Ordered Restarting Automata

Dissertation

Kent Kwee

epubli

2018

This work has been accepted by the Faculty of Electrical Engineering / Computer Science of the University of Kassel as a thesis for acquiring the academic degree of Doktor der Naturwissenschaften (Dr. rer. Nat.).

Supervisor:	Prof. Dr. Friedrich Otto
Co-Supervisor:	Prof. Dr. Martin Kutrib
Defense Day:	June 8, 2018

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at `http://dnb.dnb.de`.

Kent Kwee

Ordered Restarting Automata

ISBN: 978-3-746741-27-7

Zugl: Kassel, Univ., Diss., 2018
Published online on the document server of the University of Kassel at `https://kobra.bibliothek.uni-kassel.de`.

Meinen Eltern gewidmet

Acknowledgements

At this point I would like to express my special thanks to the following persons, without whose help the completion of this PhD thesis would never have been possible.

First of all, I would like to thank Prof. Dr. Friedrich Otto, my doctoral supervisor, for the support of this work, the friendly help and the manifold ideas which gave me a critical approach to this topic. He has also exerted a significant influence on the thesis, both through careful review of the thesis and as co-author of several publications. I will always remember the talks as an enriching and constructive exchange. I have always found our dialogues to be encouragement and motivation.

I would like to thank Prof. Dr. Martin Kutrib for his helpful and scientific support as a second referee.

I would also like to thank my scientific colleagues for their constructive and helpful comments at the university, at conferences and also through anonymous reviews. I also thank Angelika Hoffmann-Hesse for taking care of the organizational matters at the university.

Finally, I would like to thank my brother, sister and parents to whom I dedicate this thesis.

Abstract

The following thesis deals with ordered restarting automata. Restarting automata are a theoretical model used in linguistics for the *analysis by reduction*. The ordered restarting automata were introduced in the context of two-dimensional picture languages and form the underlying one-dimensional model. Here we look at different variants of this one-dimensional model and examine issues related to language classes and descriptional complexity. A common feature of all these variants is the fixed window size of 3, and that the middle character is replaced by a smaller character in a rewrite step. First of all, we examine the models in which the rewriting process is always connected to a restart. Starting from the deterministic model with states we will soon consider the stateless variant, as it also characterizes the regular languages. This gives us a characterization of the regular languages, which is as simple as that by a DFA. Instead of the states we use tape symbols to measure the size of such an automaton. In addition, we are able to present many languages and language operations more concisely. Moreover, starting from a stateless ordered restarting automaton, which may be deterministic or non-deterministic, we specify a general construction of an NFA with exponentially many states that describes the same language and we show that the construction is optimal except for the constant in the exponent. We then use this construction to show that many interesting decision problems for our stateless restarting automata are PSPACE-complete. Finally, we use the idea of this construction to introduce reversibility for our model.

Another interesting variant is the nondeterministic restarting automaton with states. Its language class is quite unusual, since it contains languages that are not even context-sensitive, but on the other hand, it does not even contain the simple linear language $\{a^n b^n \mid n \in \mathbb{N}\}$. Additionally, we prove a pumping lemma that allows us to decide emptiness and finiteness.

Finally, we consider the variants with separate rewrite and restart operations. While restarting automata with nondeterminism and states can express all context-free languages, we end up with regular languages, whenever we do not allow states or nondeterminism. We were able to show that all interesting decision problems except the word problem are undecidable in this setting.

Zusammenfassung

In der folgenden Arbeit geht es um geordnete Restartautomaten. Restartautomaten sind ein theoretisches Modell, das in der Linguistik bei der *Analyse durch Reduktion* Verwendung findet. Die geordneten Restartautomaten wurden im Zusammenhang mit zweidimensionalen Bildsprachen eingeführt und bilden das zugrundeliegende eindimensionale Modell. Von diesem eindimensionalen Modell betrachten wir verschiedene Varianten und untersuchen und vergleichen hauptsächlich Sprachklassen und Beschreibungskomplexität. Eine Gemeinsamkeit all dieser Varianten ist die feste Fenstergröße von 3, und dass beim Schreiben das mittlere Zeichen durch ein kleineres Zeichen ersetzt wird. Wir untersuchen zunächst die Modelle, bei der der Schreibvorgang immer mit einem Restart verbunden ist. Ausgehend vom deterministischen Modell mit Zuständen betrachten wir recht bald die zustandslose Variante, da diese ebenso die regulären Sprachen charakterisiert. Damit haben wir eine Charakterisierung der regulären Sprachen, die ähnlich einfach ist wie die durch DFAs. Statt der Zustände nutzen wir Bandsymbole. Dennoch können wir viele Sprachen und auch Sprachoperationen deutlich prägnanter darstellen. Zudem geben wir ausgehend von einem zustandslosen geordneten Restartautomaten, der deterministsch oder nicht-deterministisch sein darf, eine allgemeine Konstruktion für einen NFA mit exponentiell vielen Zuständen an, der dieselbe Sprache beschreibt, und zeigen, dass diese Konstruktion bis auf die Konstante im Exponenten optimal ist. Diese Konstruktion verwenden wir dann, um zu zeigen, dass viele interessante Entscheidungsprobleme für unsere zustandslosen Restartautomaten PSPACE-vollständig sind. Die Idee dieser Konstruktion nutzen wir schließlich, um Reversibilität für unser Modell einzuführen.

Eine weitere interessante Variante stellen schließlich die nicht-deterministischen geordneten Restartautomaten mit Zuständen dar. Deren Sprachklasse ist recht ungewöhnlich, da sie Sprachen enthält, die nicht einmal wachsend kontext-senstiv sind, aber andererseits enthält sie nicht einmal die einfache lineare Sprache $\{a^nb^n \mid n \in \mathbb{N}\}$. Wir leiten ein Pumping-Lemma her, das es uns erlaubt Leerheit und Endlichkeit für ORWW-Automaten zu entscheiden.

Schließlich betrachten wir noch Varianten mit getrennter Rewrite- und

Restartoperation. Während die Automaten mit Nichtdeterminismus und Zuständen alle kontextfreien Sprachen darstellen können, landen wir bei einer Einschränkung, sei es bei den Zuständen oder dem Determinismus, wieder bei den regulären Sprachen. Wir zeigen, dass im ersten Fall alle interessanten Entscheidungsprobleme, außer natürlich dem Wortproblem, unentscheidbar sind.

Contents

Chapter 1

Introduction

1.1 Motivation and Objectives

In linguistics people use the technique *analysis by reduction* in order to analyze languages with free word order.

The restarting automaton models this technique and has been introduced by Jančar et al in 1995 [17].

Since then many variants have been introduced and examined. One of them is the deterministic ordered restarting automaton which was introduced in the setting of two-dimensional picture languages [37].

As not much was known about the underlying one-dimensional model we decided to investigate it to gather a better overall understanding about this model. It turns out that the underlying model is interesting in itself, which is why we investigate it independently of its actual origin.

Remark 1.1.1. *The term "ordered" is also used in the context of the finite state automaton, where the ordering refers to the states [41], but in the context of restarting automata the ordering refers to the tape alphabet.*

1.2 Structure of this Thesis

In the first part we investigate the descriptional complexity of the deterministic ordered restarting automata concerning the representation of languages and realization of language operations. As deterministic ordered restarting automata with states can easily be converted to corresponding automata without states we settle for investigating stateless deterministic ordered restarting automata. Since it does not make sense to use states as a complexity measure for stateless ordered restarting automata we use the number of tape symbols as a measure. As the deterministic stateless ordered restarting automata exactly characterize the regular languages, we compare the complexity of language operations with the corresponding complexity for deterministic finite state automata (DFA). It turns out that the complexity of language operations for our restarting automaton is lower than or the same as the corresponding complexity for a DFA. The main reason for that is that we can simulate a DFA by a stateless deterministic ordered restarting automaton (stl-det-ORWW) of the same size. Additionally, operations like reversal can be realized exponentially more succinctly.

The main result is the simulation of a stl-det-ORWW-automaton that uses n tape symbols by a nondeterministic finite state automaton (NFA) of exponential size, that is $2^{\mathcal{O}(n)}$ states. The simulation can also be applied to its nondeterministic counterpart, the stl-ORWW-automaton. This not only represents a significant improvement of the known bound, but is also a new result in the nondeterministic case. It turns out that this bound is sharp. With the help of this simulation we show that for stl-ORWW-automata decision problems like emptiness, finiteness and equivalence are all PSPACE-complete.

We conclude the chapter by investigating how reversibility can be introduced for deterministic ordered restarting automata.

In the following chapter we investigate the ORWW-automata, the nondeterministic ordered restarting automata with states. At first we look at some examples and see that we can express some complicated languages. Using a similar idea as in the previous chapter we derive a true pumping lemma for these automata. We then use it to show that the emptiness and finiteness problems for these automata are decidable. We also show that the simple language $\{a^nb^n \mid n \geq 1\}$ is not accepted by any ORWW-automaton. Finally, we look at how we can realize language operations and find out that the ORWW-automata describe an abstract family of languages, that is not closed under reversal and complement. It can be arranged between the regular and the context sensitive languages.

Lastly, motivated by the restrictions of not being able to express the simple language from above we look at ORRWW-automata that do not have that restriction. As before we look at restricted variants first. The det-ORRWW-automaton can be simulated by a det-ORWW-automaton. Thus, the det-ORRWW-automata describe the regular languages.

Now, the stl-ORRWW-automata remain to be considered. We find out that their computations can be normalized which finally allows us to apply the Myhill-Nerode-Theorem. Thus, also these automata describe the regular languages. Finally, we look at the ORRWW-automata. They can describe all context-free languages and many more interesting languages. However, this has the consequence that all interesting decision problems are undecidable for ORRWW-automata.

In the end, we finish our thesis by reviewing the properties of the ORWW and ORRWW automata. We mention open problems that are worth investigating further.

For the reader's convenience, the chapters are designed in such a way that they are not strongly interdependent. They can be read largely independently of each other.

1.3 Publications

A part of this thesis has already been published or submitted for publication. The publications are listed here:

The work "On the Descriptional Complexity of Stateless Deterministic Ordered Restarting Automata" is published in "Information and Computation" [36].

[36] F. Otto and K. Kwee. On the descriptional complexity of stateless deterministic ordered restarting automata. *Information and Computation*, `https://doi.org/10.1016/j.ic.2017.09.006`, 2017

It is based on the conference papers of DCFS 2014 [34], DCFS 2015 [24] and DLT 2015 [35].

[34] F. Otto. On the descriptional complexity of deterministic ordered restarting automata. In H. Jürgensen, J. Karhumäki, and A. Okhotin, editors, *DCFS 2014, Proc.*, *LNCS 8614*, pages 318–329. Springer, Heidelberg, 2014

[24] K. Kwee and F. Otto. On some decision problems for stateless deterministic ordered restarting automata. In J. Shallit and A. Okhotin, editors, *DCFS 2015, Proc.*, *LNCS 9118*, pages 165–176. Springer, Heidelberg, 2015

[35] F. Otto and K. Kwee. Deterministic ordered restarting automata that compute functions. In I. Potapov, editor, *DLT 2015, Proc.*, *LNCS 9168*, pages 401–412. Springer, Heidelberg, 2015

The part about reversible ordered restarting automata was presented at the conference "Reversible Computation 2015" and published in the corresponding proceedings [38].

[38] F. Otto, M. Wendlandt, and K. Kwee. Reversible ordered restarting automata. In J. Krevine and J.-B.. Stefani, editors, *RC 2015, Proc.*, *LNCS 9138*, pages 60–75. Springer, Heidelberg, 2015

Additionally, there are some publications about nondeterministic ordered restarting automata with states.

[26] K. Kwee and F. Otto. On the effects of nondeterminism on ordered restarting automata. In R.M. et al. Freivalds, editor, *SOFSEM 2016, Proc.*, *LNCS 9587*, pages 369–380. Springer, Heidelberg, 2016

[27] K. Kwee and F. Otto. A Pumping Lemma for Ordered Restarting Automata. In G. Pighizzini and C. Câmpeanu, editors, *DCFS 2017, Proc.*, *LNCS 10316*, pages 226 – 237. Springer, Heidelberg, 2017

Finally, there is a publication about ORRWW-automata that was presented at DLT 2016.

[25] K. Kwee and F. Otto. On Ordered RRWW-Automata. In Srečko Brlek and Christophe Reutenauer, editors, *Developments in Language Theory*, *LNCS 9840*, pages 268–279, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg

Chapter 2

Background Theory

2.1 Introduction

In this chapter we give a short overview of basic terms and notation we use in this thesis. It does not claim to be complete. More detailed information can be looked up in classical formal language theory books like [16]. Additionally, we give a very brief introduction for generic restarting automata.

2.2 Basic Formal Language Theory

In computer science *formal languages* are sets of strings of symbols. That is, a formal language L is a subset of Σ^*, where the alphabet Σ denotes the set of symbols. The empty word is denoted by the symbol λ.

There are several ways to describe a formal language L. A common and essential criterion for the specification is the finite representation of the usually infinite number of words. One method for the specification of a formal language would be to specify a set of properties. The set of words obeying these properties would then be the defined language.

Another way is using formal grammars to describe the syntactic structure of a language.

A *formal grammar* is a set of production rules for strings. Formally, it can be described by a 4-tuple (N, Σ, P, S) where

- N is a finite set of nonterminal symbols,

- Σ is a finite alphabet,
- P is a set of production rules of the form $(\Sigma \cup N)^* N (\Sigma \cup N)^* \to (\Sigma \cup N)^*$, and
- $S \in N$ is the start symbol.

Depending on the restrictions on the kind of rules of the grammar the language is classified according to the famous Chomsky Hierarchy. In addition to the production rules we specify, we allow the rule $S \to \lambda$ to include the empty word in the language defined.

- Type 1 grammars have production rules of the form $\alpha A \beta \to \alpha \gamma \beta$ with $\alpha, \beta \in (\Sigma \cup N)^*$, $A \in N$ and $\gamma \in (\Sigma \cup N \setminus \{S\})^+$. They describe the context-sensitive languages (CSL). The language class CSL is also often described by monotone grammars. They have production rules $P \subseteq (N \cup T)^* \times (N \cup T)^*$, such that for $(w_1 \to w_2) \in P$, $|w_1| \leq |w_2|$. If w_1 is strictly shorter than w_2, that is $|w_1| < |w_2|$, we have a strictly monotone grammar. They describe the growing-context-sensitive languages (GCSL).
- Type 2 grammars have production rules of the form $N \to (\Sigma \cup N \setminus \{S\})^+$. They describe the context-free languages (CFL). If there is at most one nonterminal symbol at the right hand side for every production rule, we have a linear grammar which describe the linear languages (LIN).
- Type 3 grammars have production rules of the form $N \to (\Sigma \cup \Sigma N)$. They describe the regular languages (REG).

An overview can be seen in the following table where we show the grammar type and the characterization with a type of automaton.

Grammar Name	Language	Automaton Model
0. Unrestricted	Rec. enumerable	Turing Machine
1. Context-sensitive	Context-sensitive	Linear-bounded automaton
2. Context-free	Context-free	Pushdown automaton
3. Regular	Regular	Finite state automaton

The corresponding automaton model for the linear languages are one-turn pushdown automata. They have the property that they cannot execute push operations after executing a pop operation (see [13]).

In this work, we mostly use the approach to specify some device or algorithm for recognizing valid words. That is why we will mostly focus on the characterization of languages through automata.

In this context, we investigate in particular closure properties under language operations. But first of all, we present the most common language operations.

Language Operations

The most common language operations are enlisted in the following table, where L_1 and L_2 are formal languages over Σ^* and f is a morphism $f : \Sigma^* \to \Gamma^*$.

Operation	Notation and Definition
Union	$L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}$
Intersection	$L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2\}$
Concatenation	$L_1 \cdot L_2 = \{w_1 \cdot w_2 \in \Sigma^* \mid w_1 \in L_1, w_2 \in L_2\}$
Complement	$\overline{L_1} = L_1^c = \{w \in \Sigma^* \mid w \notin L_1\}$
Kleene Star	$L_1^* = \{\lambda\} \cup L_1 \cdot L_1^*$
Reversal	$L_1^R = \{w^R \mid w \in L_1\}$
Morphism	$f(L_1) = \{f(w) \mid w \in L_1\}$
Inverse Morphism	$f^{-1}(L_1) = \{w \in \Sigma^* \mid f(w) \in L_1\}$

In order to present some more language operations we discuss some string operations.

Let L be a language and Σ be its alphabet. A *substitution* is a mapping f that maps each letter $a \in \Sigma$ to a language $L_a \subseteq \Gamma^*$ where Γ is an alphabet. The mapping f is then extended canonically to a mapping on strings:

$$f(\lambda) = \{\lambda\}, f(aw) = f(a) \cdot f(w),$$

where $a \in \Sigma, w \in \Sigma^*$.

In short, a substitution is a letter-to-language mapping.

A morphism is a special case of a substitution. In that case we match letters to languages that only consist of a single word. We can then directly match a letter to this string and we have a letter-to-string mapping.

Let L be a language and Σ be its alphabet. A *morphism* is a mapping f that maps each letter $a \in \Sigma$ to a string $w_a \in \Gamma^*$. The mapping is then extended canonically to a mapping on strings.

2.3 Restarting Automata

As mentioned before, the restarting automata were introduced by Jančar, Mráz, Plátek Vogel in 1995 [17] to model the analysis by reduction.

This section is strongly influenced by the thesis of Hartmut Messerschmidt [30].

While the initial model was very restricted, many variations were introduced later.

We now give the definition for the most general model that we will work with, the RRWW-automaton.

Definition 2.3.1. *[30] A restarting automaton, RRWW-automaton for short, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, k, \delta)$, where Q is the finite set of states, Σ is the finite input alphabet, Γ is the finite tape alphabet containing Σ, the symbols $\rhd, \lhd$ are markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the read/write window, and*

$$\delta : Q \times PC^{(k)} \to 2^{Q \times (\{\mathsf{MVR}\} \cup PC^{\leq (k-1)})} \cup \{\mathsf{Restart}, \mathsf{Accept}\},$$

where $PC^{(k)}$ is the set of possible window contents of length k

$$PC^{(k)} = (\{\rhd\} \cdot \Gamma^{k-1}) \cup \Gamma^k \cup (\Gamma^{\leq k-1} \cdot \{\lhd\}) \cup (\{\rhd\} \cdot \Gamma^{k-2} \cdot \{\lhd\})$$

and

$$PC^{\leq (k-1)} = \bigcup_{i=1}^{k-1} PC^{(i)} \cup \{\lambda\}.$$

The transition relation describes four different types of transition steps:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in PC^{(k)}, u \neq \lhd$. If M is in state q and sees the string u in its read/write window, then this move-right step causes M to shift the

read/write window one position to the right and to enter state q'. However, if the content u of the read/write window is only the symbol $\lhd$, then no shift to the right is possible.

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

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

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

If no instruction with a left-hand side (q, u) exists, then the automaton halts and rejects, when it reaches state q while it sees tape content u in its window. A halting configuration is either an accepting or a rejecting configuration. A restarting automaton performs exactly one rewrite operation between two restart steps. In general an automaton M is nondeterministic which means that for a left-hand side (q, u) there can be more than one instruction. If for every left-hand side (q, u) there exists at most one instruction, then M is called deterministic and the prefix det- is used to describe the class of deterministic restarting automata. A configuration of a restarting automaton is given by a word $\alpha q \beta$ where $q \in Q$ is the current state and $\alpha\beta \in \Gamma^*$ is the current tape content including the border markers, with the read/write window scanning the first k letters of the word β. A configuration $q_0 \rhd \omega \lhd$ with $\omega \in \Gamma^*$ is called a restarting configuration, and if $\omega \in \Sigma^*$ then it is called an initial configuration.

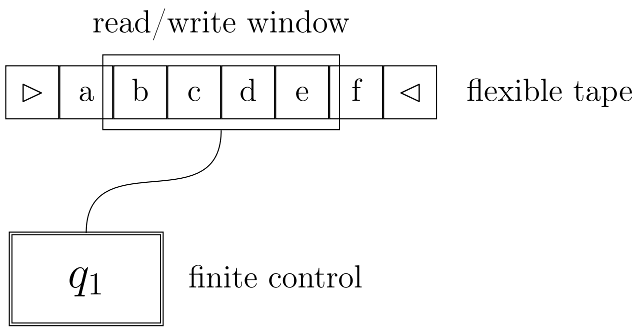


Figure 2.1: A schematic representation of a restarting automaton in the configuration $\triangleright a q_1 bcdef \triangleleft$

$\alpha q \beta$ is called an accepting configuration if there exist $\beta_1, \beta_2 \in \Gamma^*$ such that $\beta = \beta_1\beta_2$ and $\delta(q, \beta_1) = \mathsf{Accept}$.

