) = B_n$ follows. □

Together with Theorem 8 these results show the following.

Corollary 11. (a) *There is an exponential trade-off for turning a rev-stl-det-ORWW-automaton into an equivalent NFA.*
(b) *There is a double exponential trade-off for turning a rev-stl-det-ORWW-automaton into an equivalent DFA.*

5 Unary Languages

From Theorem 7 we know that a DFA with n states and an input alphabet of size m can be converted into an equivalent rev-stl-det-ORWW-automaton that has an alphabet of size $n \cdot (m+1)$. For an NFA with n states, we thus obtain an equivalent rev-stl-det-ORWW-automaton with an alphabet of size $2^n \cdot (m+1)$. Here we prove that in the unary case, that is, if $m = 1$, we can do better.

Theorem 12. *From an NFA A with n states that accepts a unary language $L(A)$, an equivalent rev-stl-det-ORWW-automaton M with an alphabet of size $O(n^2)$ can be constructed.*

Proof. In [2] it is shown that each NFA with n states can be converted into an equivalent NFA with $O(n^2)$ states that is in *Chrobak normalform*, which means that the latter NFA consists of a chain of states of length at most n^2 which leads to a finite number of disjoint loops that have altogether at most n states.

So from A we first construct an NFA $B = (S, \{a\}, s_0, F, \delta_B)$ in Chrobak normalform, where $S = C \cup P$. Here $C = \{s_0, c_1, \ldots, c_k\}$ is a chain of length $k \leq n^2$ and $P = \bigcup_{i=1}^{l} P_i$, where, for $i = 1, \ldots, l$, $P_i = \{(0, p_i), (1, p_i), \ldots, (p_i - 1, p_i)\}$ is a loop of length p_i such that $\sum_{i=1}^{l} p_i \leq n$.

The tape alphabet of the rev-stl-det-ORWW-automaton $M = (\{a\}, \mathit{\Gamma}, \rhd, \lhd, \delta, \delta^R, >)$ is

$$\mathit{\Gamma} = \{a\} \cup C \cup (P \times \{<, >\}) \cup (P \times \{0, 1, \ldots, l-1\}),$$

where the components $\{<, >\}$ are used to locate the cell where the last rewrite operation has been performed. We see that $\mathit{\Gamma}$ contains $O(n^2)$ letters.

M works as follows. As long as B is still within the chain C, M processes the input letter by letter from left to right by replacing each letter a by the state which B reaches by reading the current input symbol:

$$\delta(\rhd aa) = c_1 \text{ and } \delta(c_i aa) = c_{i+1} \text{ for } 1 \leq i \leq k-1.$$

After each restart M has to find the next a. In addition, M must accept if it reaches the right end in a final state:

$$\begin{aligned} \delta(\rhd c_1 a) &= \mathsf{MVR}, \\ \delta(\rhd c_1 c_2) &= \mathsf{MVR}, \\ \delta(c_i c_{i+1} c_{i+2}) &= \mathsf{MVR} \quad \text{for } 1 \leq i \leq k-2, \\ \delta(c_i c_{i+1} a) &= \mathsf{MVR} \quad \text{for } 1 \leq i \leq k-1, \\ \delta(c_i c_{i+1} \lhd) &= \mathsf{Accept}, \text{ if } c_{i+1} \in F. \end{aligned}$$

Clearly this part is reversible.

If the length of the input a^m exceeds the length k of the chain C, M simulates the computations of B for all loops P_i, $1 \leq i \leq l$, simultaneously. For that l subsequent symbols $(i_l, p_l, z_l), (i_{l-1}, p_{l-1}, z_{l-1}), \ldots, (i_1, p_1, z_1)$, where $z_l, \ldots, z_1 \in \{<, >\}$, are used to represent the states within the different loops that B could be in. So for an input a^{k+l+r}, M will reach the tape content

$$\rhd c_1 \cdots c_k (i_l, p_l, z_l)(i_{l-1}, p_{l-1}, z_{l-1}) \cdots (i_1, p_1, z_1) a^r \lhd$$

after $k + l$ cycles. This is interpreted as follows: B is in the state (i_1, p_1) after reading a^{k+l} and using the first loop P_1, it is in state (i_{l-1}, p_{l-1}) after reading a^{k+2} and using the loop P_{l-1}, and it is in state (i_l, p_l) after reading a^{k+1} and using the loop P_l. So the different possible ways B can use are tried sequentially by M. The computation continues by shifting the information on the various loops to the right step by step, beginning with the state of loop P_1. Here the third components $z_j \in \{<, >\}$ are used to indicate the position at which the next rewrite must be performed. After processing another factor a^j, the tape contains the prefix $\rhd c_1 \cdots c_k$, which is followed by the factor