The transition relation transforms a configuration $\alpha_0 q_0 \beta_0$ into a successor configuration $\alpha_1 q_1 \beta_1$. This is denoted as $\alpha_0 q_0 \beta_0 \vdash \alpha_1 q_1 \beta_1$ and the type of the transition relation is given as an index. $\vdash^*_{MVR}$ describes a sequence of MVR steps.

A schematic representation of a restarting automaton is shown in Figure 2.1.

Definition 2.3.2. *The language accepted by a restarting automaton M is called $L(M)$. It consists of all words $w \in \Sigma^*$ for which there is an accepting computation of M starting from the initial configuration $q_0 \triangleright w \triangleleft$. The complete language, or characteristic language, of M, $L_C(M)$ for short, consists of all words $w \in \Gamma^*$ which are accepted by M. That is,*

$$L(M) = \{w \in \Sigma^* \mid q_0 \triangleright w \triangleleft \vdash^*_M \mathsf{Accept}\}$$

and

$$L_C(M) = \{w \in \Gamma^* \mid q_0 \triangleright w \triangleleft \vdash^*_M \mathsf{Accept}\}.$$

The class name RRWW indicates which of the many restrictions of the general restarting automaton we are using. It is an acronym and consists of two parts. The first part RR refers to the restriction on the movement of the read/write window. In general the first part can be

RR, which means *no restriction*, and

R, which means that we have to execute the restart operation directly after executing a rewrite.

The second part of the acronym refers to restrictions on the rewrite instructions, where

WW denotes no restriction,

W means that we cannot use auxiliary symbols, and

$\boldsymbol{\lambda}$ means that we are only allowed to delete symbols.

To sum it up, a restarting automaton consists of a finite state control, a flexible tape containing the input word with border markers at the left and right end of the word, and a read/write window of a fixed size.

Initially, the window is at the left border marker and the automaton is in its initial state. This is the initial configuration. By using move-right steps the automaton moves the window over the tape depending on the window content and its current state. Additionally, it can execute Rewrite, Restart, and Accept operations. Not considering the move steps the Rewrite- and Restart-operations have to alternate starting with the rewrite operation.

When executing a rewrite operation the window content u is replaced by the strictly shorter word v and the window is placed to the right of v. When executing a restart operation the window is placed to the initial position at the left border marker.

A computation consists of several phases. A *cycle* starts in the initial configuration, also called restarting configuration, and ends with a rewrite operation. The accepting *tail* is the phase that starts in a restarting configuration and ends with an Accept operation.

With this definition in mind an RWW-automaton is a restarting automaton that can use auxiliary symbols but has to restart directly after executing a rewrite.

Otto has shown in [33] that for every RRWW-automaton M there exists an RRWW-automaton M' that only performs restart operations at the right sentinel which accepts the same language. This allows us to split one cycle into several parts.

The first part starts in the restarting configuration and contains only MVR operations. The next part only consists of the rewrite operation. It is followed by a part with additional MVR steps until the right border marker is reached which is immediately followed by a restart operation.

The accepting tail only consists of MVR operations which ends with an Accept operation.

Surely, the accepting tail could also contain a rewrite operation, but as we accept anyways, we can simply omit the rewrite operation.

The splitting into several parts allows us the introduction of meta-instructions.

The MVR phase can be described by a regular expression because the automaton behaves like a finite state machine. The rewrite phase can be described by writing down the pair of strings consisting of the string to be replaced and the string that replaces it. The final MVR phase until the restart can again be described by a regular expression.

Definition 2.3.3. *A meta-instruction for an RRWW-automaton M is either of the form $(E_1, u \to v, E_2)$ or (E_1, Accept), where E_1, E_2 are regular expressions, $u, v \in \Gamma^*$. We can apply a rule of a form $(E_1, u \to v, E_2)$ if the word w on the tape can be written as a factorization $w = sut$ with $s, u, t \in \Gamma^*$ such that E_1 matches $\rhd s$ and E_2 matches $t\lhd$. In that case we can replace the word $w = sut$ by the word svt. A rule of the form (E_1, Accept) describes an accepting tail in a computation and it can be applied if E_1 matches $\rhd w\lhd$. All rules and the factorization are chosen nondeterministically. If no rule can be applied the automaton gets stuck and we reject the input.*

Meta-instructions for RWW-automata look a bit different as there are no MVR operations after the rewrite operations. Thus, we omit the last regular expression. A meta-instruction for an RWW-automaton M is either of the form $(E, u \to v)$ or (E, Accept), where E is a regular expressions and $u, v \in \Gamma^*$. We can apply a rule of a form $(E, u \to v)$ if the word w on the tape can be written as a factorization $w = sut$ with $s, u, t \in \Gamma^*$ such that E matches $\rhd s$. In that case we can replace the word $w = sut$ by the word svt. A rule of the form (E, Accept) describes an accepting tail in a computation and it can be applied if E matches $\rhd w\lhd$.

In addition to the classical language classes of the Chomsky Hierarchy, we also look at some typical languages that appear in the context of restarting automata.

First of all, there are the Church-Rosser languages. Classically, they are defined by a string rewriting system.

Definition 2.3.4. *Let Σ be a finite alphabet. A* string-rewriting system *R on Σ is a subset of $\Sigma^* \times \Sigma^*$. It induces the following reduction relations on Σ^*.*

Starting with the single-step reduction relation

$$\rightarrow_R = \{(ulv, urv) \mid u, v \in \Sigma^*, (l \rightarrow r) \in R\}$$

we get the reflexive and transitive closure $\rightarrow_R^*$ of $\rightarrow_R$.

If we have a string $u \in \Sigma^*$ and there is no string $v \in \Sigma^*$ such that $u \rightarrow_R v$ holds, u is called irreducible. IRR(R) *denotes the set of all irreducible strings. The string-rewriting system R is called* length-reducing *if $|l| > |r|$ holds for each rule $(l \rightarrow r) \in R$. It is called* confluent *if, for all $u, v, w \in \Sigma^*$ with $u \rightarrow_R^* v$ and $u \rightarrow_R^* w$, v and w have a common descendant, that is, there exists a word $s \in \Sigma^*$ such that $v \rightarrow_R^* s$ and $w \rightarrow_R^* s$.*

Now, we can define what a Church-Rosser language is.

Definition 2.3.5. *A language $L \subseteq \Sigma^*$ is a Church-Rosser language if there exist a working alphabet $\Gamma \supset \Sigma$, a finite length-reducing confluent string rewriting system R on Γ, two strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap \mathrm{IRR}(R)$, and a symbol $Y \in (\Gamma \setminus \Sigma)^* \cap \mathrm{IRR}$ such that for all $w \in \Sigma^* : t_1 w t_2 \rightarrow_R^* Y$ if and only if $w \in L$.*

The Church-Rosser languages (CRL) are often seen together with the growing context-sensitive languages, or GCSL for short.

We are going to integrate the language classes described by our ordered restarting automata and their variations into the hierarchy of common language classes.

The common acronyms for the different language classes are the following:

REG	Regular language
DLIN	Deterministic linear language
LIN	Linear language
DCFL	Deterministic context-free language
CFL	Context-free language
CRL	Church-Rosser language

$$\begin{tikzcd}
 & \mathsf{CSL} \\
 & \mathsf{GCSL} \arrow[u] \\
\mathsf{CRL} \arrow[ur] & \mathsf{CFL} \arrow[u] \\
\mathsf{DCFL} \arrow[u] \arrow[ur] & \mathsf{LIN} \arrow[u] \\
\mathsf{DLIN} \arrow[u] \arrow[ur] & \\
 & \mathsf{REG} \arrow[ul] \arrow[uu]
\end{tikzcd}$$

Figure 2.2: The hierarchy of common language classes. The arrow indicates a proper inclusion.

GCSL	Growing context-sensitive language
CSL	Context-sensitive language

The hierarchy regarding inclusion is shown in figure 2.2.

Chapter 3

Ordered Restarting Automata

In this chapter we introduce the ordered restarting automaton and some variants of it. It differs from the classical restarting automaton we introduced in the previous chapter in that it cannot shorten the input, as it can only replace one symbol with a smaller symbol with respect to a given partial ordering. This means that each of these automata can be simulated by a linearly bounded automaton and their language class is included in the context-sensitive languages.

In particular, we examine the language class described by each type of automaton and investigate how concise its presentation is. First of all, we start with a generic definition of the ordered restarting automaton, or ORWW-automaton for short. This model is nondeterministic and has states. After that we look at certain restrictions of the ordered restarting automaton. Initially, we have a look at the most restricted variant, the stateless deterministic ordered restarting automaton. It turns out that we can simulate a deterministic ordering restarting automaton with states, a det-ORWW-automaton, by a deterministic ordering restarting automaton without states, a stl-det-ORWW-automaton. For this reason, we shift our focus to the stateless ordered restarting automaton in order to investigate the descriptional complexity of language operations in Section 3.3.

We will show that for a stl-det-ORWW automaton M with a tape alphabet of size n we can construct an NFA A with $2^{\mathcal{O}(n)}$ states that accepts the same language. This is the main result of this chapter.

Before we describe the construction and prove its correctness, we briefly have a look at the nondeterministic stateless ordered restarting automaton, that is, the stl-ORWW-automaton.

It turns out that for nondeterministic ordered restarting automata the same construction can be used as for deterministic ones, which is why we directly describe the more general case in section 3.5.

As a preparation for the final construction of the NFA we have a closer look at the properties of valid computations and how they can be manipulated.

We then use this construction to show that many decisions problems for stl-ORWW-automata are PSPACE-complete.

We supplement the chapter by exploring how stl-det-ORWW-automata can be extended to be reversible.

3.1 Definition

An *ORWW-automaton* is a nondeterministic one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, where Q is a finite set of states containing the initial state q_0, 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,

$$\delta : (Q \times ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\}) \cup \{\rhd\lhd\})) \to 2^{(Q\times\{\mathsf{MVR}\})\cup\Gamma} \cup \{\mathsf{Accept}\}$$

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

(1) A *move-right step* has the form $(q', \mathsf{MVR}) \in \delta(q, a_1a_2a_3)$, where $q, q' \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, and $a_2, a_3 \in \Gamma$. It causes M to shift the window one position to the right and to change from state q into state q'. Observe that no move-right step is possible, if the window contains the symbol $\lhd$.

(2) A *rewrite/restart step* has the form $b \in \delta(q, a_1a_2a_3)$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2, b \in \Gamma$, 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 moved back to the left end of the tape, and M reenters its initial state q_0.

(3) An *accept step* has the form $\delta(q, a_1a_2a_3) = \mathsf{Accept}$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$. It causes M to halt and accept. In addition, we allow an accept step of the form $\delta(q_0, \rhd\lhd) = \mathsf{Accept}$.

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

If $|\delta(q, u)| \leq 1$ for all q and u, then M is a *deterministic ORWW-automaton* (det-ORWW-automaton). We use the partial transition function

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

to emphasize determinism.

If $Q = \{q_0\}$, that is, if the initial state is the only state of M, then we call M a *stateless ORWW-automaton* (stl-ORWW-automaton) or a *stateless deterministic ORWW-automaton* (stl-det-ORWW-automaton), as in this case the state is actually not needed. Accordingly, for stateless ORWW-automata, we will drop the components that refer to states to simplify the notation. In that case we formally use the 6-tuple $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, and the transition function

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

for a stl-det-ORWW-automaton and the transition function

$$\delta : ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \cup \{\rhd\lhd\} \to 2^{\{\mathsf{MVR}\} \cup \Gamma} \cup \{\mathsf{Accept}\}$$

for a the stl-ORWW-automaton.

We note that in the case of nondeterministic automata, we do not allow other operations if we can execute the Accept operation. The simple reason for this is that we would never perform the other operations as it does not make sense not to accept when we can.

The same terms and notations that exist for restarting automata are naturally transferred to the ordered restarting automata.

A *configuration* of an ORWW-automaton M is a word $\alpha q \beta$, where $q \in Q$ is the current state, $|\beta| \geq 3$, 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 $q_0 \rhd \lhd$. A *restarting configuration* has the form $q_0 \rhd w \lhd$; if $w \in \Sigma^*$, then $q_0 \rhd w \lhd$ is also called an *initial configuration.* Further, we use Accept to denote the *accepting configurations*, which are those configurations that M reaches by an accept step. An example for a configuration and its graphical representation can be seen in Figure 3.1.

For stateless automata this representation is not so practical because we would have to give the constant state a name. For that reason we introduce an additional notation for configurations of stateless ordered restarting automata. 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.*

As an alternative we also allow to simply underline the three characters of the window content, which roughly corresponds to the graphical representation of a configuration.

Any computation of an 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. It can be seen as the *rewrite relation* that is realized by M on the set of restarting configurations.

An input word $w \in \Sigma^*$ is accepted by M if there is a computation of M which starts with the initial configuration $q_0 \rhd w \lhd$ and ends with an accept step. The language consisting of all words that are accepted by M is denoted by $L(M)$.

As each cycle ends with a rewrite operation, which replaces a symbol a by a

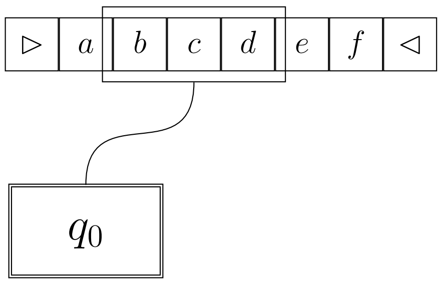


Figure 3.1: An ORWW-automaton in the configuration $\triangleright a q_0 bcdef \triangleleft$

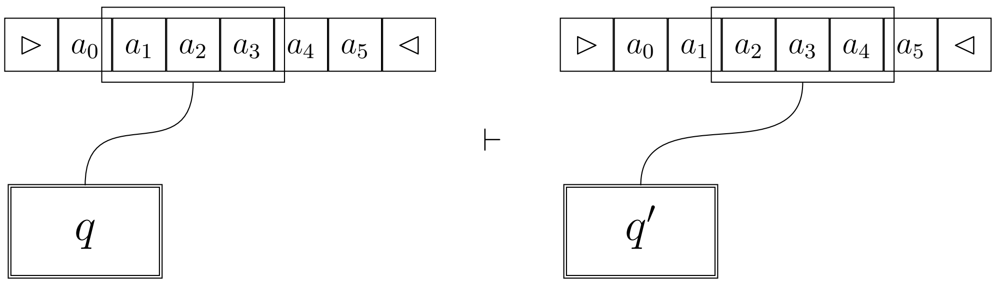
symbol b that is strictly smaller than a with respect to the given ordering $>$, each computation of M on an input of length n consists of at most $(|\Gamma|-1) \cdot n$ cycles. Thus, M can be simulated by a nondeterministic single-tape Turing machine in time $\mathcal{O}(n^2)$.

We now illustrate the single-step relation using the above-mentioned possible operations. Given an ORWW-automaton M in the configuration $\triangleright a_0 q a_1 a_2 a_3 a_4 a_5 \triangleleft$ there could be three different kinds of operations.

(a) If we execute the MVR operation $\delta(q, a_1a_2a_3) \to (q', \mathsf{MVR})$, we would have the following transition between configurations:

$$\triangleright a_0 q a_1 a_2 a_3 a_4 a_5 \triangleleft \vdash_M \triangleright a_0 a_1 q' a_2 a_3 a_4 a_5 \triangleleft$$

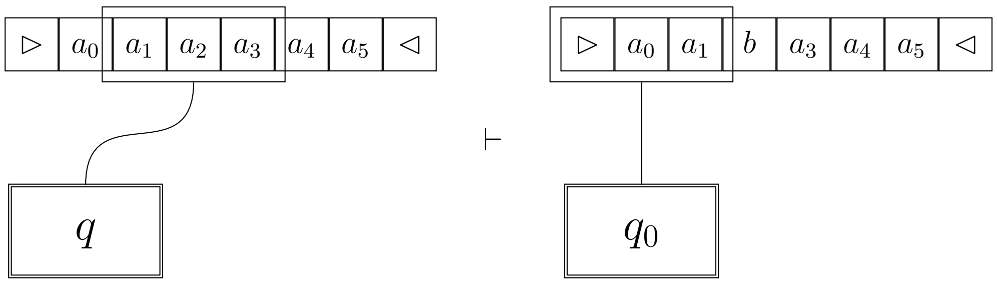
which can be represented graphically in the following way:



(b) Then we could execute the Rewrite operation $\delta(q, a_1a_2a_3) \to b$, which would lead to the following transition between configurations:

$$\triangleright a_0 q a_1 a_2 a_3 a_4 a_5 \triangleleft \vdash_M q_0 \triangleright a_0 a_1 b a_3 a_4 a_5 \triangleleft$$

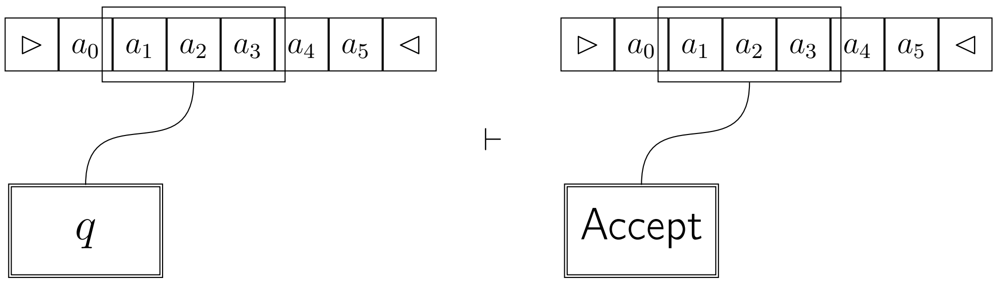
which can be represented graphically as follows:



(c) Finally, we could execute the Accept operation $\delta(q, a_1a_2a_3) \to \mathsf{Accept}$, which would look like follows

$$\rhd a_0 q a_1 a_2 a_3 a_4 a_5 \lhd \vdash_M \mathsf{Accept}$$

with the following graphical representation:



We will see in the following sections that 3 of the 4 variants presented characterize the regular languages. Due to the special characteristics of the most generic variant, that is the ORWW-automaton, we dedicate a separate chapter to this one.

We will first deal with the most restricted variant, the stateless deterministic ordered restarting automaton (stl-det-ORWW-automaton).

3.2 Deterministic Ordered Restarting Automata

In this section we deal with stateless deterministic ordered restarting automata, the simplest of our variants. As we have already given the definition, we start with an example to illustrate the way in which a stateless det-ORWW-automaton works.

Example 3.2.1. *Let $n \geq 2$ be a fixed integer, and let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be defined by taking $\Sigma = \{a, b\}$ and $\Gamma = \Sigma \cup \{\, a_i, b_i, x_i \mid 1 \leq i \leq n-1 \,\}$, by* *choosing the ordering $>$ such that $a > a_i > x_j$ and $b > b_i > x_j$ hold for all $1 \leq i, j \leq n-1$, and by defining the transition function δ in such a way that M proceeds as follows: On input $w = w_1 w_2 \dots w_m$, $w_1, w_2, \dots, w_m \in \Sigma$, M numbers the first $n-1$ letters of w from left to right, by replacing $w_i = a$ (b) by a_i (b_i) for $i = 1, 2, \dots, n-1$. If $w_n \neq a$, then the computation fails, but if $w_n = a$, then M continues by replacing the last $n-1$ letters of w from right to left using the letters x_1 to x_{n-1}. If the n-th last letter is b or some b_i, then M accepts, otherwise the computation fails again.*

Then $L(M) = \{ w \in \{a, b\}^m \mid m > n, w_n = a, \text{ and } w_{m+1-n} = b \}$. It is well-known (and easily checked) that a DFA for this language needs $\mathcal{O}(2^n)$ states. Observe that M is stateless and that it has an alphabet of size $3n - 1$ only.

First we show that the stateless deterministic ordered restarting automaton can represent all regular languages.

Proposition 3.2.2. [34] *For each DFA $A = (Q, \Sigma, q_0, F, \Delta)$, there exists a stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ such that $L(M) = L(A)$ and $|\Gamma| = |Q| + |\Sigma|$.*

Proof. We take $\Gamma = \Sigma \cup Q$ and define $a > q$ for all $a \in \Sigma$ and all $q \in Q$. On input of a word $w = a_1 a_2 \dots a_n$, where $n \geq 1$ and $a_1, a_2, \dots, a_n \in \Sigma$, the stl-det-ORWW-automaton M simply replaces each letter a_i by the state $\Delta(q_0, a_1 a_2 \dots a_i)$, proceeding from left to right. It accepts if and when the last letter a_n has been replaced by a final state of A. Also, we add the transition $\delta(\rhd\lhd) = \mathsf{Accept}$ if $\lambda \in L(A)$.

In detail, we treat the words of length one as a special case to get a slightly simpler transition function. With this in mind the transition function looks as follows. At first we have the two special cases

$$\begin{aligned} \delta(\rhd\lhd) &= \mathsf{Accept} && \text{if } \lambda \in L(A), \\ \text{and } \delta(\rhd a \lhd) &= \mathsf{Accept} && \text{if } a \in L(A). \end{aligned}$$

Then we have the case for words of length ≥ 2, where $a, b \in \Sigma$ and $p, q, r \in Q$:

$$\delta(\rhd ab) = \Delta(q_0, a),$$

$$\begin{aligned}
\delta(\triangleright pa) &= \mathsf{MVR}, \\
\delta(pab) &= \Delta(p,a), \\
\delta(\triangleright pq) &= \mathsf{MVR}, \\
\delta(pqa) &= \mathsf{MVR}, \\
\delta(pqr) &= \mathsf{MVR}, \\
\delta(pa\triangleleft) &= \Delta(p,a), \\
\delta(pq\triangleleft) &= \mathsf{Accept} && \text{if } q \in F.
\end{aligned}$$

This means that for words of length ≥ 2 there is a corresponding cycle in the ORWW for each transition in the DFA. If a word $w = w_1w_2\dots w_n, n \geq 2$ is accepted by the DFA A by the computation

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_{n-1}} q_{n-1} \xrightarrow{w_n} q_n,$$

the word w is accepted by the ORWW M by the computation

$$\underline{\triangleright w_1w_2}\dots w_n \triangleleft \vdash^c \underline{\triangleright q_1w_2}\dots w_n\triangleleft \vdash^c \underline{\triangleright q_1q_2}w_3\dots w_n\triangleleft \vdash^{c^*} \underline{\triangleright q_1q_2}\dots q_{n-1}w_n\triangleleft$$
$$\vdash^c \underline{\triangleright q_1q_2}\dots q_{n-1}q_n\triangleleft \vdash^* \mathsf{Accept}$$

□

Note, as we have only used restarting configurations in the last computation, underlining the content of the read/write window is not necessary.

With this we have seen that

$$\mathsf{REG} \subseteq \mathcal{L}(\mathsf{stl\text{-}det\text{-}ORWW})$$

holds.

Now the question arises what happens when we add states. Obviously, $\mathcal{L}(\mathsf{stl\text{-}det\text{-}ORWW}) \subseteq \mathcal{L}(\mathsf{det\text{-}ORWW})$ holds as a stl-det-ORWW-automaton is just a special det-ORWW-automaton, namely one with exactly one state.

The only question is whether the inclusion is proper.

It turns out that the stateless det-ORWW-automata are as expressive as the variants with states. This can be deduced as follows.

As the automaton is deterministic we can make the following observation.

Remark 3.2.3. *Let M be a det-ORWW-automaton. As each cycle ends with a rewrite/restart step, which replaces a letter a by a letter b that is strictly smaller than a with respect to the given ordering $>$, we see that each computation of M on an input of length n consists of at most $(|\Gamma|-1) \cdot n$ cycles. Assume now that within a computation, M performs a rewrite/restart step at some tape position $i \geq 3$, that is,*

$$\rhd u_1 u_2 \dots u_{i-3} u_{i-2} qabcv \lhd \vdash_M q_0 \rhd u_1 u_2 \dots u_{i-3} u_{i-2} ab'cv \lhd .$$

As M is deterministic, it must next execute MVR steps until the newly written letter b' appears in its window. Thus, this computation continues with a sequence of $i-2$ MVR steps, that is,

$$q_0 \rhd u_1 u_2 \dots u_{i-3} u_{i-2} ab'cv \lhd \vdash_M^{i-2} \rhd u_1 u_2 \dots u_{i-3} p u_{i-2} ab'cv \lhd$$

for some state p. By combining the rewrite/restart step with these subsequent MVR steps, we obtain an operation that first rewrites the letter in the middle of the window by a smaller letter with respect to the ordering $>$ and that then moves the window simply one position to the left. This implies in particular that a det-ORWW-automaton can be simulated by a deterministic single-tape Turing machine in linear time.

This observation allows us to simulate a det-ORWW-automaton by a stl-det-ORWW-automaton. A similar simulation has already been presented in [34].

Proposition 3.2.4. *Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be a det-ORWW-automaton. Then there exists a stl-det-ORWW-automaton $M' = (\Sigma, \Theta, \rhd, \lhd, \delta', >')$ such that $L(M') = L(M)$ and $|\Theta| = |Q| \cdot |\Gamma|^2 + |\Gamma|$.*

Proof. The idea of the construction is that the stl-det-ORWW-automaton M' stores the actual state of M on its tape. To be more specific, whenever we would execute a move-right step, we first store the state, M would change to, in our current tape field and execute the MVR operation in the next cycle. In this way, the current state is always stored in the first symbol of the read/write window. If the first symbol is the left sentinel $\rhd$, we are obviously in the initial state q_0. Now, we need a partial ordering for our tape alphabet. As we could change to different states from one tape field when the symbol to the right changes, we also store this symbol together with the state in order to define a partial ordering. When simulating the automaton, we do not need that symbol to decide what operation is to be executed. The symbol is only used to ensure that we replace symbols with smaller symbols. Therefore, we use the following tape alphabet

$$\Theta = \Gamma \cup \{\, [q,a,b] \mid q \in Q, a, b \in \Gamma \,\}$$

with the partial ordering

$$a >' [q,a,x] >' [p,a,y] >' b$$

where $a, b, x, y \in \Gamma, p, q \in Q, a > b, x > y$.

Accordingly, we can construct M' from M as follows:

- δ' is defined as follows, where $a, b, c, x, y, z, A \in \Gamma$, $D \in \Gamma \cup \{\lhd\}$, and $p, q, r \in Q$:

  First we treat the empty word and single letter words as a special case:

  $$\delta'(\rhd u \lhd) = \mathsf{Accept} \quad \text{for all } u \in (\Sigma \cup \{\lambda\}) \cap L(M).$$

  Then we have the possible cases for rewrites:

  $$\begin{aligned} \delta'(\rhd ab) &= A && \text{if } \delta(q_0, \rhd ab) = A, \\ \delta'(\rhd [q,a,x] b) &= A && \text{if } \delta(q_0, \rhd ab) = A, \\ \delta'([q,a,x] bD) &= A && \text{if } \delta(q, abD) = A, \\ \delta'([q,a,x][p,b,y] D) &= A && \text{if } \delta(q, abD) = A. \end{aligned}$$

  And here are the possible cases for Accept operations:

  $$\begin{aligned} \delta'(\rhd ab) &= \mathsf{Accept} && \text{if } \delta(q_0, \rhd ab) = \mathsf{Accept}, \\ \delta'(\rhd [q,a,x] b) &= \mathsf{Accept} && \text{if } \delta(q_0, \rhd ab) = \mathsf{Accept}, \\ \delta'([q,a,x] bD) &= \mathsf{Accept} && \text{if } \delta(q, abD) = \mathsf{Accept}, \\ \delta'([q,a,x][p,b,y] D) &= \mathsf{Accept} && \text{if } \delta(q, abD) = \mathsf{Accept}. \end{aligned}$$

And this is how we update the states on our tape fields. We take into account the partial ordering for the second and fourth instruction, since the letter y must have been located in third position before the current letter c. Therefore, $y > c$ holds.

$$\begin{aligned}
\delta'(\rhd ab) &= [q, a, b] && \text{if } \delta(q_0, \rhd ab) = (q, \mathsf{MVR}),\\
\delta'(\rhd [p, b, y]c) &= [r, b, c] && \text{if } \delta(q_0, \rhd bc) = (r, \mathsf{MVR}) \text{ and } p \neq r,\\
\delta'([q, a, x]bc) &= [p, b, c] && \text{if } \delta(q, abc) = (p, \mathsf{MVR}),\\
\delta'([q, a, x][p, b, y]c) &= [r, b, c] && \text{if } \delta(q, abc) = (r, \mathsf{MVR}) \text{ and } p \neq r.
\end{aligned}$$

If the stored state is correct, we can execute the MVR step:

$$\begin{aligned}
\delta'(\rhd [q, a, x]b) &= \mathsf{MVR} && \text{if } \delta(q_0, \rhd ab) = (q, \mathsf{MVR}),\\
\delta'([q, a, x][p, b, y]c) &= \mathsf{MVR} && \text{if } \delta(q, abc) = (p, \mathsf{MVR}),\\
\delta'(\rhd [q, a, x][p, b, y]) &= \mathsf{MVR},\\
\delta'([q, a, x][p, b, y][r, c, z]) &= \mathsf{MVR}.
\end{aligned}$$

It remains to be shown that $L(M') = L(M)$ holds. Given an input $u \in \Sigma \cup \{\lambda\}$, M' will accept immediately if and when $u \in L(M)$ holds. So let us consider an input $w = a_1 a_2 \dots a_n$, where $n \geq 2$ and $a_1, a_2, \dots, a_n \in \Sigma$. M will scan the input from left to right until it detects the first letter, say a_i, that is to be rewritten into b, that is, M executes the following cycle:

$$\begin{aligned}
q_0 \rhd a_1 a_2 \dots a_n \lhd &\vdash_M^{i-1} \rhd a_1 \dots a_{i-2} q_{i-1} a_{i-1} a_i a_{i+1} \dots a_n \lhd\\
&\vdash_M q_0 \rhd a_1 a_2 \dots a_{i-1} b a_{i+1} \dots a_n \lhd .
\end{aligned}$$

Now M' will simulate this cycle as follows:

$$\begin{aligned}
\rhd a_1 a_2 \dots a_n \lhd \ &\vdash_{M'}^{c} \ \rhd [q_1, a_1, a_2]\, a_2 \dots a_n \lhd\\
&\vdash_{M'}^{c} \ \rhd [q_1, a_1, a_2][q_2, a_2, a_3] a_3 \dots a_n \lhd\\
&\vdash_{M'}^{c^*} \ \rhd [q_1, a_1, a_2] \dots [q_{i-1}, a_{i-1}, a_i] a_i a_{i+1} \dots a_n \lhd\\
&\vdash_{M'}^{c} \ \rhd [q_1, a_1, a_2] \dots [q_{i-1}, a_{i-1}, a_i] b a_{i+1} \dots a_n \lhd
\end{aligned}$$

that is, scanning the tape from left to right, M' replaces the letter a_j $(1 \leq j \leq i-1)$ by the triple $[q_j, a_j, a_{j+1}]$, where q_j is the state in which M reaches the window contents $a_j a_{j+1} a_{j+2}$ with $a_{n+1} = \lhd$. When M reaches the letter a_i, it realizes that it must simulate a rewrite/restart step of M, and accordingly it replaces the letter a_i by the letter b. By induction on the number of cycles that M executes it follows that $L(M') = L(M)$ holds. $\square$

This shows that

$$\mathcal{L}(\textsf{stl-det-ORWW}) = \mathcal{L}(\textsf{det-ORWW})$$

holds.

For this reason, we concentrate on the stl-det-ORWW-automata in what follows, as the det-ORWW-automata can be easily simulated by stl-det-ORWW-automata. Through this simulation the results we get for the stl-det-ORWW-automata can be easily transferred to the det-ORWW-automata.

Up to this point we have shown that we can express all regular languages with stl-det-ORWW-automata.

Summed up we have

$$\textsf{REG} \subseteq \mathcal{L}(\textsf{stl-det-ORWW}) = \mathcal{L}(\textsf{det-ORWW}).$$

Before we actually present a simulation of a stl-det-ORWW-automaton by an NFA it is helpful to get a better understanding for valid computations. We therefore look how we can realize language operations with the stl-det-ORWW-automaton and examine their descriptional complexity.

3.3 The Descriptional Complexity of Language Operations

In the literature many results can be found on the descriptional complexity of language operations in the sense that, given a DFA of size n for a language L, determine the size of a DFA for the language $\text{op}(L)$, where op is an operation like reversal, complement, Kleene star, and correspondingly for binary operations like union, intersection, or product (see [7, 11] for surveys). For example, it is known that the bound for reversal is 2^n [31], for the Kleene star, it is $2^{n-1}+2^{n-2}$, for union and intersection, it is $m \cdot n$, and for product, it is $(m-1) \cdot 2^n + 2^{n-1}$

[44] where the two languages used are accepted by DFAs of size m and n, respectively. We can ask the same question for stl-det-ORWW-automata, using the size of the tape alphabet as complexity measure. Note, that we could also use the total number of window contents for which an operation is defined. Because the number of tape symbols limits this number, we stick with the number of tape symbols.

As it is easier to make generic constructions when we can make assumptions about some properties of the automata we show that every stl-det-ORWW-automaton can be normalized by adding one tape symbol such that the automaton only accepts at the right sentinel.

Lemma 3.3.1. [24] *For each stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, there exists a stl-det-ORWW-automaton $M' = (\Sigma, \Delta, \rhd, \lhd, \delta', >)$ such that $L(M') = L(M)$ and $|\Delta| = |\Gamma|+1$, but M' can only accept when its window contains the right sentinel $\lhd$.*

Proof. Let $\Delta = \Gamma \cup \{\diamond\}$, where $\diamond \notin \Gamma$ is a new symbol, and the transition function δ' is defined as follows, where $a, b, c \in \Gamma \cup \{\rhd, \lhd\}$, and $d, e \in \Gamma \cup \{\rhd, \diamond\}$:

$$\begin{array}{lcll lcl}
\delta'(\rhd\lhd) & = & \mathsf{Accept}, & \text{if } \delta(\rhd\lhd) = \mathsf{Accept}, & \delta'(d\diamond\lhd) & = & \mathsf{Accept}, \\
\delta'(abc) & = & \delta(abc), & \text{if } c = \lhd \text{ or } \delta(abc) \neq \mathsf{Accept}, & \delta'(de\diamond) & = & \mathsf{MVR}, \\
\delta'(abc) & = & \diamond, & \text{if } c \neq \lhd \text{ and } \delta(abc) = \mathsf{Accept}, & \delta'(\diamond bc) & = & \diamond. \\
\delta'(d\diamond e) & = & \mathsf{MVR}, & \text{if } e \neq \lhd, & & &
\end{array}$$

The basic idea is that the diamond symbol is written down if M would accept and the read/write window does not contain the right sentinel.

It is easily seen that M' accepts only with the right sentinel $\lhd$ in its window and that $L(M') = L(M)$ holds. We first show that $L(M) \subseteq L(M')$ holds. Let w be a word that is accepted by M, i.e. $w \in L(M)$. If w is the empty word, it is also accepted by M' by construction and $w \in L(M')$ holds.

Otherwise we have the valid computation of M

$$\rhd w \lhd \vdash_M^* \rhd v_1 \cdots v_{k-1} \underline{v_k} \ldots v_n \lhd \vdash_M \mathsf{Accept}$$

and we get a valid computation of M' that writes the symbol $\diamond$ at the position at which M would accept followed by cycles writing diamond symbols to the right of it.

$$\begin{aligned}&\rhd\, w \lhd \vdash_{M'}^{*} \rhd v_1 \cdots v_{k-1}\underline{v_k} \ldots v_n \vdash_{M'} \rhd v_1 \cdots v_{k-1} \Diamond \ldots v_n \lhd \vdash_{M'}^{c^{n-k}} \\ &\quad \rhd\, v_1 \cdots v_{k-1} \Diamond \ldots \Diamond \lhd \vdash_{M'} \mathsf{Accept}\end{aligned}$$

Thus, we see that w is accepted by M' and $L(M) \subseteq L(M')$ holds.

To show that the opposite direction $L(M') \subseteq L(M)$ is also true is a completely analogous approach. We just take a valid computation of M', look at the cycle where the symbol $\Diamond$ is written, and replace the writing of $\diamond$ by the Accept instruction. □

Now let us have a look at some language operations.

Theorem 3.3.2. [34] *If a language L is accepted by a stl-det-ORWW-automaton with n letters, then its reversal L^R is accepted by a stl-det-ORWW-automaton with a tape alphabet of size $n^2 + 2n$.*

Proof. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton such that $L(M) = L$ and $n = |\Gamma|$. From M we construct a stl-det-ORWW-automaton $M^R = (\Sigma, \Delta, \rhd, \lhd, \delta', >')$ for L^R that proceeds as follows. Given a word w^R as input, M^R scans it from left to right, which corresponds to M scanning the corresponding input w from right to left. Therefore, M^R uses code letters of the form $[a, b]$ to mark the position in w^R to which M advances. Accordingly, during a computation the tape contents of M^R will be of the form $\rhd ua[b, c]v\lhd$, where $u \in \Gamma^*$, $a, b \in \Gamma$, $c \in \Gamma \cup \{\rhd\}$, and $v \in (\Delta \smallsetminus \Gamma)^*$. This factorization tells us that M would move right across the prefix corresponding to $(bv)^R$. Now if $\delta(cba) = b'$, then M would replace b by b' in the next cycle, and correspondingly, M^R rewrites $\rhd ua[b, c]v\lhd$ into $\rhd uab'v\lhd$.

As tape alphabet for M^R we take $\Delta = \Gamma \cup (\Gamma \times (\Gamma \cup \{\rhd\}))$, which shows that $|\Delta| = n + (n \cdot (n + 1)) = n^2 + 2n$, and we define the ordering $>'$ on Δ as follows:

$$\forall a, b \in \Gamma : \text{ if } a > b, \text{ then } a >' [a, c] >' b \text{ for all } c \in \Gamma \cup \{\rhd\},$$

that is, the new letters of the form $[a, c]$ are inserted right below the letter a itself. Further, the transition function δ' is defined as follows, where $a, b, c, c' \in \Gamma$ and $d \in \Gamma \cup \{\rhd\}$:

$$
\begin{array}{rllll}
(1) & \delta'(\rhd a \lhd) & = & \delta(\rhd a \lhd) & \text{for all } a \in \Gamma \cup \{\lambda\}, \\
(2) & \delta'(\rhd ab) & = & \mathsf{MVR}, & \\
(3) & \delta'(abc) & = & \mathsf{MVR}, & \\
(4) & \delta'(ab\lhd) & = & [b, \rhd], & \text{if } \delta(\rhd ba) = \mathsf{MVR}, \\
(5) & \delta'(ab\lhd) & = & c, & \text{if } \delta(\rhd ba) = c, \\
(6) & \delta'(ab\lhd) & = & \mathsf{Accept}, & \text{if } \delta(\rhd ba) = \mathsf{Accept}, \\
(7) & \delta'(ab[c,d]) & = & [b, c], & \text{if } \delta(dcb) = \mathsf{MVR}, \\
(8) & \delta'(ab[c,d]) & = & \mathsf{MVR}, & \text{if } \delta(dcb) = c', \\
(9) & \delta'(b[c,d]A) & = & c', & \text{if } \delta(dcb) = c', \text{ and } A \in (\Delta \smallsetminus \Gamma) \cup \{\lhd\}, \\
(10) & \delta'(ab[c,d]) & = & \mathsf{Accept}, & \text{if } \delta(dcb) = \mathsf{Accept}, \\
(11) & \delta'(\rhd b[c,d]) & = & [b, c], & \text{if } \delta(dcb) = \mathsf{MVR}, \\
(12) & \delta'(\rhd b[c,d]) & = & \mathsf{MVR}, & \text{if } \delta(dcb) = c', \\
(13) & \delta'(b[c,d]A) & = & c', & \text{if } \delta(dcb) = c', \text{ and } A \in (\Delta \smallsetminus \Gamma) \cup \{\lhd\}, \\
(14) & \delta'(\rhd b[c,d]) & = & \mathsf{Accept}, & \text{if } \delta(dcb) = \mathsf{Accept}
\end{array}
$$

To illustrate the way in which M^R works, we present a simple example. Assume that M executes the following computation on input $w = aab$, where we underline the factor in each configuration that the window of M contains:

$$\underline{\rhd aa}b\lhd \vdash_M \rhd\underline{aab}\lhd \vdash_M \underline{\rhd ab}b\lhd \vdash_M \underline{\rhd bb}b\lhd \vdash_M \mathsf{Accept}.$$

Then M^R executes the following computation on input $w^R = baa$:

$$
\begin{array}{llllllll}
\underline{\rhd ba}a\lhd & \vdash^2_{M^R} & \rhd b\underline{aa\lhd} & \vdash_{M^R} & \underline{\rhd ba}[a,\rhd]\lhd & \vdash_{M^R} & \rhd b\underline{a[a,\rhd]}\lhd \\
 & \vdash_{M^R} & \underline{\rhd b[a,a]}[a,\rhd]\lhd & \vdash_{M^R} & \rhd\underline{b[a,a][a,\rhd]}\lhd & \vdash_{M^R} & \underline{\rhd bb}[a,\rhd]\lhd \\
 & \vdash_{M^R} & \rhd b\underline{b[a,\rhd]}\lhd & \vdash_{M^R} & \rhd b\underline{b[a,\rhd]\lhd} & \vdash_{M^R} & \underline{\rhd bb}b\lhd \\
 & \vdash_{M^R} & \rhd\underline{bbb}\lhd & \vdash_{M^R} & \rhd b\underline{bb\lhd} & \vdash_{M^R} & \mathsf{Accept}.
\end{array}
$$

It can now be shown that indeed $L(M^R) = L^R$ holds. □

In the same way as for DFAs, we obtain the following result.