$$(i_{l,0}, p_l, z_{l,0})(i_{l,1}, p_l, z_{l,1}) \cdots (i_{l,j}, p_l, z_{l,j})(i_{l-1,j}, p_{l-1}, z_{l-1,j}) \cdots (i_{1,j}, p_1, z_{1,j}),$$

which is followed by the suffix $a \cdots a\lhd$. The corresponding transitions are defined as follows:

$$\begin{aligned}
\delta((i,p_j,>)(i',p_{j'},>)a) &= \mathsf{MVR}, \\
\delta((i,p_j,>)(i',p_{j'},>)(i'',p_{j''},>)) &= \mathsf{MVR}, \\
\delta((i,p_j,>)(i',p_{j'},>)(i'',p_{j''},<)) &= \mathsf{MVR}, \text{ if } p_{j'} \neq p_{j''}, \\
\delta((i,p_j,>)(i',p_{j'},<)(i'',p_{j''},<)) &= (i',p_{j'},>), \\
\delta((i,p_j,>)(i',p_{j'},>)(i'',p_{j''},<)) &= (i+1 \bmod p_j, p_j, <), \text{ if } p_{j'} = p_{j''} \text{ and } i' < l, \\
\delta((i,p_j,>)(i',p_{j'},>)(i'',p_{j''},<)) &= \mathsf{MVR}, \text{ if } p_{j'} = p_{j''} \text{ and } i' = l, \\
\delta((i,p_1,>)aa) &= (i+1 \bmod p_1, p_1, <).
\end{aligned}$$

In the above situation there are two possibilities for further rewrite steps. If, for each letter $t = (i, p_j, z)$ it holds that $z => $, then the previous rewrite just took place at the rightmost of these symbols, and the next rewrite operation has to rewrite the first of the remaining letters a.

The other possibility is that there is exactly one position on the tape where the components $>$ and $<$ are side by side, that is, there is a factor of the form t_0t_1, where $t_0 = (i, p_j, >)$ and $t_1 = (i', p'_j, <)$. Then the next rewrite operation either rewrites t_0 or t_1. If $p_j = p'_j$ and $j < l$, then M is in the process of shifting the current cycle simulations one step to the right, and accordingly, the next rewrite operation is applied to t_0. Otherwise, M starts a new shifting process and so, it must find the right end. Accordingly, the next rewrite operation is applied to t_1. Together with the fact that the previous inscription can be restored from the information in the left and right neigbouring letters this ensures the property of working reversibly. Of course, there must be special transitions for moving from the chain states to the loop states of B, and also the special case of an empty chain must be taken care of.

Finally, when the rewrites of M reach the right end of the tape, then it must be checked whether at least one of these cycles accepts. For this, M sends a signal to the left that tests, for one loop after another, whether it would lead to acceptance at the right end of the input:

$$\begin{aligned}
\delta((i,p_2,>)(i',p_1,>)\lhd) &= \mathsf{Accept}, & &\text{if } (i',p_1) \in F, \\
\delta((i,p_2,>)(i',p_1,>)\lhd) &= (i',p_1,0), & &\text{if } (i',p_1) \notin F, \\
\delta(s(i,p_j,>)(i',p_{j'},r)) &= \mathsf{Accept}, & &\text{if } (i'+r \bmod p_{j'}, p_{j'}) \in F, \\
\delta(s(i,p_j,>)(i',p_{j'},r)) &= (i,p_j,r+1), & &\text{if } (i'+r \bmod p_{j'}, p_{j'}) \notin F,
\end{aligned}$$

where $1 \leq j \leq l$, and $s \in (P \times \{>,<\}) \cup C \cup \{\rhd\}$.

Again this part is reversible, since the letter (i, p_j, r), where the last rewrite step has been executes, is the first one with a number r. If the input is not in $L(M)$, then none of these tests is successful. It follows that $L(M) = L(B) = L(A)$ holds. □

The main idea of the construction in the proof of Theorem 12 can be also used to get an exponential lower bound for the conversion of a rev-stl-det-ORWW-automaton into a DFA or an NFA. In [2] the simulation costs between DFAs and NFAs and two-way DFAs (2DFAs) are investigated. In many cases an upper bound for the simulation costs is given by the Landau function [8,9]

$$F(n) = \max\{\, \mathrm{lcm}(p_1, p_2 \ldots, p_l) \mid p_1, p_2, \ldots, p_l \geq 1 \text{ and } p_1 + p_2 + \cdots + p_l = n \,\},$$

where lcm denotes the least common multiple. The best known approximation for F is shown in [15]. Bounds derived from this result [3] are

$$F(n) \in \Omega\left(e^{\sqrt{n \cdot \ln(n)}}\right) \text{ and } F(n) \in O\left(e^{\sqrt{n \cdot \ln(n)}(1+o(1))}\right).$$

It can be concluded from the results of [2] that a DFA as well as an NFA needs at least $F(n)$ states for accepting the language $L_n = \{\, a^m \mid m \bmod p_i \equiv 0 \text{ for all } 1 \leq i \leq l \,\}$, where the p_i are chosen such that $p_1, p_2, \ldots, p_l \geq 2$, $p_1 + p_2 + \cdots + p_l \leq n$, and $\mathrm{lcm}(p_1, p_2, \ldots, p_l) = F(n)$.