Proposition 3.3.3. [34] *If a language L is accepted by a stl-det-ORWW-*

automaton M with an alphabet of size n, then so is its complement $\overline{L}$.

Proof. We want to accept all words for which the automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ halts without accepting and reject the words which the automaton M accepts. Thus, we construct an automaton $M' = (\Sigma, \Gamma, \rhd, \lhd, \delta', >)$ for the language $\overline{L}$ by removing all Accept instructions and adding Accept instructions for window contents that were previously undefined. That is,

$$\begin{aligned}
\delta'(\rhd\lhd) &= \mathsf{Accept} && \text{if } \lambda \notin L(M)\\
\delta'(abc) &= A && \text{if } \delta(abc) = A\\
\delta'(abc) &= \mathsf{MVR} && \text{if } \delta(abc) = \mathsf{MVR}\\
\delta'(abc) &= \mathsf{Accept} && \text{if } \delta(abc) \notin \Gamma \cup \{\mathsf{MVR}, \mathsf{Accept}\},
\end{aligned}$$

where $a \in \Gamma \cup \{\rhd\}, b \in \Gamma, c \in \Gamma \cup \{\lhd\}$. □

Next, we look how we realize intersections which was already done in [34].

Theorem 3.3.4. *If the languages L_1 and L_2 are accepted by stl-det-ORWW-automata with tape alphabets of size m and n, respectively, then their intersection $L_1 \cap L_2$ is accepted by a stl-det-ORWW-automaton with a tape alphabet of size $m \cdot n + \min(m, n)$.*

Proof. Let $M_1 = (\Sigma, \Gamma_1, \rhd, \lhd, \delta_1, >_1)$ be a stl-det-ORWW-automaton such that $L(M_1) = L_1$, and let $M_2 = (\Sigma, \Gamma_2, \rhd, \lhd, \delta_2, >_2)$ be a stl-det-ORWW-automaton such that $L(M_2) = L_2$, where $|\Gamma_1|= m$ and $|\Gamma_2|= n$. Assume that $m \geq n$. Then we obtain a stl-det-ORWW-automaton M for $L = L_1 \cap L_2$ from M_1 and M_2 as follows. As tape alphabet we take

$$\Delta = \Sigma \cup (\{\, [a, b] \mid a \in \Gamma_1, b \in \Sigma \,\} \smallsetminus \{\, [a, a] \mid a \in \Sigma \,\}) \cup \{\, \bar{b} \mid b \in \Gamma_2 \,\},$$

which is of size $m \cdot |\Sigma|-|\Sigma|+n + |\Sigma|\leq m \cdot n + n$. For the sake of simplicity, we write $[a, a]$ instead of a for $a \in \Sigma$.

$$[a, c] > [b, c] > \overline{d}, \quad a, b \in \Gamma_1, a >_1 b,\ c \in \Sigma, d \in \Gamma_2.$$

Now M works as follows:

1. First M interprets each input letter $a \in \Sigma$ as the pair $[a, a]$.

2. Then M simulates M_1 using the first component of each letter. When M_1 accepts, then M replaces each pair of the form $[a, b]$ by the letter $\bar{b}$, proceeding from right to left. Here we assume without loss of generality that M_1 accepts at the right end of the tape (see Lemma 3.3.1).

3. Finally, M simulates M_2 interpreting each letter of the form $\bar{b}$ just as M_2 would interpret the letter b. If and when M_2 accepts, then so does M.

Obviously, $L(M) = L_1 \cap L_2$ follows. □

The same construction also works for the union operation. We can normalize the automaton in a way such that the automaton always halts at the right sentinel, both when accepting or rejecting. Then we just accept when M_1 would accept and rewrite the symbols from right to left in case M_1 rejects. But we don't have to provide an independent construction because we can use the latter two results.

Corollary 3.3.5. [34] *If the languages L_1 and L_2 are accepted by stl-det-ORWW-automata with tape alphabets of size m and n, respectively, then their union $L_1 \cup L_2$ is accepted by a stl-det-ORWW-automaton with a tape alphabet of size $m \cdot n + \cdot \min(m, n)$.*

Proof. The statement follows directly from

$$L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$$

because complementing does not change the size of the automaton according to Proposition 3.3.3. Therefore, the automaton for $\overline{L_1} \cap \overline{L_2}$ is at most of size $m \cdot n + \min(m, n)$ according to Theorem 3.3.4. □

Thus, for realizing the Boolean operations, stl-det-ORWW-automata are essentially just as efficient as DFAs, while for the operation of reversal, they are really much more efficient. Unfortunately, no results are known so far on how to realize the operations of product and Kleene star efficiently by stl-det-ORWW-automata. Of course, as we will see that we can construct an NFA for any given stl-ORWW-automaton, given two stl-det-ORWW-automata M_1 and M_2, one can first construct NFAs A_1 and A_2 such that $L(A_i) = L(M_i)$, $i = 1, 2$, then one constructs an NFA A for the product $L = L(A_1) \cdot L(A_2)$, and finally, one can convert A into a stl-det-ORWW-automaton M for L. Unfortunately, this construction is quite inefficient. By Corollary 3.5.11, the NFAs A_1 and A_2 may be of exponential size 2^m and 2^n (measured in the size m of the alphabet of M_1 and the size n of the alphabet of M_2), and so the size of A may be up to $2^m + 2^n$, which means that M may need a tape alphabet of size $2^{2^m+2^n}$ by Proposition 3.2.2.

Finally, regular languages might also be given through DFAs. Therefore, we consider the problem of constructing a small stl-det-ORWW-automaton for the intersection or union of languages that are given through DFAs.

Theorem 3.3.6. [34] *If the languages L_1 and L_2 over an alphabet of size k are accepted by DFAs of size m and n, respectively, then their intersection $L_1 \cap L_2$ is accepted by a stl-det-ORWW-automaton with a tape alphabet of size $k \cdot (\min(m, n) + 1) + \max(m, n)$.*

Proof. Let $A_1 = (Q_1, \Sigma, q_0, F_1, \delta_1)$ be a DFA such that $L(A_1) = L_1$, where $|Q_1|= m$ and $|\Sigma|= k$, and let $A_2 = (Q_2, \Sigma, p_0, F_2, \delta_2)$ be a DFA such that $L(A_2) = L_2$, where $|Q_2|= n$. W.l.o.g. we assume that $m \leq n$. From A_1 and A_2 we construct a stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ for $L = L_1 \cap L_2$ as follows:

- $\Gamma = \Sigma \cup \{ [q, a] \mid q \in Q_1 \text{ and } a \in \Sigma \} \cup Q_2$, that is, $|\Gamma|= k + k \cdot m + n = k \cdot (\min(m, n) + 1) + \max(m, n)$,
- the ordering $>$ is defined by taking $a > [q, b] > p$ for all $a, b \in \Sigma$, $q \in Q_1$, and $p \in Q_2$,
- and the transition function δ is defined by the following equations, where $a, b, c \in \Sigma$, $q, q' \in Q_1$, and $p, p', p_1, p_2 \in Q_2$:

$$\begin{array}{llcll}
(1) & \delta(\rhd a \lhd) & = & \mathsf{Accept}, & \text{if } a \in (\Sigma \cup \{\lambda\}) \cap L_1 \cap L_2,\\
(2) & \delta(\rhd ab) & = & [q, a], & \text{where } q = \delta_1(q_0, a),\\
(3) & \delta(\rhd [q, a]b) & = & \mathsf{MVR}, & \\
(4) & \delta([q, a]bc) & = & [q', b], & \text{where } q' = \delta_1(q, b),\\
(5) & \delta(\rhd [q, a][q', b]) & = & p, & \text{where } p = \delta_2(p_0, a),\\
(6) & \delta(\rhd p[q, a]) & = & \mathsf{MVR}, & \\
(7) & \delta(p[q, a]b) & = & \mathsf{MVR}, &
\end{array}$$

$$\begin{array}{rlcll}
(8) & \delta(p[q,a][q',b]) & = & p', & \text{where } p' = \delta_2(p,a),\\
(9) & \delta(\rhd pp') & = & \mathsf{MVR}, & \\
(10) & \delta(pp_1p_2) & = & \mathsf{MVR}, & \\
(11) & \delta(pp_1[q,a]) & = & \mathsf{MVR}, & \\
(12) & \delta([q,a]b\lhd) & = & [q',b], & \text{where } q' = \delta_1(q,b),\\
(13) & \delta(p[q,a]\lhd) & = & \mathsf{Accept}, & \text{if } q \in F_1 \text{ and } \delta_2(p,a) \in F_2.
\end{array}$$

Thus, M works as follows:

1. First M simulates two steps of A_1 by turning the first two input letters ab into the pairs $[q,a][q',b]$, where $q = \delta_1(q_0,a)$ and $q' = \delta_1(q,b)$.
2. On the prefix $[q,a][q',b]$ the first step of A_2 can already be simulated by rewriting $[q,a]$ into p, where $p = \delta_2(p_0,a)$.
3. Now, M alternates between simulating a step of A_1 and a step of A_2.
4. If the last two letters are of the form $p[q,c]$ for some $p \in Q_2$ and $q \in F_1$, and if $\delta_2(p,c) \in F_2$, then M accepts.

It is easily seen that $L(M) = L(A_1) \cap L(A_2)$. □

Observe that by first building the product DFA of A_1 and A_2 and by then turning this into an equivalent stl-det-ORWW-automaton, we would obtain a stl-det-ORWW-automaton for the language $L = L(A_1) \cap L(A_2)$ with a tape alphabet of size $k + m \cdot n$. As typically m and n are large, while k is small (e.g., often binary languages are considered, that is, $k = 2$), Theorem 3.3.6 gives a much better bound. Obviously, the above construction can be adopted to the operation of union. Also, it can easily be extended to the intersection or union of $t \geq 3$ DFAs.

Corollary 3.3.7. [34] *For $t \geq 3$, if A_i is a DFA with n_i states over an alphabet of size k for all $1 \leq i \leq t$, then there exists a stl-det-ORWW-automaton M with a tape alphabet of size $k \cdot (1 + n_1 + \ldots + n_{t-1}) + n_t$ such that $L(M) = \bigcap_{i=1}^{t} L(A_i)$ (or $L(M) = \bigcup_{i=1}^{t} L(A_i)$).*

In fact, this result can even be extended to an arbitrary Boolean combination of t DFAs. By turning the left-to-right simulation of a DFA as described in the proof of Proposition 3.2.2 into a right-to-left simulation, we obtain the following result.

Proposition 3.3.8. [34] *If a language L over an alphabet of size k is accepted by a DFA of size n, then the language L^R is accepted by a stl-det-ORWW-automaton with a tape alphabet of size $k + n$.*

Finally, we compare the upper bounds for the mentioned language operations in the following table where n and m are the sizes of the corresponding automata.

Operation	DFA	stl-det-ORWW
Reversal $\cdot^R$	2^n	$n^2 + 2n$
Complement $\overline{L_1}$	n	n
Union $L_1 \cup L_2$	$m \cdot n$	$m \cdot n + \min(m, n)$
Intersection $L_1 \cap L_2$	$m \cdot n$	$m \cdot n + \min(m, n)$

Every DFA with m states and an alphabet of size n can be simulated by a stl-det-ORWW with a tape alphabet of size $m + n$.

Therefore, the stl-det-ORWW-automaton can represent languages at least as concise as the DFA.

For some operations like reversal the stl-det-ORWW-automaton is much more efficient. As a result, regarding complexity on standard operations, the stl-det-ORWW-automaton is beneficial.

Before we explain how we construct a new computation from two computations we introduce the nondeterministic stateless variant because the simulation method is nearly the same for both types of ordered restarting automata.

3.4 Nondeterministic Stateless Restarting Automata

We remember that formally the *nondeterministic stateless ordered restarting automaton* is a one-tape Turing machine that is described by a 6-tuple $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, where Σ is a finite input alphabet, Γ a finite tape alphabet that contains Σ, but not the symbols $\rhd$ and $\lhd$ that serve as markers for the left and right border of the input word which at the same time mark the working space. Furthermore,

$$\delta : (((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \cup \{\rhd\lhd\}) \to 2^{\{\mathsf{MVR}\} \cup \Gamma} \cup \{\mathsf{Accept}\}$$

is a transition function, and $>$ is a partial ordering on Γ.

In this section we will see how this automaton compares to the stl-det-ORWW-automaton regarding conciseness and descriptional complexity of language operations. As far as the language class is concerned, we have already announced that it also corresponds to the regular languages.

Because of the nondeterminism we can simulate an NFA by a stl-ORWW-automaton in the same way as we can simulate a DFA by a stl-det-ORWW-automaton.

Proposition 3.4.1. *For each NFA $A = (Q, \Sigma, q_0, F, \Delta)$, there exists a stl-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ such that $L(M) = L(A)$ and $|\Gamma| = |Q| + |\Sigma|$.*

Proof. We take $\Gamma = \Sigma \cup Q$ and define $a > q$ for all $a \in \Sigma$ and all $q \in Q$. On input of a word $w = a_1 a_2 \dots a_n$, where $n \geq 1$ and $a_1, a_2, \dots, a_n \in \Sigma$, the stl-ORWW-automaton M simply replaces each letter a_i by a state from $\Delta(q_0, a_1 a_2 \dots a_i)$, proceeding from left to right. It accepts if and when the last letter a_n has been replaced by a final state of A. Also, we add the transition $\delta(\rhd\lhd) = \mathsf{Accept}$ if $\lambda \in L(A)$. $\square$

Since we now have nondeterminism at our disposal, it makes sense to examine the effect on the descriptional complexity of language operations. For convenience, we work with normalized automata again and show that we can assume that the automata only accept at the right sentinel.

We can use the the same construction as for the det-stl-ORWW-automata in Lemma 3.3.1.

Lemma 3.4.2. *For each stl-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, there exists a stl-ORWW-automaton $M' = (\Sigma, \Delta, \rhd, \lhd, \delta', >)$ such that $L(M') = L(M)$ and $|\Delta| = |\Gamma| + 1$, but M' can only accept when its window contains the right sentinel $\lhd$.*

Proof. The construction works exactly as the one for det-stl-ORWW-automata. If we would accept, we write down a special symbol. From this point on, we write down the special symbol until we reach the right sentinel and accept.

Formally, this is expressed in the following way.

Let $\Delta = \Gamma \cup \{\diamond\}$, where $\diamond \notin \Gamma$ is a new symbol, and the transition function δ' is defined as follows, where $a, b, c \in \Gamma \cup \{\rhd, \lhd\}$, and $d, e \in \Gamma \cup \{\rhd, \diamond\}$:

$$\begin{array}{lcll lcl}
\delta'(\rhd\lhd) & = & \mathsf{Accept}, & \text{if } \delta(\rhd\lhd) = \mathsf{Accept}, & \delta'(d\diamond\lhd) & = & \mathsf{Accept}, \\
\delta'(abc) & = & \delta(abc), & \text{if } \delta(abc) \neq \mathsf{Accept}, & \delta'(de\diamond) = \delta'(d\diamond e) & = & \{\mathsf{MVR}\}, \\
\delta'(abc) & = & \{\diamond\}, & \text{if } \delta(abc) = \mathsf{Accept}, & \delta'(\diamond bc) & = & \{\diamond\}.
\end{array}$$

□

In the same way as we can achieve that the automaton accepts at the right sentinel, we can achieve that it accepts at the left sentinel. This will become beneficial for simplifying the argumentation for some constructions.

Lemma 3.4.3. *For each stl-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, there exists a stl-ORWW-automaton $M' = (\Sigma, \Delta, \rhd, \lhd, \delta', >)$ such that $L(M') = L(M)$ and $|\Delta| = |\Gamma| + 1$, but M' can only accept when its window contains the left sentinel $\rhd$.*

Proof. The construction works exactly as the one for the det-stl-ORWW-automata. If we would accept we write down a special symbol. From this point on, we write down the special symbol when we see it in the read/write window. This continues until we see the special symbol at the left most position where we accept. Formally this is expressed in the following way.

Let $\Delta = \Gamma \cup \{\diamond\}$, where $\diamond \notin \Gamma$ is a new symbol, and the transition function δ' is defined as follows, where $a, b, c \in \Gamma \cup \{\rhd, \lhd\}$, and $d \in \Gamma \cup \{\lhd, \diamond\}$:

$$\begin{array}{lcll lcl}
\delta'(\rhd\lhd) & = & \mathsf{Accept}, & \text{if } \delta(\rhd\lhd) = \mathsf{Accept}, & \delta'(\rhd\diamond d) & = & \mathsf{Accept}, \\
\delta'(abc) & = & \delta(abc), & \text{if } \delta(abc) \neq \mathsf{Accept}, & \delta'(ab\diamond) & = & \{\diamond\}, \\
\delta'(abc) & = & \{\diamond\}, & \text{if } \delta(abc) = \mathsf{Accept}. & & &
\end{array}$$

□

For the operations reversal and intersection we can use the same constructions as in Theorems 3.3.2 and 3.3.4.

We cannot use the same construction for union because the construction for complement does not work for nondeterminism. Given a stl-ORWW-automaton there may not be a stl-ORWW-automaton of the same size for the complement language. But first, let us see the construction for union.

Proposition 3.4.4. *If the languages L_1 and L_2 are accepted by stl-ORWW-automata with tape alphabets of size m and n, respectively, then their union $L_1 \cup L_2$ is accepted by a stl-ORWW-automaton with a tape alphabet of size $m + n + \min(m, n)$.*

Proof. We use the tape alphabet

$$\Theta = \Sigma \cup (\Gamma_1 \times \{1\}) \cup (\Gamma_2 \times \{2\}).$$

Thus, we have a tape alphabet of size $|\Theta| = |\Sigma| + |m| + |n| \leq |m| + |n| + \min(m, n)$.

The automaton M works as follows. In the first cycle M moves to the right sentinel. At this position it nondeterministically marks the last letter with 1 or 2. After that every letter is marked with the same number from right to left. Finally, we simulate M_1 or M_2 depending on the marking.

□

Additionally, we have efficient constructions for concatenation and the Kleene star operator because we can make use of nondeterminism.

Closure under product: Let $M_1 = (\Sigma, \Gamma_1, \rhd, \lhd, \delta_1, >_1)$ and $M_2 = (\Sigma, \Gamma_2, \rhd, \lhd, \delta_2, >_2)$ be two stl-ORWW-automata. Without loss of generality we may assume that both these automata accept at the right end of their tapes. We present an ORWW-automaton M for the language $L(M_1) \cdot L(M_2)$. It proceeds as follows:

1. Given a word $w \in \Sigma^*$ as input, M rewrites w from right to left, letter by letter, such that each letter of a suffix v of w is marked by an index 2, and then each letter of the corresponding prefix u is marked by an index 1. The right-most letter is allowed to have an index 1 if $L(M_2)$ contains the empty word. In this way $w \in \Sigma^*$ is nondeterministically split into $w = uv$ with the idea that $u \in L(M_1)$ and $v \in L(M_2)$ are to be checked.
2. Then M simulates M_1 on the prefix u. During this process, the leftmost occurrence of a letter with index 2 is interpreted as the right delimiter $\lhd$.
3. When the simulated computation of M_1 on u accepts, then M realizes this with either the right delimeter $\lhd$ or with the leftmost letter with index 2 in its window. In the former case, it accepts if and only if $\lambda \in L(M_2)$, while in the latter case it executes a MVR step to simulate M_2.
4. In all following cycles M will move to this position to allow the simulation

of M_2. Now M accepts if and only if this computation of M_2 accepts.
5. If in step 1, all letters are marked with an index 2, that is, $v = w$ and $u = \lambda$ are chosen, then M simply simulates M_2 on v, provided $\lambda \in L(M_1)$; otherwise, it simply halts without acceptance.

M is described in detail in the proof of the following lemma.

Lemma 3.4.5. *Given two stl-ORWW-automata* $M_1 = (\Sigma, \Gamma_1, \rhd, \lhd, \delta_1, >_1)$ *and* $M_2 = (\Sigma, \Gamma_2, \rhd, \lhd, \delta_2, >_2)$ *we can construct a stl-ORWW-automaton* M *of size* $m + n + \min(m, n)$ *such that* $L(M) = L(M_1) \cdot L(M_2)$*.*

Proof. The automaton M is obtained as follows. First we assume that both automata accept only at the right sentinel. To recognize a word $w = w_1 w_2, w_1 \in L(M_1), w_2 \in L(M_2)$, we use two disjoint tape alphabets to distinguish between w_1 and w_2.

We use two different sets of symbols to mark the two sections of the word. For an easier representation we use pairs for the symbols which leads to the following tape alphabet

$$\Gamma = \Sigma \cup (\Gamma_1 \times \{1\}) \cup (\Gamma_2 \times \{2\}).$$

The transition function δ is described as follows

$$\begin{aligned}
\delta(\rhd\lhd) &= \mathsf{Accept} \text{ if } \lambda \in L(M_1) \cdot L(M_2),\\
\delta(\rhd a\lhd) &= \mathsf{Accept} \text{ if } a \in L(M_1) \cdot L(M_2),\\
\delta(\rhd ab) &= \{\mathsf{MVR}\},\\
\delta(abc) &= \{\mathsf{MVR}\},\\
\delta(ab\lhd) &\ni [b,1] \text{ if } \lambda \in L(M_2),\\
\delta(ab\lhd) &\ni [b,2],\\
\delta(ab[c,1]) &\ni [b,1],\\
\delta(\rhd b[c,1]) &\ni [b,1]\\
\delta(ab[c,2]) &= \{[b,1],[b,2]\},\\
\delta(\rhd b[c,2]) &\ni [b,1],\\
\delta(\rhd b[c,2]) &\ni [b,2] \text{ if } \lambda \in L(M_1),\\
\delta(\rhd[b,1][c,2]) &\ni \mathsf{MVR} \text{ if } b \in L(M_2),
\end{aligned}$$

$$\begin{aligned}
\delta(\rhd[a,i][b,i]) &= (\delta_i(\rhd ab) \cap (\Gamma_i \times \{i\})) \cup (\delta_i(\rhd ab) \cap \{\mathsf{MVR}\}),\\
\delta([a,i][b,i][c,i]) &= (\delta_i(abc) \cap (\Gamma_i \times \{i\})) \cup (\delta_i(abc) \cap \{\mathsf{MVR}\}),\\
\delta([a,1][b,1][c,2]) &= \mathsf{MVR}, \text{ if } \delta_1(ab\lhd) = \mathsf{Accept},\\
\delta([a,1][b,1][c,2]) &\supseteq (\delta_1(ab\lhd) \cap \Gamma_1) \times \{1\},\\
\delta([a,1][b,2][c,2]) &\supseteq (\delta_2(\rhd bc) \cap \Gamma_2) \times \{2\},\\
\delta([a,1][b,2]\lhd) &= \mathsf{Accept} \text{ if } b \in L(M_2),\\
\delta([a,1][b,1]\lhd) &\supseteq (\delta_1(ab\lhd) \cap \Gamma_1) \times \{1\},\\
\delta([a,1][b,1]\lhd) &= \mathsf{Accept} \text{ if } \delta_1(ab\lhd) = \mathsf{Accept},\\
\delta([a,2][b,2]\lhd) &\supseteq (\delta_2(ab\lhd) \cap \Gamma_2) \times \{2\},\\
\delta([a,2][b,2]\lhd) &= \mathsf{Accept} \text{ if } \delta_2(ab\lhd) = \mathsf{Accept}.
\end{aligned}$$

This gives us an automaton M for the product $L(M_1) \cdot L(M_2)$. □

Closure under Kleene star: Given a stl-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ of size n we can construct a stl-ORWW-automaton M' of size $3n$ such that $L(M') = L(M)^*$.

Here the idea is essentially the same as for the operation of product. Given a word $w \in \Sigma^*$ as input, M' rewrites the word from right to left, letter by letter, attaching indices 1 or 2 to these letters. In this way a factorization $w = u_1 u_2 \dots u_m$ is chosen nondeterministically, and it remains to check that $u_1, u_2, \dots, u_m \in L(M)$ hold.

The details of the automaton are given in the proof of the following lemma.

Lemma 3.4.6. *Given a stl-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ of size n we can construct a stl-ORWW-automaton $M' = (\Sigma, \Theta, \rhd, \lhd, \delta', >')$ of size $3n$ such that $L(M') = L(M)^*$.*

Proof. Again we assume that M only accepts at the right sentinel to simplify the description of the transition function. We use the alphabet

$$\Theta = \Sigma \cup \Gamma \times \{1, 2\}$$

Thus we have a tape alphabet of size

$$|\Theta| = |\Sigma| + |\Gamma \times \{1,2\}| \leq n + 2n = 3n.$$

The transition function is defined as follows, where $a, b, c \in \Gamma$ and $(i, j) \in \{(1,2),(2,1)\}$.

$$\begin{aligned}
\delta'(\rhd\lhd) &= \mathsf{Accept}, && \\
\delta'(\rhd a\lhd) &= \mathsf{Accept} && \text{if } a \in L(M), \\
\delta'(\rhd ab) &= \{\mathsf{MVR}\}, && \\
\delta'(abc) &= \{\mathsf{MVR}\}, && \\
\delta'(ab\lhd) &= \{[b,1],[b,2]\}, && \\
\delta'(ab[c,i]) &= \{[b,1],[b,2]\}, && \\
\delta'(\rhd b[c,i]) &= \{[b,1]\}, && \\
\delta'(\rhd[a,1][b,1]) &= (\delta(\rhd ab) \cap \{\mathsf{MVR}\}) && \\
&\quad \cup ((\delta(\rhd ab) \cap \Gamma) \times \{1\}), && \\
\delta'(\rhd[a,1][b,2]) &= \{\mathsf{MVR}\} && \text{if } a \in L(M), \\
\delta'([a,i][b,i][c,i]) &= (\delta(abc) \cap \{\mathsf{MVR}\}) && \\
&\quad \cup ((\delta(abc) \cap \Gamma) \times \{i\}), && \\
\delta'([a,j][b,i][c,j]) &= \{\mathsf{MVR}\} && \text{if } b \in L(M), \\
\delta'([a,j][b,i][c,i]) &= (\delta(\rhd bc) \cap \{\mathsf{MVR}\}) && \\
&\quad \cup (\delta(\rhd bc) \cap \Gamma) \times \{i\}, && \\
\delta'([a,i][b,i][c,j]) &= \{\mathsf{MVR}\} && \text{if } \delta(ab\lhd) = \mathsf{Accept}, \\
\delta'([a,i][b,i][c,j]) &= (\delta(ab\lhd) \cap \Gamma) \times \{i\} && \text{if } \delta(ab\lhd) \neq \mathsf{Accept}, \\
\delta'([a,i][b,i]\lhd) &= \mathsf{Accept} && \text{if } \delta(ab\lhd) = \mathsf{Accept}, \\
\delta'([a,i][b,i]\lhd) &= (\delta(ab\lhd) \cap \Gamma) \times \{i\} && \text{if } \delta(ab\lhd) \neq \mathsf{Accept}, \\
\delta'([a,i][b,j]\lhd) &= \mathsf{Accept} && \text{if } b \in L(M).
\end{aligned}$$

In this way, the automaton M accepts the language $L(M)^*$. $\square$

Now that we are a little bit more familiar with stl-ORWW-automata, we finally present the main construction of this thesis.

3.5 Constructing an NFA from a stl-ORWW Automaton

In this section we show that the language class described by deterministic (det-stl-ORWW) as well as nondeterministic (stl-ORWW) stateless ordered restarting automata is the class of regular languages. We have already shown that stl-ORWW-automata can recognize all regular languages. Here we will finally show that they do not recognize any other languages.

We show this by using the Myhill-Nerode Theorem (see e.g. [16]). One important aspect of the proof is a technique for constructing a new accepting computation from two given accepting computations. For this purpose we introduce the concept of *patterns*.

Furthermore, we will use the equivalence classes we get from the patterns to describe a simulation by an NFA. We will also apply this simulation to the det-stl-ORWW-automaton which will improve the upper bound of the previously known conversion [36].

Finally, we modify the NFA such that we only need to check local conditions for the transition function. This drastically reduces the amount of work required for the NFA computations which is important for the decision algorithms.

In order to simplify our argumentation we normalize our ordered automata. We demand that the automaton accepts at the left sentinel. Again, this is no real restriction, as an ORWW-automaton can be easily adapted such that it only accepts at the left sentinel (see Lemma 3.4.3).

Now, we start with the proof for regularity.

Theorem 3.5.1. *Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_M, >)$ be a stl-ORWW-automaton. Then $L(M)$ is a regular language.*

Proof. We want to show that stl-ORWW-automata can only accept regular languages. So let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-ORWW-automaton accepting the language $L = L(M)$. By Lemma 3.4.3 we can assume that M always accepts at the left end of its tape, that is, immediately after executing a rewrite/restart operation or right at the beginning.

The transition relation δ of M can be represented by a finite set of four-tuples of the form (a, b, c, r), where $a \in \Gamma \cup \{\rhd\}$, $b \in \Gamma$, $c \in \Gamma \cup \{\lhd\}$ and

$r \in \Gamma \cup \{\mathsf{MVR}, \mathsf{Accept}\}$. We can partially order these four-tuples as follows:

$$\begin{array}{ll} (a,b,c,d) > (a',b',c',e) & \text{if } a \geq a', b \geq b', c \geq c' \text{ and } (a,b,c) \neq (a',b',c'); \\ (a,b,c,\mathsf{MVR}) > (a,b,c,b') & \text{for all } b' \in \Gamma. \end{array}$$

As $|\Gamma| = n$, this set consists of $K \leq (n+1) \cdot n \cdot (n+1) \cdot (n-1+2) = n \cdot (n+1)^3$ many four-tuples. Hence, we can introduce a new alphabet $\Omega = \{t_1, t_2, \ldots, t_K\}$ the symbols of which are in 1-to-1 correspondence to these four-tuples. The partial ordering of four-tuples induces a partial ordering on Ω.

Now, let $w \in L$, and let C be an accepting computation of M on input w. With each integer j, $1 \leq j \leq |w|$, we can associate a word $x_j \in \Omega^*$ that corresponds to the sequence of operations that M executes within the computation C at position j, that is, when the j-th letter is in the middle of its window. Let $x_j = t_{i_1} t_{i_2} \ldots t_{i_s}$, where $t_{i_r} \in \Omega$, $1 \leq r \leq s$. Then, for each $r = 1, 2, \ldots, s-1$, $t_{i_r} = t_{i_{r+1}}$ or $t_{i_r} > t_{i_{r+1}}$, and equality can occur only for move-right operations. Now by $A_j^C(x)$ we denote the word that we obtain from x_j by ignoring all repetitions of letters. Then $A_j^C(x) = \hat{t}_{i_1} \hat{t}_{i_2} \ldots \hat{t}_{i_{\hat{s}}}$ such that $\hat{t}_{i_1} > \hat{t}_{i_2} > \ldots > \hat{t}_{i_{\hat{s}}}$. We call that word a *pattern*. In addition, we have $\hat{s} \leq 4n$ as there are at most $n-1$ rewrites and $3 \cdot (n-1) + 1$ different window contents for MVR operations. Thus, the number of different patterns for M is finite.

Claim 1. For all $x, y, z, u \in \Sigma^*$ if there exist a computation C_{xz} for the word xz and a computation C_{yu} for the word yu such that $\alpha := A_{|x|}^{C_{xz}}(xz) = A_{|y|}^{C_{yu}}(yu)$, then there exists a computation C' for the word yz.

Proof. Let $c_1, c_2, \ldots, c_{k_1}$ be the sequence of cycles of the computation C_{xz}, and let $d_1, d_2, \ldots, d_{k_2}$ be the sequence of cycles of the computation C_{yu}. A cycle c_i (d_i) is called a *short cycle* if the rewrite/restart operation of this cycle is executed within the prefix x (y), which means that this cycle does not contribute to the sequence of operations described by α. All other cycles are called *long cycles*. From the cycles of C_{xz} and C_{yu} we now construct the sequence of cycles of the computation C' inductively.

Our goal is to execute every rewrite operation that is executed in the y-part of the C_{yu} computation and the z-part of the C_{xz} computation. It is ensured that when executing all three computations up to a certain pattern letter a_i, that is, the remaining cycles do not have the operations a_i at the chosen position, the tape contents of common parts coincide then, that is, we have the tape contents of the form $x'z', y'u'$ and $y'z'$.

We start with the empty sequence, and we consider the cycles $d_1, d_2, \ldots, d_{k_2}$ of C_{yu} one after the other. If the current cycle d_i considered is a short cycle, then we append it to the sequence of cycles of C'. By doing this we execute the rewrite operation in the y-part.

If the current cycle d_i is a long cycle that executes a rewrite/restart operation at position $|y|$, which means that this cycle contributes a rewrite/restart operation to α, then again we simply append it to C'. The computation stays valid as the rewrite operation occurs in both computations. Therefore we have the correct window content.

Finally, if the current cycle d_i is a long cycle that executes a move-right operation t at position $|y|$, then let $d_i^{(1)}$ denote the initial part of this cycle up to this operation. Now, let $c_j, c_{j+1}, \ldots, c_{j+s}$ be all those long cycles of C_{xz} that contain the operation t at position $|x|$. Short cycles that might be in between do not contain that operation and are being ignored. Then the cycle d_i and all the cycles $c_j, c_{j+1}, \ldots, c_{j+s}$ contribute the same letter t to α. Let $c_j^{(2)}, c_{j+1}^{(2)}, \ldots, c_{j+s}^{(2)}$ be the suffixes of the latter cycles that start after the move-right operation t. We now combine the prefix $d_i^{(1)}$ with all these suffixes and append the resulting cycles $d_i^{(1)}c_j^{(2)}, d_i^{(1)}c_{j+1}^{(2)}, \ldots, d_i^{(1)}c_{j+s}^{(2)}$ in this order to C'. In addition, we skip all other cycles of C_{yu} that execute the operation t at position $|x|$. These cycles can be executed as we do not change anything in the y-part. This ensures that we can execute the prefix $d_i^{(1)}$ and the suffixes have the same effect as in the computation C_{xz}.

Finally, once all cycles of C_{yu} have been dealt with, we append the accepting tail of C_{yu} to C'. It is easily seen that C' is indeed an accepting computation of M on input yz, and that $A^{C'}_{|y|}(yz) = \alpha$. □

Now, with a word $w \in \Sigma^*$, we associate the following set of words of length at most $4n$ over Ω:

$$S(w) = \{\, A^{C}_{|w|}(wz) \mid z \in \Sigma^* \text{ and } C \text{ is an accepting computation of } M \text{ for } wz \,\},$$

that is, $x \in S(w)$ if there exist a word $z \in \Sigma^*$ and an accepting computation C of M on input wz such that x describes the sequence of operations that

M executes within C at position $|w|$. Of course, there are only finitely many different such subsets of Ω^*.

For proving that the language $L = L(M)$ is regular, we now use the Myhill-Nerode Theorem (see, e.g., [16]). By $\sim_L$ we denote the Myhill-Nerode relation on Σ^* that is induced by L: $x \sim_L y$ if $\forall z \in \Sigma^* : (xz \in L \text{ iff } yz \in L)$.

Claim 2. For all $x, y \in \Sigma^*$, if $S(x) = S(y)$, then $x \sim_L y$.

Proof. Let $x, y \in \Sigma^*$ such that $S(x) = S(y)$, and let $z \in \Sigma^*$ such that $xz \in L$. Then M has an accepting computation C_{xz} on input xz. Hence, $\alpha := A^{C_{xz}}_{|x|}(xz) \in S(x)$. As $S(x) = S(y)$, $\alpha \in S(y)$, that is, there exist a word $u \in \Sigma^*$ and an accepting computation C_{yu} of M on input yu such that $A^{C_{yu}}_{|y|}(yu) = \alpha$. According to Claim 1 M then also has an accepting computation C' on input yz.

It follows that, for all $z \in \Sigma^*$, if $xz \in L$, then also $yz \in L$. Hence, by symmetry, we obtain that $x \sim_L y$ holds. □

As there are only finitely many different sets $S(x)$, the index of the relation $\sim_L$ is finite, which implies that $L = L(M)$ is a regular language. □

Now it is time to illustrate the concept of patterns by an example.

Example 3.5.2. *Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_M, >)$ with $\Sigma = \{a, b, c\}$, $\Gamma = \Sigma \cup \{a', b', c'\}$. For the sake of completeness, we provide the transition function, which we will only use indirectly, since we determine the possible operations on the basis of the valid computations. Let δ be defined by the following equations:*

$$\begin{aligned} \delta(\rhd ab) = \delta(\rhd ca) = \delta(cab) = \delta(abb) = \delta(bbb) &= \{\mathsf{MVR}\} \\ \delta(bbc) = \delta(bb\lhd) = \delta(bbb') = \delta(abb') &= \{b'\} \\ \delta(cab') &= \{a'\} \\ \delta(\rhd ca') = \delta(\rhd ab') &= \mathsf{Accept} \end{aligned}$$

This permits the following valid computation for the word abbb.

$$C = \rhd abbb\lhd \vdash^c \rhd abbb'\lhd \vdash^c \rhd abb'b'\lhd \vdash^c \rhd ab'b'b'\lhd \vdash \mathsf{Accept}$$

Thus, we execute these operations in the following order.

Cycle c_1	$(\rhd, a, b, \mathsf{MVR})$	(a, b, b, MVR)	(b, b, b, MVR)	$(b, b, \lhd, b')$
Cycle c_2	$(\rhd, a, b, \mathsf{MVR})$	(a, b, b, MVR)	(b, b, b', b')	
Cycle c_3	$(\rhd, a, b, \mathsf{MVR})$	(a, b, b', b')		
Tail	$(\rhd, a, b', \mathsf{Accept})$			

Without using a special alphabet we get

$$x_2 = (a, b, b, \mathsf{MVR})(a, b, b, \mathsf{MVR})(a, b, b', b')$$

by looking at all operations we execute at position 2.

By removing the repetitions we get the pattern

$$A_2^C(abbb) = (a, b, b, \mathsf{MVR})(a, b, b', b')$$

We now specify a second computation C_2 for a word with the same pattern at some position. We use the word cabbc.

$$C_2 = \rhd cabbc \lhd \vdash^c \rhd cabb'c \lhd \vdash^c \rhd cab'b'c \lhd \vdash^c \rhd ca'b'b'c \lhd \vdash \mathsf{Accept}$$

Thus, we execute these operations in the following order.

Cycle d_1	$(\rhd, c, a, \mathsf{MVR})$	(c, a, b, MVR)	(a, b, b, MVR)	(b, b, c, b')
Cycle d_2	$(\rhd, c, a, \mathsf{MVR})$	(c, a, b, MVR)	(a, b, b', b')	
Cycle d_3	$(\rhd, c, a, \mathsf{MVR})$	(c, a, b', a')		
Tail	$(\rhd, c, a', \mathsf{Accept})$			

So we get

$$A_3^{C_2}(cabbc) = (a, b, b, \mathsf{MVR})(a, b, b', b')$$

as pattern at position 3 for the word cabbc

As $A_2^C(abbb)$ and $A_3^{C_2}(cabbc)$ coincide, Claim 1 of the previous proof tells us that there is also a valid computation C_{cabbb} for the word cabbb. To match the same notation we take $x = ab, z = bb, y = cab,$ and $u = bc$.

According to the proof we start by considering the cycles d_1, d_2, d_3 one after another. The first cycle d_1 has the MVR operation (a, b, b, MVR) at position 3. Thus, we have a long cycle and have to use the initial part

$$d_1^{(1)} = (\rhd, c, a, \mathsf{MVR})(c, a, b, \mathsf{MVR})(a, b, b, \mathsf{MVR})$$

for the cycles of C that have the same MVR operation at position 2 *in order* *to reach that specific position. These are the cycles c_1 and c_2 and we get the partial cycles*

$$c_1^{(2)} = (b, b, b, \mathsf{MVR})(b, b, \lhd, b')$$

and

$$c_2^{(2)} = (b, b, b', b').$$

We now get the first two cycles by appending the new cycles $d_1^{(1)}c_1^{(2)}$ and $d_1^{(1)}c_2^{(2)}$ to our currently empty computation C_{cabbb}.

The next cycle d_2 executes a rewrite operation at position 3. Thus, we simply append it to our new computation. We do the same with cycle d_3 because it is a short cycle, as it executes a rewrite at position 2. Finally, we append the accepting tail to our computation. To sum it up we get to execute the following operations in the constructed valid computation:

$d_1^{(1)}c_1^{(2)}$	$(\rhd, c, a, \mathsf{MVR})$	(c, a, b, MVR)	(a, b, b, MVR)	(b, b, b, MVR)	$(b, b, \lhd, b')$
$d_1^{(1)}c_1^{(2)}$	$(\rhd, c, a, \mathsf{MVR})$	(c, a, b, MVR)	(a, b, b, MVR)	(b, b, b', b')	
d_2	$(\rhd, c, a, \mathsf{MVR})$	(c, a, b, MVR)	(a, b, b', b')		
d_3	$(\rhd, c, a, \mathsf{MVR})$	(c, a, b', a')			
Tail	$(\rhd, c, a', \mathsf{Accept})$				

Written as computation relation we can write

$$\rhd cabbb \lhd \vdash^c \rhd cabbb' \lhd \vdash^c \rhd cabb'b' \lhd \vdash^c \rhd cab'b'b' \lhd \vdash^c \rhd ca'b'b'b' \lhd \vdash \mathsf{Accept}$$

Claim 2 in the proof of Theorem 3.5.1 allows additionally to construct an NFA of size $2^{\mathcal{O}(n)}$ for $L(M)$.