We now modify the behavior of M from the proof of Theorem 12 so that it accepts L with $O(n)$ symbols. Again M simulates the computation of all loops $P_1, P_2, \ldots, P_l$. It works in the same way as above until the signal of p_1 reaches the right end, but then the modified M checks whether *all* of the l possible loops fit the input. This can be achieved easily by a modification of the accepting transitions. Thus, we have the following lower bound result.

Corollary 13. *For each $n \in \mathbb{N}$, there is a rev-stl-det-ORWW-automaton with $O(n)$ tape symbols that accepts a unary language L such that any DFA or NFA for L needs at least $F(n)$ states.*

6 Conclusion and Open Problems

We have introduced a type of *reversible* stl-det-ORWW-automaton and shown that it characterizes the regular languages. We have studied its descriptional complexity by taking the size of the tape alphabet as the complexity measure for such an automaton, and we have established an exponential (double exponential) trade-off for turning a rev-set-det-ORWW-automaton into an equivalent NFA (DFA). For the converse transformation we have an exponential upper bound in the case of NFAs, and we have presented a transformation that turns any stl-det-ORWW-automaton into an equivalent rev-stl-det-ORWW-automaton at the cost of an exponential increase in size. However, the following questions are still open:

1. What is the trade-off for turning a stl-det-ORWW-automaton into an equivalent stl-det-ORWW-automaton that is reversible? In Theorem 3 an exponential upper bound is given. Can this bound be improved, or is there a matching lower bound?

2. What is the trade-off for turning an NFA into an equivalent rev-stl-det-ORWW-automaton? Based on Theorem 7 we have an exponential upper bound. Can this bound be improved, or is there a matching lower bound?
3. In the unary case we have a quadratic trade-off for turning an NFA into an equivalent rev-stl-det-ORWW-automaton (Theorem 12). Is there a matching lower bound?

References

1. Bennett, C.H.: Logical reversibiliy of computation. IBM J. Res. Dev. **17**, 525–532 (1973)
2. Chrobak, M.: Finite automata and unary languages. Theoretical Computer Science **47**, 149–158 (1986)
3. Ellul, K.: Descriptional complexity measures of regular languages. Master's thesis, University of Waterloo (2004)
4. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Restarting automata. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 965, pp. 283–292. Springer, Heidelberg (1995)
5. Kutrib, M., Malcher, A.: Reversible pushdown automata. J. Comput. System Sci. **78**, 1814–1827 (2012)
6. Kutrib, M., Malcher, A., Wendlandt, M.: Reversible queue automata. In: Bensch, S., Freund, R., Otto, F., (eds.) Proc. Sixth Workshop on Non-Classical Models of Automata and Applications (NCMA 2014), books@ocg.at, Band 304, pp. 163–178. Oesterreichische Computer Gesellschaft, Wien (2014)
7. Kwee, K., Otto, F.: On some decision problems for stateless deterministic ordered restarting automata. In: Shallit, J., Okhotin, A. (eds.) DCFS 2015. LNCS, vol. 9118, pp. 165–176. Springer, Heidelberg (2015)
8. Landau, E.: Über die Maximalordnung der Permutationen gegebenen Grades. Archiv. der Math. und Phys. **3**, 92–103 (1903)
9. Landau, E.: Handbuch von der Lehre der Verteilung der Primzahlen, vol. I. Teubner, Leipzig (1909)
10. Mráz, F., Otto, F.: Ordered restarting automata for picture languages. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 431–442. Springer, Heidelberg (2014)
11. 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, Heidelberg (2006)
12. Otto, F.: On the descriptional complexity of deterministic ordered restarting automata. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) DCFS 2014. LNCS, vol. 8614, pp. 318–329. Springer, Heidelberg (2014)
13. Pin, J.-E.: On reversible automata. In: Simon, I. (ed.) LATIN 1992. LNCS, vol. 583, pp. 401–416. Springer, Heidelberg (1992)
14. Průša, D.: Weight-reducing Hennie machines and their descriptional complexity. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 553–564. Springer, Heidelberg (2014)
15. Szalay, M.: On the maximal order in S_n and S_n^*. Acta Arithmetica **37**, 321–331 (1980)