Theorem 3.5.3. *Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_M, >)$ be a stl-ORWW-automaton that only accepts when its window contains the left sentinel $\rhd$. Then an NFA $A = (Q, \Sigma, \Delta, q_0, F)$ can be constructed from M such that $L(A) = L(M)$ and $|Q| \leq 2^{13\cdot|\Gamma|}$.*

Proof. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_M, >)$ be a stl-ORWW-automaton that only accepts when its window contains the left sentinel $\rhd$, and let $n = |\Gamma|$. Like in the previous proof we introduce a new finite alphabet $\Omega = \{t_1, t_2, \ldots, t_K\}$, the symbols of which are in 1-to-1 correspondence to the operations M can execute. The NFA $A = (Q, \Sigma, \Delta, q_0, F)$ works as follows. The states Q consist of the initial state q_0, the final state q_F and all possible operation patterns. As the length of operation patterns is limited by $4n$, the number of states is also limited: $Q \subseteq \{q_0, q_F\} \cup \Omega^{\leq 4n}$. Only q_F is a final state: $F = \{q_F\}$.

The transition relation is given through the following rules, where $x \in \Sigma$ and $\alpha, \alpha_1, \alpha_2 \in \Omega^{\leq 4n}$ are operation patterns.

- $\Delta(q_0, \lambda) \ni q_F$ if $\delta_M(\rhd\lhd) = \mathsf{Accept}$
- $\Delta(q_0, x) \ni \alpha$ if there exist $w = w_1 w_2 \dots w_m$ and an accepting computation C for w such that $w_1 = x$ and $A_1^C(w) = \alpha$.
- $\Delta(\alpha_1, x) \ni \alpha_2$ if there exist $w = w_1 w_2 \dots w_m$, $1 \leq i < m$, and an accepting computation C for w such that $w_{i+1} = x$, $A_i^C(w) = \alpha_1$ and $A_{i+1}^C(w) = \alpha_2$.
- $\Delta(\alpha, \lambda) \ni q_F$ if there exist $w = w_1 w_2 \dots w_m$ and an accepting computation C for w such that $A_m^C(w) = \alpha$.

We will prove that $L(A) = L(M)$ holds. As usual this proof is divided into two parts.

Claim 1. $L(M) \subseteq L(A)$.
Let $w \in \Sigma^*$ be a word that belongs to the language $L(M)$. Thus, there is an accepting computation C that starts with the initial configuration $\rhd w \lhd$. If $w = \lambda$, then $\delta_M(\rhd\lhd) = \mathsf{Accept}$, which implies that $q_F \in \Delta(q_0, \lambda)$. It follows that $w \in L(A)$ holds in this case.

Assume that $w = w_1 w_2 \dots w_m$ for some $m \geq 1$ and letters $w_1, w_2, \dots, w_m \in \Sigma$. We now claim that

$$q_0 \to A_1^C(w) \to A_2^C(w) \to \dots \to A_m^C(w) \to q_F$$

is an accepting path for the word w in the NFA A. This is obviously true, because we can always take w as the word and C as the accepting computation which fulfills the transition conditions. $\square$

Now, we have to prove the other direction.

Claim 2. $L(A) \subseteq L(M)$.
Let $w \in \Sigma^*$ be a word that belongs to the language $L(A)$. Thus, there is an accepting path that accepts w. We will proceed by induction on the length

of w. If $w = \lambda$, then $q_0 \to q_F$ is the only accepting path. Thus, we have $\Delta(q_0, \lambda) \ni q_F$, which implies $\delta_M(\rhd\lhd) \ni \mathsf{Accept}$.

We claim that for every word $v = v_1 \dots v_m$ for which A has the execution sequence

$$q_0 \to \alpha_1 \to \alpha_2 \to \dots \to \alpha_m$$

there exist a word $u \in \Sigma^*$ and an accepting computation for the word vu such that $A_m^C(vu) = \alpha_m$.

For $m = 1$ we get the execution sequence $q_0 \to \alpha_1$ for the word $v = v_1$, which tells us that $\Delta(q_0, v_1) \ni \alpha_1$ holds. This is only the case if a word $w = w_1 w_2 \dots w_m$ and an accepting computation for w exist such that $w_1 = v_1$ and $A_1^C(w) = \alpha$. Therefore, our claim holds for $u = w_2 \dots w_m$ and the given computation.

For $m > 1$ we get the execution sequence $q_0 \to \alpha_1 \to \dots \to \alpha_{m-1} \to \alpha_m$ for the word $v = v_1 \dots v_{m-1} v_m$. We can apply our induction hypothesis to the execution sequence $q_0 \to \alpha_1 \to \dots \to \alpha_{m-1}$ for the word $v' = v_1 \dots v_{m-1}$, which tells us that we have some word u' and a computation C' such that $A_{m-1}^C(v'u') = \alpha_{m-1}$. From the transition $\Delta(\alpha_{m-1}, v_m) \ni \alpha_m$ we get a word $w = w_1 \dots w_i w_{i+1} \dots w_k$ and a computation C'' such that $w_{i+1} = v_m$, $A_i^{C''}(w) = \alpha_{m-1}$ and $A_{i+1}^{C''}(w) = \alpha_m$. With the help of the construction from the previous proof we can construct a new computation C for the word $v_1 v_2 \dots v_{m-1} w_{i+1} \dots w_k$. Thus, with the word $u = w_{i+2} \dots w_k$ the computation C is a computation for the word vu with $A_m^C(vu) = \alpha_m$. Therefore, our claim holds for all $m \geq 1$.

Now, if we have the accepting path

$$q_0 \to \alpha_1 \to \alpha_2 \to \dots \to \alpha_m \to q_F$$

for the word $w = v_1 \dots v_m$, there exists a word $u \in \Sigma^*$ and a computation C such that it is a computation for the word vu with $A_m^C(vu) = \alpha_m$. As α_m directly leads to the accepting state q_F, all operations of α_m contain the right sentinel $\lhd$ and u must be the empty word λ.

It remains to verify that $|Q| \leq 2^{13 \cdot n}$ holds. For doing that, it suffices to show that $2^{13 \cdot n}$ is an upper bound for the number of patterns that can occur within the NFA A. Since we need precise information about the operations for the estimation, we no longer abstract with the newly introduced alphabet, but use a matrix to represent a pattern. Let

$$\Pi = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \mathrm{op}_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \mathrm{op}_2 \\ \cdots & \cdots & \cdots & \cdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \mathrm{op}_m \end{bmatrix}$$

be such a pattern. Each row represents an operation which can be written as a four-tuple. Let us assume that there are v different letters in the first column, r different letters in the second and s different letters in the third column, where $v, r, s \in \{1, 2, \ldots, \alpha\}$. As $a_{i+1,j} \leq a_{i,j}$ for all $1 \leq i \leq m-1$ and $1 \leq j \leq 3$, each of these columns can be represented by a subset of Γ, which yields $\binom{n}{v}$, $\binom{n}{r}$, and $\binom{n}{s}$ choices for the first, second, and third column, respectively.

Next, we need to encode the information on the letter $a_{j,3}$ that is used in the rewriting of $a_{j,2}$ into $a_{j+1,2}$. This we can do by giving a multiset of cardinality $r-1$ with elements from the set of integers $\{1, 2, \ldots, s\}$, where the j-th integer k_j tells us that $a_{j,3}$ is the k_j-th element from the subset that contains the letters from the third column. This yields another $\binom{s+r-1-1}{r-1}$ choices, as there are $\binom{s+r-1-1}{r-1}$ such multisets. As from one row of Π to the next, at most one of the letters in column 2 and column 3 can change, it follows that all letters $a_{j,3}$ that are not chosen in this multiset are only used for move-right operations. However, some of the letters that are used for rewrite operations may also be used for move-right operations before the rewrite operation is actually executed. Let's say this happens t times, where $0 \leq t \leq r-1$. This gives us another $\binom{r-1}{t}$ choices. Further, in the last row we either have a move-right operation, if $a_{1,3} \neq \lhd$, or we have an accept step, if $a_{1,3} = \lhd$.

So far we have described the last three columns of Π which contain up to $r+s+t$ different triples and for which we have $\binom{n}{r} \cdot \binom{n}{s} \cdot \binom{s+r-2}{r-1} \cdot \binom{r-1}{t}$ choices. It remains to combine these triples with the information on the v letters in the first column, which can change from any row to the next. Here we add to each triple $(a_{j,2}\ a_{j,3}\ \mathsf{MVR})$ the index ℓ_j of the last letter from the first column such that $(a_{\ell_j,1}\ a_{j,2}\ a_{j,3}\ \mathsf{MVR})$ is a row in Π. This yields $\binom{v+r+s+t-1}{r+s+t}$ choices, as there are this many corresponding multisets. In addition, we add to each of the triples the information of whether the last letter from the first column that is used with the previous triple is also used with the current triple, that is, whether the entry in the first column stays the same when going from the previous triple to the new one. This yields another 2^{r+s+t} choices. Thus, altogether this yields $\binom{n}{r} \cdot \binom{n}{s} \cdot \binom{s+r-2}{r-1} \cdot \binom{r-1}{t} \cdot \binom{n}{v} \cdot \binom{v+r+s+t-1}{r+s+t} \cdot 2^{r+s+t}$ choices.

As v, r, s can range from 1 to n, this yields the following upper bound for the number of states $|Q|$:

$$\begin{array}{rcl} |Q| & \leq & 1 + \sum_{r,s,v \in \{1,2,\ldots,n\}} \sum_{t=0}^{r-1} \binom{n}{r} \cdot \left(\binom{n}{s} \cdot \binom{s+r-2}{r-1} \cdot \binom{r-1}{t} + 1\right) \cdot \\ & & \left(\binom{n}{v} \cdot \binom{v+r+s+t-1}{r+s+t} \cdot 2^{r+s+t} + 1\right) \\ & \leq & 1 + \sum_{r,s,v \in \{1,2,\ldots,\alpha\}} \sum_{t=0}^{r-1} \binom{n}{r} \cdot \binom{n}{s} \cdot \binom{2n}{r-1} \cdot \binom{n}{t} \cdot \binom{n}{v} \cdot \binom{4n}{r+s+t} \cdot 2^{r+s+t} \\ & = & 2^n \cdot 2^n \cdot 2^{2n} \cdot 2^n \cdot 2^n \cdot 2^{4n} \cdot 2^{3n} = 2^{13 \cdot n}. \end{array}$$

□

If M is a stateless deterministic ORWW-automaton, then we obtain a better upper bound. Again we have $r \leq \alpha$ different letters in the second column and $s \leq \alpha$ different letters in the third column. However, M is deterministic, and so, if a row $(a_{j,1}\ a_{j,2}\ a_{j,3}\ \mathsf{MVR})$ contains a move-right step, then the next rewrite operation in column 2 can occur only after the letter in row 3 has been rewritten. Thus, a pair $(a_{j,2}, a_{j,3})$ can only occur in a move-right step or in a rewrite step. Hence, we need a multiset of $r-1$ integers from the set $\{1, 2, \ldots, s\}$ to indicate the letters from column 3 that are used in the various rewrite steps, which yields again $\binom{s+r-2}{r-1}$ options. Further, the letter in the first column can change, that is, it can be rewritten, only after the letter in the second column has been rewritten, and so we need only a multiset of cardinality r from Γ that indicates the letter in column 1 that occur together with a given letter from the second column, which yields another $\binom{n+r-1}{r}$ options. Hence, in this case we get the following upper bound for the size of the NFA:

$$\begin{array}{rcl} |Q| & \leq & 1 + \sum_{r=1}^{n} \sum_{s=1}^{n} \binom{n}{r} \cdot \left(\binom{\alpha}{s} \cdot \binom{s+r-2}{r-1} + 1\right) \cdot \left(\binom{n+r-1}{r} + 1\right) \\ & \leq & 1 + \sum_{r=1}^{n} \sum_{s=1}^{n} \binom{n}{r} \cdot \binom{n}{s} \cdot \binom{2n}{r-1} \cdot \binom{2n}{r} \\ & = & 2^n \cdot 2^n \cdot 2^{2n} \cdot 2^{2n} = 2^{6 \cdot n}. \end{array}$$

Thus, the construction in the proof of Theorem 3.5.3 yields the following result, which improves the result given in [24].

Corollary 3.5.4. *From a stl-det-ORWW-automaton M with a tape alphabet of size α, an NFA $A = (Q, \Sigma, p_0, F, \delta_A)$ can be constructed such that $|Q| \leq 2^{6 \cdot (\alpha+1)}$*

and $L(A) = L(M)$.

Proof. From the automaton M we can construct a stl-det-ORWW-automaton M for the same language with a tape alphabet of size $\alpha + 1$ that only accepts at the left sentinel. From this automaton we construct an NFA like described in the proof of Theorem 3.5.3. □

Thus, stateless ORWW-automata can just accept the same languages as their deterministic counterparts. However, stateless ORWW-automata can describe some regular languages much more succinctly than stateless deterministic ORWW-automata as we will see in Lemma 3.5.14 and Proposition 3.5.15.

As we use existential conditions in the previous NFA construction these conditions are not easily verified. Now, we use the introduced matrix representation to describe the NFA in more practical terms by using conditions that can be verified locally. As it feels more natural we assume that we only accept at the right sentinel this time. This can also be achieved by adding just one tape symbol.

Now we can prepare the description of the construction.

Theorem 3.5.5. *From a stl-ORWW-automaton that only accepts at the right sentinel and that has an alphabet of size n we can construct an NFA of size 2^{13n}.*

In order to prove the theorem we specify how the patterns have to look like and what patterns can be next to each other. After that we have to verify that these conditions are sufficient to describe a valid computation.

For technical reasons we divide the patterns into several blocks.

Let Π_1 and Π_2 be two patterns of the stl-ORWW-automaton M:

$$\Pi_1 = \begin{bmatrix} a_1 & b_1 & c_1 & \mathrm{op}_1 \\ a_2 & b_2 & c_2 & \mathrm{op}_2 \\ \dots & \dots & \dots & \dots \\ a_m & b_m & c_m & \mathrm{op}_m \end{bmatrix} \text{ and } \Pi_2 = \begin{bmatrix} a'_1 & b'_1 & c'_1 & \mathrm{op'}_1 \\ a'_2 & b'_2 & c'_2 & \mathrm{op'}_2 \\ \dots & \dots & \dots & \dots \\ a'_r & b'_r & c'_r & \mathrm{op'}_r \end{bmatrix}$$

A right block (R-block) RB of Π_1 is a vertical section of maximal size such that the second and third elements of each row do not change from row to row. That is, there exist an index i_1 and a number s_1 such that

$$RB = \begin{bmatrix} a_{i_1} & b_{i_1} & c_{i_1} & \mathrm{op}_{i_1} \\ a_{i_1+1} & b_{i_1+1} & c_{i_1+1} & \mathrm{op}_{i_1+1} \\ \dots & \dots & \dots & \dots \\ a_{i_1+s_1} & b_{i_1+s_1} & c_{i_1+s_1} & \mathrm{op}_{i_1+s_1} \end{bmatrix}$$

where $(b_{i_1}, c_{i_1}) = (b_{i_1+1}, c_{i_1+1}) = \ldots = (b_{i_1+s_1}, c_{i_1+s_1})$ and for all $k < i_1$ or $k > i_1 + s_1$: $(b_k, c_k) \neq (b_{i_1}, c_{i_1})$.

A left block (L-block) LB of Π_2 is a vertical section of maximal size such that the first two elements of each row do not change from row to row. That is there exists an index j_1 and a number t_1 such that

$$LB = \begin{bmatrix} a'_{j_1} & b'_{j_1} & c'_{j_1} & \mathrm{op'}_{j_1} \\ a'_{j_1+1} & b'_{j_1+1} & c'_{j_1+1} & \mathrm{op'}_{j_1+1} \\ \dots & \dots & \dots & \dots \\ a'_{j_1+t_1} & b'_{j_1+t_1} & c'_{j_1+t_1} & \mathrm{op'}_{j_1+t_1} \end{bmatrix}$$

where $(a'_{j_1}, b'_{j_1}) = (a'_{j_1+1}, b'_{j_1+1}) = \ldots = (a'_{j_1+t_1}, b'_{j_1+t_1})$ and for all $k < j_1$ or $k > j_1 + t_1$: $(a'_k, b'_k) \neq (a'_{j_1}, b'_{j_1})$.

The right block RB is right-compatible to the left block LB if $(b_{i_1}, c_{i_1}) = (a'_{j_1}, b'_{j_1})$.

Π_1 is divided into several R-Blocks

$$\Pi_1 = \begin{bmatrix} a_1 & b_1 & c_1 & \mathrm{op}_1 \\ a_2 & b_2 & c_2 & \mathrm{op}_2 \\ \dots & \dots & \dots & \dots \\ a_m & b_m & c_m & \mathrm{op}_m \end{bmatrix} = \begin{bmatrix} RB_1 \\ RB_2 \\ \dots \\ RB_u \end{bmatrix}$$

and Π_2 is separated into the following L-blocks

$$\Pi_2 = \begin{bmatrix} a'_1 & b'_1 & c'_1 & \mathrm{op'}_1 \\ a'_2 & b'_2 & c'_2 & \mathrm{op'}_2 \\ \dots & \dots & \dots & \dots \\ a'_m & b'_m & c'_m & \mathrm{op'}_r \end{bmatrix} = \begin{bmatrix} LB_1 \\ LB_2 \\ \dots \\ LB_v \end{bmatrix}$$

We say that the pattern Π_2 is right-compatible to Π_1 if they satisfy the following conditions.

1. The number u of R-blocks of Π_1 must be at least as large as the number v of L-blocks of Π_2, that means $v \leq u$.
2. We denote the j-th R-block that contains MVR operations with RB_{i_j}. There are exactly v of these blocks and the right block RB_{i_j} fits to the left block LB_j for $1 \leq j \leq v$.
3. Only one of the matching blocks RB_{i_j} and LB_j can end with a rewrite operation.
4. There is a total number of $u-1$ rewrite operations in both patterns.
5. If one corresponding pair of blocks does not contain a rewrite operation, they are the blocks RB_u and LB_v.

Because of our design criteria the following result can be easily checked.

Lemma 3.5.6. *Let $w = w_1 w_2 \dots w_n \in L_C(M)$, let C be an accepting computation of M for w, and let Π_i be the pattern of the i-th tape field obtained from the computation C for $1 \leq i \leq n$. Then Π_{i+1} is right-compatible to Π_i for $1 \leq i \leq n$.*

The following theorem tells us that our definition of the local right-compatibility condition is sufficient to extract a valid computation for a word.

Theorem 3.5.7. *Let $w_1, w_2, \dots w_n \in \Gamma$, $w_0 = \rhd$, $w_{n+1} = \lhd$, $w = w_1 w_2 \dots w_n$, and let $(\Pi_1, \Pi_2, \dots, \Pi_n)$ be a sequence of patterns where*

$$\Pi_i = \begin{bmatrix} a_1^i & b_1^i & c_i^i & \mathrm{op}_1^i \\ a_2^i & b_2^i & c_2^i & \mathrm{op}_2^i \\ \dots & \dots & \dots & \dots \\ a_{k_i}^i & b_{k_i}^i & c_{k_i}^i & \mathrm{op}_{k_i}^i \end{bmatrix}.$$

Additionally,

- $a_1^i = w_{i-1}, b_1^i = w_i, c_1^i = w_{i+1}$ *for* $1 \leq i \leq n$,
- Π_{i+1} *is right-compatible to* Π_i *for* $1 \leq i \leq n-1$.

Then $w \in L_C(M)$, and an accepting computation of M for w can be extracted from the sequence of patterns $(\Pi_1, \dots, \Pi_n)$.

Proof. Let N be the combined number of rewrite operations that occur in the patterns $\Pi_1, \Pi_2, \ldots, \Pi_n$. We claim that there exists an accepting computation C for the word w that consists of N cycles. We will prove this by induction over N.

If we have no rewrite at all, that is, $N = 0$, then the pattern Π_i has the following form for $1 \leq i \leq n$:

$$\Pi_i = \begin{bmatrix} w_{i-1} & w_i & w_{i+1} & \mathrm{op}_i \end{bmatrix},$$

where $\mathrm{op}_i = \mathsf{MVR}$ for $1 \leq i \leq n-1$ and $\mathrm{op}_n = \mathsf{Accept}$.

Note, that for $n = 1$ there is only the operation

$$(w_0, w_1, w_2, \mathsf{Accept}) = (\rhd, w_1, \lhd, \mathsf{Accept})$$

which leads to the computation $\rhd w \lhd = \rhd w_1 \lhd \vdash_M \mathsf{Accept}$. For $n > 1$ we get the operations $(w_0, w_1, w_2, \mathsf{MVR}), (w_1, w_2, w_3, \mathsf{MVR}), \ldots, (w_{n-1}, w_n, w_{n+1}, \mathsf{Accept})$ which describes the accepting tail computation $\rhd w \lhd = \rhd w_1 w_2 \ldots w_n \lhd \vdash_M^* \mathsf{Accept}$.

If there are some rewrites, that is, $N > 0$, we assume that we can construct a valid computation from valid pattern sequences that have less than N rewrites. The patterns are in their most generic form. Now, let K be the minimal number such that $\mathrm{op}_1^K \in \Gamma$. This leads to $\mathrm{op}_1^i = \mathsf{MVR}$ for all $1 \leq i \leq K-1$. The first cycle is therefore described by the sequence of operations $(\rhd, w_1, w_2, \mathsf{MVR}), (w_1, w_2, w_3, \mathsf{MVR}), \ldots,$ $(w_{K-2}, w_{K-1}, w_K, \mathsf{MVR}), (w_{K-1}, w_K, w_{K+1}, w'_K)$ with $w'_K = b_2^K$ It can be written as $\rhd w \lhd = \rhd w_1 w_2 \ldots w_n \lhd \vdash_M^c \rhd w_1 w_2 \ldots w_{K-1} w'_K w_{K+1} \ldots w_n \lhd = \rhd w' \lhd$, where we take $w' = w_1 w_2 \ldots w_{K-1} w'_K w_{K+1} \ldots w_n$.

We are going to construct a new sequence of patterns $(\Pi'_1, \Pi'_2, \ldots, \Pi'_n)$ for the word w' that contains $N - 1$ rewrites.

Patterns that are to the right of our rewritten tape field stay the same, that is, $\Pi'_j = \Pi_j$ for $j > K$.

Let Π_K have j rows. Then Π_K looks like

$$\Pi_K = \begin{bmatrix} a_1^K & b_1^K & c_1^K & b_2^K \\ a_2^K & b_2^K & c_2^K & \mathrm{op}_2^K \\ \dots & \dots & \dots & \dots \\ a_j^K & b_j^K & c_j^K & \mathrm{op}_j^K \end{bmatrix}$$

we simply delete the first row from Π_k because we do not execute the operation $(w_{K-1}, w_K, w_{K+1}, b_2^K)$ and get

$$\Pi'_K = \begin{bmatrix} a'^K_1 & b'^K_1 & c'^K_1 & \mathrm{op}'^K_1 \\ \dots & \dots & \dots & \dots \\ a'^K_{j'} & b'^K_{j'} & c'^K_{j'} & \mathrm{op}'^K_{j'} \end{bmatrix} = \begin{bmatrix} a_2^K & b_2^K & c_2^K & \mathrm{op}_2^K \\ \dots & \dots & \dots & \dots \\ a_j^K & b_j^K & c_j^K & \mathrm{op}_j^K \end{bmatrix}$$

As a MVR operation in a pattern can represent multiple operations, we have to have a closer look at the MVR operations of the cycle we want to remove.

The K-th symbol changes from w_K to $w'_K = b_2^K$ and therefore we also get $(a_1^K, b_1^K) \neq (a'^K_1, b'^K_1) = (a_2^K, b_2^K)$. This means that the topmost operation of Π'_K is not compatible to the topmost operation of Π_{K-1} and thus we have to remove the first line of Π_{K-1} to get the new pattern Π'_{K-1}.

We do the same for the remaining pattern Π_i for $1 \leq i \leq K-2$, where we treat the rightmost patterns first. If $(a_1^{i+1}, b_1^{i+1}) \neq (a'^{i+1}_1, b'^{i+1}_1)$, that is one right block is being removed, we have to remove the first row of Π_i to get the new pattern Π'_i.

This concludes the proof.

□

Now we can specify the construction.

Theorem 3.5.8. *Given a stl-ORWW-automaton M with an alphabet of size α, an NFA $A = (Q, \Sigma, S, F, \delta_A)$ can be constructed such that $|Q| \leq 2^{13\cdot\alpha}$ and $L(A) = L(M)$. In the case of M being deterministic there exists an NFA A with $|Q| \leq 2^{6\cdot\alpha}$.*

Proof. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-ORWW-automaton. Without loss of generality we may assume that M only accepts at the right sentinel. Let $\alpha = |\Gamma|$ be the size of $M's$ tape alphabet.

The NFA $A = (Q, \Sigma, S, F, \delta_A)$ is constructed as follows:

- The set of states Q consists of valid patterns as defined in the previous theorem and an additional state q_+.
- The set of initial states S contains all states that correspond to patterns, which have the left sentinel $\rhd$ at position 1. The state q_+ is added if $\lambda \in L(M)$, that is, $L(M)$ contains the empty word.
- The set of final states only contains the state q_+.
- The transition function δ is defined as follows, where $a \in \Sigma$ and Π is a pattern with a at position 2

$$\delta_A(\Pi, a) = \begin{cases} \{\Pi' \mid \Pi' \text{ is comp. to } \Pi\}, & \text{if } \Pi \text{ has } b \in \Sigma \text{ at pos. 3 in row 1,} \\ \{q_+\}, & \text{if } \Pi \text{ has } \lhd \text{ at position 3.} \end{cases}$$

It remains to show that the NFA A really describes the same language as the stl-ORWW-automaton M.

Let $w \neq \lambda, w = w_1 w_2 \dots w_n$ be a word that is accepted by the NFA A. During the accepting computation A guesses the sequence of patterns $(\Pi_1, \Pi_2, \dots, \Pi_n)$ such that

- Π_1 has $\rhd$ at position 1.
- Π_n has $\lhd$ at position 3 and an Accept operation in the last row.
- Π_i has the letter w_i at position 2 in the first row for $1 \leq i \leq n$.
- Π_i is right-compatible to Π_{i+1} for $1 \leq i \leq n-1$.

According to the previous theorem, we conclude that $w \in L(M)$ holds.

Now let $w \in L(M)$. Then there exists an accepting computation C of M for input w. Let $(\Pi_1, \Pi_2, \dots, \Pi_n)$ be the sequence of patterns obtained from the computation C. Then $\Pi_1 \to \Pi_2 \to \dots \to \Pi_n$ is an accepting path of the NFA A for the word w.

Since here again we are talking about possible pattern sequences and only the conditions were reformulated as local, the proof and the results of the previous construction can be reused. This gives us the estimate of the theorem. $\square$

Actually, the bound given in Theorem 3.5.8 is sharp with respect to its order of magnitude. To show this, we consider a sequence of example languages $(B_n)_{n\geq 3}$. Let $\Sigma = \{0, 1, \#, \$\}$ be an input alphabet, let $n \geq 3$ be a positive integer, and let B_n be the following language over Σ:

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

Hence, if $w \in B_n$, then $w = v_1\#v_2\#\ldots\#v_m\$u$, where $v_1, v_2, \ldots, v_m, u$ are factors of length n over $\{0, 1\}$ such that at least one of the first m factors coincides with the last factor u. It follows that $|w| = (m + 1) \cdot n + m$ for some positive integer m, and it is easily seen that B_n is a regular language over Σ.

Proposition 3.5.9. [34] *B_n is accepted by a stl-det-ORWW-automaton $M_n = (\Sigma, \Gamma_n, \rhd, \lhd, \delta_n, >)$ with an alphabet Γ_n of size $18 \cdot n$.*

Proof. The stl-det-ORWW-automaton $M_n = (\Sigma, \Gamma_n, \rhd, \lhd, \delta_n, >)$ will work in n phases. Let $w = v_1\#v_2\#\ldots\#v_m\$u$ be given as input, where $m \geq 1$ and $v_1, v_2, \ldots, v_m, u \in \{0, 1\}^n$, $v_j = v_{j,1}v_{j,2}\ldots v_{j,n}$, $1 \leq j \leq m$, and $u = u_1u_2\ldots u_n$. In phase i, $1 \leq i \leq n$, M_n will shift the information about the letter u_{n+1-i} to the left until this information reaches the first letter of w. While doing so, it compares this letter to the letter $v_{j,n+1-i}$ of v_j for all $1 \leq j \leq m$, and it stores the result of this comparison on the tape by replacing the letter $v_{j,n+1-i}$ by some appropriate auxiliary letter. Finally, after phase n has been completed, M_n moves across the current tape content and checks whether there is a syllable all of its letters have been matched successfully. In the affirmative, M_n accepts.

Now, we describe the stl-det-ORWW-automaton M_n in detail. First we define the tape alphabet Γ_n. It will contain the input alphabet $\Sigma = \{0, 1, \#, \$\}$, marked copies $[0, i]$ and $[1, i]$, $1 \leq i \leq n$, of the letters 0 and 1 that M_n will use to mark the letters of the syllable u that have been compared already to the corresponding letters of the other syllables, letters of the form $[a, i, b]$, where $a \in \{0, 1, +, -, \#, \$\}$, $1 \leq i \leq n$, and $b \in \{0, 1\}$, that will be used to shift the information on the letters of u to the left, and letters of the form $\overline{[c, i, b]}$, where

$c \in \{+,-\}$, $2 \leq i \leq n$, and $b \in \{0,1\}$. Thus, we take

$$\begin{array}{rcl} \Gamma_n & = & \{0,1,\#,\$\} \cup \{\, [0,i],[1,i] \mid 1 \leq i \leq n \,\} \\ & & \cup \{\, [a,i,b], \mid a \in \{0,1,+,-,\#,\$\},\ 1 \leq i \leq n,\ b \in \{0,1\} \,\} \\ & & \cup \{\, \overline{[+,i,b]}, \overline{[-,i,b]} \mid 2 \leq i \leq n,\ b \in \{0,1\} \,\}. \end{array}$$

We see that $|\Gamma_n| = 18 \cdot n$ as stated above. Next, we define the partial order $>$ on Γ_n:

$$\begin{array}{ccll} a & > & [a,i] & \text{for all } a \in \{0,1\} \text{ and } 1 \leq i \leq n, \\ a & > & [a,n,b] & \text{for all } a \in \{0,1,\#,\$\} \text{ and } b \in \{0,1\}, \\ [a,i+1,b] & > & [a,i,b'] & \text{for all } a \in \{0,1,\#,\$,+,-\},\ b,b' \in \{0,1\}, \\ & & & \text{and } 1 \leq i < n, \\ a & > & \overline{[c,n,b]} > [c,n,b] & \text{for all } a,b \in \{0,1\} \text{ and } c \in \{+,-\}, \\ [a,i+1,b] & > & \overline{[c,i,b']} > [c,i,b'] & \text{for all } a,b,b' \in \{0,1\},\ c \in \{+,-\}, \\ & & & \text{and } 2 \leq i < n, \\ [a,2,b] & > & [c,1,b'] & \text{for all } a,b,b' \in \{0,1\} \text{ and } c \in \{+,-\}. \end{array}$$

Finally, we describe the transition function δ_n, dividing this description into n phases as indicated above.

1.1. The information on the last letter u_n of u is to be shifted to the left and compared to the last letters of the other syllables. Accordingly, M_n moves its read/write window to the right until it contains the right sentinel $\lhd$ by using the following transitions, where $a_1, a_2, a_3 \in \{0,1\}$:

$$\begin{array}{lll} \delta_n(\rhd a_1 a_2) = \mathsf{MVR}, & \delta_n(a_1 a_2 a_3) = \mathsf{MVR}, & \delta_n(a_1 a_2 \#) = \mathsf{MVR}, \\ \delta_n(a_1 \# a_2) = \mathsf{MVR}, & \delta_n(\# a_1 a_2) = \mathsf{MVR}, & \delta_n(a_1 a_2 \$) = \mathsf{MVR}, \\ \delta_n(a_1 \$ a_2) = \mathsf{MVR}, & \delta_n(\$ a_1 a_2) = \mathsf{MVR}. & \end{array}$$

1.2. Now the letter u_n, that is, the letter immediately to the left of the right sentinel $\lhd$, is rewritten into the letter $[u_n, n]$, and then the information on u_n is shifted to the left. Additionally, the last letter of each other syllable, that is, the letter immediately to the left of the letter \$ or a letter $\#$, is compared to u_n, and a positive result is indicated by the letter $\overline{[+,n,u_n]}$, and a negative result is indicated by the letter $\overline{[-,n,u_n]}$. The corresponding transitions are the following, where $a_1, a_2, a_3, b \in \{0,1\}$

and $c \in \{+, -\}$:

$$\begin{array}{lcllcl}
\delta_n(a_1 b \lhd) & = & [b, n], & \delta_n(a_1 a_2 [b, n]) & = & [a_2, n, b], \\
\delta_n(a_1 a_2 [a_3, n, b]) & = & [a_2, n, b], & \delta_n(\$ a_1 [a_2, n, b]) & = & [a_1, n, b], \\
\delta_n(a_1 \$ [a_2, n, b]) & = & [\$, n, b], & \delta_n(a_1 a_2 [\$, n, b]) & = & \overline{[-, n, b]} \text{ for } a_2 \neq b, \\
\delta_n(a_1 b [\$, n, b]) & = & \overline{[+, n, b]}, & \delta_n(a_1 a_2 \overline{[c, n, b]}) & = & [a_2, n, b], \\
\delta_n(\# a_1 [a_2, n, b]) & = & [a_1, n, b], & \delta_n(a_1 \# [a_2, n, b]) & = & [\#, n, b], \\
\delta_n(a_1 b [\#, n, b]) & = & \overline{[+, n, b]}, & \delta_n(a_1 a_2 [\#, n, b]) & = & \overline{[-, n, b]} \text{ for } a_2 \neq b, \\
\delta_n(\rhd a_1 [a_2, n, b]) & = & [a_1, n, b]. & & &
\end{array}$$

Using these transitions M_n executes the following computation given the word

$$\begin{gathered}
w = v_{1,1} \ldots v_{1,n-1} v_{1,n} \# v_{2,1} \ldots v_{2,n-1} v_{2,n} \# \ldots \\
\ldots \# v_{m,1} \ldots v_{m,n-1} v_{m,n} \$ u_1 \ldots u_{n-1} u_n
\end{gathered}$$

as input, where we assume that $v_{1,n} \neq u_n$ and $v_m = u$:

$$\begin{aligned}
&\rhd v_{1,1} \ldots v_{1,n-1} v_{1,n} \# v_{2,1} \ldots v_{2,n-1} v_{2,n} \# \ldots \# v_{m,1} \ldots v_{m,n-1} v_{m,n} \$ u_1 \ldots \\
&\quad u_{n-1} u_n \lhd \vdash^{c}_{M_n} \rhd v_{1,1} \ldots v_{1,n-1} v_{1,n} \# v_{2,1} \ldots v_{2,n-1} v_{2,n} \# \ldots \# v_{m,1} \ldots \\
&\quad v_{m,n-1} v_{m,n} \$ u_1 \ldots u_{n-1} [u_n, n] \lhd \vdash^{c}_{M_n} \rhd v_{1,1} \ldots v_{1,n-1} v_{1,n} \# v_{2,1} \ldots \\
&\quad v_{2,n-1} v_{2,n} \# \ldots \# v_{m,1} \ldots v_{m,n-1} v_{m,n} \$ u_1 \ldots [u_{n-1}, n, u_n][u_n, n] \lhd \\
&\quad \vdash^{c^*}_{M_n} \rhd v_{1,1} \ldots v_{1,n-1} v_{1,n} \# \ldots \# v_{m,1} \ldots \\
&\quad v_{m,n-1} v_{m,n} [\$, n, u_n][u_1, n, u_n] \ldots [u_{n-1}, n, u_n][u_n, n] \lhd \vdash^{c}_{M_n} \rhd v_{1,1} \ldots \\
&\quad v_{1,n-1} v_{1,n} \# \ldots \# v_{m,1} \ldots v_{m,n-1} \overline{[+, n, u_n]} [\$, n, u_n][u_1, n, u_n] \ldots \\
&\quad [u_{n-1}, n, u_n][u_n, n] \lhd \vdash^{c^*}_{M_n} \rhd [v_{1,1}, n, u_n] \ldots \\
&\quad [v_{1,n-1}, n, u_n] \overline{[-, n, u_n]} [\#, n, u_n] \ldots [\#, n, u_n][v_{m,1}, n, u_n] \ldots \\
&\quad [v_{m,n-1}, n, u_n] \overline{[+, n, u_n]} [\$, n, u_n] \ldots [u_{n-1}, n, u_n][u_n, n] \lhd =: k_1.
\end{aligned}$$

1.3. Now that the information on the letter u_n has been sent to all other letters, each auxiliary letter of the form $\overline{[c, n, u_n]}$ is rewritten into the letter $[c, n, u_n]$ proceeding from left to right. In this way the restart configuration k_1 is transformed into the configuration

$$\begin{aligned} k_2 := \rhd[v_{1,1}, n, u_n] \ldots [v_{1,n-1}, n, u_n][-, n, u_n][\#, n, u_n] \ldots \\ [\#, n, u_n][v_{m,1}, n, u_n] \ldots [v_{m,n-1}, n, u_n][+, n, u_n][\$, n, u_n] \ldots \\ [u_{n-3}, n, u_n][u_{n-2}, n, u_n][u_{n-1}, n, u_n][u_n, n]\lhd, \end{aligned}$$

and the window of M_n is moved to the factor $[u_{n-2}, n, u_n][u_{n-1}, n, u_n][u_n, n]$.

2.1. Starting with the above configuration the next phase begins. In this phase the information on the letter u_{n-1} is shifted to the left and compared with the corresponding letters of the syllables $v_1, v_2, \ldots, v_m$. This is done by first rewriting the letter $[u_{n-1}, n, u_n]$ into the letter $[u_{n-1}, n-1]$ and by then moving the information on the pair $(n-1, u_{n-1})$ to the left. In addition, in each syllable the letter immediately to the left of the letter of the form $[c, n, u_n]$ is compared to u_{n-1}, and the result is encoded in the same way as above. However, during this phase, a letter of the form $[d, n, b]$ is replaced by $\overline{[-, n-1, d]}$ if its right-hand neighbor is of the form $[-, n-1, d]$. In this way the negative result of the previous test, which is stored in the latter letter, is also carried over to the present letter, even if the test for the new letter d would return a positive result.

Using these transitions the configuration k_2 is transformed into the restarting configuration

$$\begin{aligned} \rhd[v_{1,1}, n-1, u_{n-1}] \ldots \overline{[-, n-1, u_{n-1}]}[-, n-1, u_{n-1}][\#, n-1, u_{n-1}] \ldots \\ [\#, n-1, u_{n-1}][v_{m,1}, n-1, u_{n-1}] \ldots \overline{[+, n-1, u_{n-1}]}[+, n-1, u_{n-1}] \\ [\$, n-1, u_{n-1}] \ldots [u_{n-2}, n-1, u_{n-1}][u_{n-1}, n-1][u_n, n] \lhd . \end{aligned}$$

2.2. Before the next phase can start, M_n replaces the letters $\overline{[a, n-1, d]}$ by $[a, n-1, d]$, proceeding from left to right, which yields the following configuration:

$$\begin{aligned} \rhd\, [v_{1,1}, n-1, u_{n-1}] \ldots [-, n-1, u_{n-1}][-, n-1, u_{n-1}][\#, n-1, u_{n-1}] \\ \ldots [\#, n-1, u_{n-1}][v_{m,1}, n-1, u_{n-1}] \end{aligned}$$

$$\begin{aligned}&\ldots[+,n-1,u_{n-1}][+,n-1,u_{n-1}][\$,n-1,u_{n-1}]\\&\ldots[u_{n-4},n-1,u_{n-1}][u_{n-3},n-1,u_{n-1}][u_{n-2},n-1,u_{n-1}]\\&[u_{n-1},n-1][u_n,n]\lhd,\end{aligned}$$

where the window of M_n contains the factor $[u_{n-3},n-1,u_{n-1}][u_{n-2},n-1,u_{n-1}][u_{n-1},n-1]$.

3.1. In the following $n-3$ phases, the letters $u_{n-2},\ldots,u_2$ are moved to the left using corresponding transitions. These phases yield the following restarting configuration:

$$\begin{aligned}k_3 := {}&\rhd[v_{1,1},2,u_2][-,2,u_2]\ldots[-,2,u_2][\#,2,u_2]\ldots[\#,2,u_2]\\&[v_{m,1},2,u_2][+,2,u_2]\ldots\\&[+,2,u_2][\$,2,u_2][u_1,2,u_2][u_2,2][u_3,3]\ldots[u_n,n]\lhd .\end{aligned}$$

3.2 In the final phase, the letter u_1 is moved to the left and compared to the first letter of the other syllables. Accordingly, the letter $[u_1,2,u_2]$ is rewritten into the letter $[u_1,1]$, the information on the pair $(1,u_1)$ is sent to the left, and the first letter of each other syllable is compared to u_1. This is done in such a way that it ensures that the computation of M will get stuck on reaching a syllable that is not of the required length n. These transitions are defined as follows, where $a_1,a_2,b,d\in\{0,1\}$ and $c,c',c''\in\{+,-\}$:

$$\begin{aligned}\delta_n([\$,2,b][a_1,2,a_2][a_2,2]) &= [a_1,1],\\ \delta_n([c,2,b][\$,2,b][a_1,1]) &= [\$,1,a_1],\\ \delta_n([c,2,b][c',2,b][\$,1,d]) &= [c',1,d],\\ \delta_n([c,2,b][c',2,b][c'',1,d]) &= [c',1,d],\\ \delta_n([a_1,2,b][c,2,b][c',1,d]) &= [c,1,d],\\ \delta_n([\#,2,b][d,2,b][+,1,d]) &= [+,1,d],\\ \delta_n([\#,2,b][a_1,2,b][+,1,d]) &= [-,1,d] \quad \text{for } a_1\neq d,\\ \delta_n([\#,2,b][a_1,2,b][-,1,d]) &= [-,1,d],\end{aligned}$$

$$\begin{aligned}
\delta_n([c,2,b][\#,2,b][c',1,d]) &= [\#,1,d],\\
\delta_n([c,2,b][c',2,b][\#,1,d]) &= [c',1,d],\\
\delta_n(\rhd[d,2,b][+,1,d]) &= [+,1,d],\\
\delta_n(\rhd[a_1,2,b][+,1,d]) &= [-,1,d] \quad \text{for } a_1 \neq d,\\
\delta_n(\rhd[a_1,2,b][-,1,d]) &= [-,1,d].
\end{aligned}$$

Using these transitions the restarting configuration k_3 is transformed into the restarting configuration

$$\begin{aligned}
&\rhd[-,1,u_1][-,1,u_1]\ldots[-,1,u_1][\#,1,u_1]\ldots[\#,1,u_1]\\
&\quad[+,1,u_1][+,1,u_1]\ldots[+,1,u_1][\$,1,u_1][u_1,1][u_2,2][u_3,3]\ldots[u_n,n]\lhd\,.
\end{aligned}$$

3.3. Finally, M_n scans its tape from left to right. The final tape contents must begin with a letter of the form $[c,1,b]$ for some $c \in \{+,-\}$, and M_n accepts as soon as it detects a factor of the form $\rhd[+,1,b]$ or $\#[+,1,b]$.

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

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

(a) $w = v_1\#v_2\#\ldots\#v_m\$u$ for some $m \geq 1$ and some words $v_1, v_2, \ldots, v_m, u \in \{0,1\}^*$.

(b) $|u| = n$.

(c) $|v_1| = |v_2| = \ldots = |v_m| = n$.

(d) There exists an index $i \in \{1,2,\ldots,m\}$ such that $v_i = u$ holds.

Proof of (a) and (b). In each phase i the rewriting process is initiated by rewriting the $(n+1-i)$-th letter $a \in \{0,1\}$ from the right into the letter $[a, n+1-i]$. In phase 1 this is ensured by the corresponding transition, as a letter of the form $[a,n]$ can only be produced immediately to the left of the right sentinel $\lhd$, and for the other phases this is ensured as a letter of the form $[a,i]$

can only be produced immediately to the left of a letter of the form $[a', i+1]$. Finally, a letter of the form $[a, 1]$ can only be produced when additionally its left neighbor is a letter of the form $[\$, 2, b]$. It follows that w must end with a suffix of the form $\$u$, where $u \in \{0,1\}^n$.

Information of the form $(1, b)$ is sent left across the letter \$ only if the right neighbor has the form $[a, 1]$. Thus, if there are any additional occurrences of the letter \$ in w, then this information does not reach the first letter on the tape, which means that M_n would not accept. Hence, w is of the form as described in (a). □

Proof of (c). If a syllable v_i is of length larger than n, then its n-th letter from the right cannot be rewritten into a letter of the form $[c, 1, b]$, as its left neighbor is not the letter $\#$. On the other hand, if $|v_i| = s < n$, then after phase s, all letters of v_i will have been replaced by letters of the form $[c, s, b]$ for some $c \in \{+, -\}$, and then, in phase $s+1$, there would be no letter left for the corresponding check, that is, this phase would get stuck at this syllable. Thus, (c) follows. □

Proof of (d). Each syllable v_i is compared to u by proceeding from right to left. As soon as a mismatch is found, the corresponding letter of v_i is replaced by a letter of the form $[-, i, b]$. Now in the following phases the letter $-$ is propagated left through this syllable, which means that its first letter will be replaced by a letter of this very form. Thus, only if $v_i = u$ holds, then the first letter of v_i is eventually rewritten into a letter of the form $[+, 1, b]$. As M_n accepts only in this case, we see that (d) holds, too. □

Thus, we see that indeed $L(M_n) = B_n$. This completes the proof of Proposition 3.5.9. □

On the other hand, we have the following lower bound results on B_n.

Lemma 3.5.10. *(a) If $A = (Q, \Sigma, q_0, F, \delta)$ is a DFA such that $L(A) = B_n$, then $|Q| \geq 2^{2^n}$.*

(b) If $C = (Q, \Sigma, q_0, F, \delta)$ is an NFA such that $L(C) = B_n$, then $|Q| \geq 2^n$.

Proof. (a) Let $V = \{v_1, v_2, \ldots, v_m\}$ be a non-empty subset of $\{0,1\}^n$. Without loss of generality we can assume that $v_1 \leq v_2 \leq \ldots \leq v_m$ holds, where $\leq$ is the lexicographical ordering on $\{0,1\}^n$. With V we associate the word

$w_V = v_1\#v_2\#\ldots\#v_m\$$, and further, we take $q_V = \delta(q_0, w_V)$, that is, q_V is the state that A reaches from its initial state q_0 after reading the word w_V. As the word $w_V v_1 \in B_n$, the state q_V exists, and obviously, $q_V \neq q_0$.

Now, let V_1 and V_2 be two non-empty subsets of $\{0,1\}^n$ such that $V_1 \neq V_2$. Then there exists a word $u \in \{0,1\}^n$ that belongs to the symmetric difference of V_1 and V_2. Without loss of generality, we assume that $u \in V_1$, and hence, $u \notin V_2$. Then $w_{V_1}u \in B_n$, but $w_{V_2}u \notin B_n$. Hence, it follows that the states q_{V_1} and q_{V_2} must not be identical. As there are $2^{2^n} - 1$ different non-empty subsets of $\{0,1\}^n$, we see that the set $\{\, q_V \mid \emptyset \neq V \subseteq \{0,1\}^n \,\} \subseteq Q \smallsetminus \{q_0\}$ contains $2^{2^n} - 1$ many different states. Thus, $|Q| \geq 2^{2^n}$ follows.

(b) We can easily prove the lower bound by contradiction. If we had an NFA with $|Q| < 2^n$ states, we could use the powerset construction to get a DFA with $|Q| < 2^{2^n}$ states for the language B_n which does not exist as we have shown in the paragraph above.

The lower bound can also be shown by using the technique of *fooling sets* from [4] (or also [14]). Let $X = \{0,1\}^n$, and for each $x \in X$, let $u_x = x\$$ and $v_x = x$. Then, for all $x \in X$, we have $u_x v_x = x\$x \in B_n$, but for all $x, y \in X$, if $x \neq y$, then $u_x v_y = x\$y \notin B_n$. It follows that $|Q| \geq |X| = 2^n$ by the lemma on p. 188 of [4]. □

In combination with Theorem 3.5.8, the results on B_n yield the following consequence.

Corollary 3.5.11. *For converting a stl-(det-)ORWW-automaton with n letters into an equivalent NFA, $2^{\mathcal{O}(n)}$ states are sufficient, and there are cases in which these many states are also necessary.*

For the conversion into DFAs, we obtain the following result.

Corollary 3.5.12. *For converting a stl-det-ORWW-automaton with n letters into an equivalent DFA, $2^{2^{\mathcal{O}(n)}}$ states are sufficient, and there are cases in which these many states are also necessary.*

This is a clear improvement over the upper bound of $2^{2^{\mathcal{O}(n^2 \log n)}}$ given in Corollary 8 (a) of [34]. Actually, the lower bound expressed by Corollary 3.5.11 even holds in the unary case. To see this, we consider the family of languages $(U_n)_{n\geq 3}$, where $U_n = \{a^{2^n}\}$. Obviously, an NFA for the language U_n needs at least $2^n + 1$ states. However, there exists a stl-det-ORWW-automaton with only $\mathcal{O}(n)$ letters that accepts this language.

Lemma 3.5.13. *For each $n \geq 3$, the language U_n is accepted by a stl-det-ORWW-automaton that works on an alphabet of size $3n - 1$.*

Proof. For $n \geq 3$, let

$$\begin{aligned}\Gamma_n &= \{a, [1,+,-], [1,-,+], [n,-,-], [n,-,+]\} \\ &\quad \cup \{\, [i,+,-], [i,-,+], [i,-,-] \mid i = 2, 3, \ldots, n-1 \,\},\end{aligned}$$

that is, $|\Gamma_n| = 3n - 1$. The stl-det-ORWW-automaton M_n will process a given input in n phases. In phase i, the letters of the form $[i, r, s]$ are used to rewrite the current tape content from right to left, letter by letter. The index r indicates whether the current position corresponds to an input letter that has already been marked as deleted ($r = -$) or not ($r = +$), and the index s indicates whether the next undeleted letter to the left must be deleted ($s = -$) or whether it is to be maintained as undeleted ($s = +$). These indices are used in such a way that in each phase, every second undeleted position is kept, and the other previously undeleted positions are marked as 'deleted'. Finally, in phase n, exactly one previously undeleted position is marked as 'deleted', and M_n accepts if the first position is the second previously undeleted position from the right in phase n. Accordingly, $M_n = (\{a\}, \Gamma_n, \rhd, \lhd, \delta_n, >)$ is obtained by taking the ordering

$$\begin{array}{rcll} a & > & [i,r,s] & \text{for all } i \geq 1 \text{ and } r, s \in \{+,-\}, \\ [i,r,s] & > & [i+1, r', s'] & \text{for all } i = 1, 2, \ldots, n-1 \text{ and all } r, s, r', s' \in \{+,-\}, \end{array}$$

and by defining the transition function as follows, where $r, s, r', s', r'', s'' \in \{+,-\}$:

$$\begin{array}{lcl} \delta(\rhd aa) & = & \mathsf{MVR}, \\ \delta(aaa) & = & \mathsf{MVR}, \\ \delta(aa\lhd) & = & [1,-,+], \\ \delta(aa[1,-,+]) & = & [1,+,-], \\ \delta(aa[1,+,-]) & = & [1,-,+], \end{array}$$

$$\begin{array}{llll}
\delta(\triangleright a[1,-,+]) & = & [1,+,-], & \\
\delta(\triangleright[i,r,s][i,r',s']) & = & \mathsf{MVR} & \text{for all } 1 \le i \le n-1, \\
\delta([i,r,s][i,r',s'][i,r'',s'']) & = & \mathsf{MVR} & \text{for all } 1 \le i \le n-1, \\
\delta([i,r,s][i,-,s']\triangleleft) & = & [i+1,-,-] & \text{for all } 1 \le i \le n-1, \\
\delta([i,r,s][i,-,s'][i+1,r'',s'']) & = & [i+1,-,s''] & \text{for all } 1 \le i \le n-1, \\
\delta([i,r,s][i,+,s'][i+1,r'',-]) & = & [i+1,-,+] & \text{for all } 1 \le i \le n-1, \\
\delta([i,r,s][i,+,s'][i+1,r'',+]) & = & [i+1,+,-] & \text{for all } 1 \le i \le n-2, \\
\delta(\triangleright[i,+,-][i+1,r'',+]) & = & [i+1,+,-] & \text{for all } 1 \le i \le n-2, \\
\delta(\triangleright[n-1,+,-][n,-,+]) & = & \mathsf{Accept}. &
\end{array}$$

For example, for $n = 3$, M_3 executes the following computation on input $w = a^8$:

$$\begin{array}{r}
\triangleright aaaaaaaa\triangleleft \vdash^c \triangleright aaaaaaa[1,-,+]\triangleleft \vdash^{c^7} \\
\triangleright[1,+,-][1,-,+][1,+,-][1,-,+][1,+,-][1,-,+][1,+,-][1,-,+]\triangleleft \vdash^{c^8} \\
\triangleright[2,+,-][2,-,+][2,-,+][2,-,-][2,+,-][2,-,+][2,-,+][2,-,-]\triangleleft \vdash^{c^7} \\
\triangleright[2,+,-][3,-,+][3,-,+][3,-,+][3,-,+][3,-,-][3,-,-][3,-,-]\triangleleft \vdash \\
\mathsf{Accept}.
\end{array}$$

It follows that $L(M_n) = U_n$. □

Every two-way NFA for U_n has at least $2^n + 1$ states [5]. Thus, also the trade-off for converting a stl-det-ORWW-automaton into an equivalent two-way NFA is exponential.

Finally, we show that stateless ORWW-automata can describe some (regular) languages much more succinctly than stateless det-ORWW-automata.

Let $\Sigma = \{a, b, \#\}$, and let, for $n \ge 1$, C_n denote the following language over Σ:

$$C_n = \{\, u_1\#u_2\#\ldots\#u_m \mid m \ge 2, u_1, u_2, \ldots, u_m \in \{a,b\}^n, \exists i < j : u_i = u_j \,\}.$$

Lemma 3.5.14. *The language C_n is accepted by a stl-ORWW-automaton M_n with a tape alphabet of size $20n + 1$.*

Proof. In Proposition 3.5.9 a stl-det-ORWW-automaton A_n with $18n$ letters is given that accepts the language

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

The stl-ORWW-automaton M_n is obtained from A_n by adding the $2n+1$ letters $a'_1, a'_2, \ldots, a'_n, b'_1, b'_2, \ldots, b'_n$, and $\#'$. It proceeds as follows.

It first marks the letters from right to left. In a syllable $v = v^{(1)}v^{(2)}\ldots v^{(n)}$, the last letter is replaced by $v_n^{(n)'}$, $v^{(n-1)}$ is replaced by $v_{n-1}^{(n-1)'}$, and so forth until finally $v^{(1)}$ is replaced by $v_1^{(1)'}$, and then the preceding letter $\#$ is replaced by $\#'$. In this way it is ensured that all $\{a,b\}$-syllables have length n. This continues until M_n nondeterministically chooses to replace a symbol $\#$ by the symbol \$. In this way the input of the form $u_1\#u_2\#\ldots\#u_k\#u_{k+1}\#\ldots\#u_m$ is transformed into a word of the form $u_1\#u_2\#\ldots\#u_k\$\hat{u}_{k+1}\#'\ldots\#'\hat{u}_m$, where $\hat{u}_i$ $(k+1 \leq i \leq m)$ denotes the word that is obtained from $u_i \in \{a,b\}^n$ by the above marking process. Now on the prefix $u_1\#u_2\#\ldots\#u_k\$\hat{u}_{k+1}$, M_n simulates the stl-det-ORWW-automaton A_n, and it accepts if the computation of A_n being simulated accepts. It is obvious now that $L(M_n) = C_n$. □

Now we claim that every stl-det-ORWW-automaton for C_n needs at least $2^{\mathcal{O}(n)}$ letters, that is, the above presentation by a stl-ORWW-automaton is exponentially more succinct than any presentation by a stl-det-ORWW-automaton.

Proposition 3.5.15. *Let $s : \mathbb{N} \to \mathbb{N}$ be a function such that, for each $n \geq 1$, there exists a stl-det-ORWW-automaton $D_n = (\Sigma, \Gamma_n, \rhd, \lhd, \delta_n, >)$ such that $L(D_n) = C_n$ and $|\Gamma_n| \leq s(n)$. Then $s(n) \notin o(2^n)$.*

Proof. Let $n \geq 1$, and let $D_n = (\Sigma, \Gamma_n, \rhd, \lhd, \delta_n, >)$ be a stl-det-ORWW-automaton such that $L(D_n) = C_n$ and $|\Gamma_n| \leq s(n)$. As D_n is deterministic, we obtain a stl-det-ORWW-automaton $E_n = (\Sigma, \Gamma_n, \rhd, \lhd, \eta_n, >)$ such that $L(E_n) = C_n^c = \Sigma^* \smallsetminus D_n$ simply by interchanging accept steps with undefined steps (see Proposition 3.3.3). From E_n we can construct an NFA F_n of size $2^{r \cdot s(n)}$ for C_n^c, where $r \in \mathbb{N}_+$ is a constant.

We now present a large fooling set for C_n^c. Let A be a subset of $\{a,b\}^n$ of size 2^{n-1}. Then also the set $\bar{A} = \{a,b\}^n \smallsetminus A$ has size 2^{n-1}. With these sets we associate the following languages:

$$\begin{array}{rcll} P_A & = & \{ u_1\#u_2\#\ldots\#u_{2^{n-1}} & \mid \; A = \{u_1, u_2, \ldots, u_{2^{n-1}}\} \} \text{ and} \\ Q_A & = & \{ \#v_1\#v_2\#\ldots\#v_{2^{n-1}} & \mid \; \bar{A} = \{v_1, v_2, \ldots, v_{2^{n-1}}\} \}. \end{array}$$

Then $uv \in C_n^c$ for all $u \in P_A$ and all $v \in Q_A$. On the other hand, if B is a subset of $\{a,b\}^n$ of size 2^{n-1} such that $A \neq B$, then $uv \in C_n$ for all $u \in P_A$ and all $v \in Q_B$, because there is a word $x \in \{a,b\}^n$ such that $x \in A$ and $x \in \bar{B}$. Hence, by choosing a pair $(u_A, v_A) \in P_A \times Q_A$ for all subsets A of $\{a,b\}^n$ of size 2^{n-1}, we obtain a fooling set for F_n of size $\binom{2^n}{2^{n-1}} = \frac{(2^n)!}{(2^{n-1})! \cdot (2^{n-1})!} > 2^{2^{n-1}}$. This implies by [4] that the number $2^{r \cdot s(n)}$ of states of the NFA F_n satisfies the inequality $2^{r \cdot s(n)} \geq 2^{2^{n-1}}$, that is, $r \cdot s(n) \geq 2^{n-1}$. Hence, $\frac{s(n)}{2^n} \geq \frac{1}{2r}$, which clearly shows that $\liminf_{n\to\infty} \frac{s(n)}{2^n} \geq \frac{1}{2r} > 0$, that is, $s(n) \notin o(2^n)$. □

From the proof above we can also derive the following complexity result.

Corollary 3.5.16. *Let $s : \mathbb{N} \to \mathbb{N}$ be a function such that, for each $n \geq 1$, there exists a stl-ORWW-automaton $E_n = (\Sigma, \Gamma_n, \rhd, \lhd, \delta_n, >)$ such that $L(E_n) = C_n^c$ and $|\Gamma_n| \leq s(n)$. Then $s(n) \notin o(2^n)$.*

Proof. By Theorem 3.5.1 we can construct an NFA F_n of size $2^{r \cdot s(n)}$ for C_n^c from E_n. Now the proof of Proposition 3.5.15 shows that $s(n) \notin o(2^n)$. □

As by Lemma 3.5.14 the language C_n is accepted by a stl-ORWW-automaton M_n with a tape alphabet of size $20n + 1$, Corollary 3.5.16 shows that the conversion of a stl-ORWW-automaton of size n into a stl-ORWW-automaton for the complement of $L(M)$ can actually increase the alphabet size exponentially.

Now we use the construction from Theorem 3.5.8 to solve some decision problems.

3.6 Decision Problems

As the stateless ORWW-automata exactly describe the regular languages and can be converted into NFAs all decision problems of interest are decidable. However, we can still ask ourselves how difficult it is to answer these decision problems. In this section we will show that many decision problems are PSPACE-complete, that is, they can be answered by using an amount of memory that is polynomial in the input length, and that every other problem that can be solved in polynomial space can be reduced to them in polynomial time.

The *emptiness problem* for NFAs is the problem of deciding whether the language accepted by a given NFA is empty. If $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA of size $|Q| = m$, then emptiness of $L(A)$ can be decided nondeterministically using space $\mathcal{O}(\log m)$ [18] by checking whether there is a path in the state graph $G(A)$ of A that leads from the node corresponding to the initial state q_0 to a node corresponding to an accepting state. Below we will use this result to show that the emptiness problem for stl-ORWW-automata is decidable in polynomial space. It is the generalization of the theorem for stl-det-ORWW-automata from [24].

Theorem 3.6.1. *The* emptiness problem *for stl-ORWW-automata is PSPACE-complete.*

Proof. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-ORWW-automaton such that $|\Gamma| = n$. By Theorem 3.5.8, there exists an NFA $A = (Q, \Sigma, \Delta, S, \{q_+\})$ of size at most $2^{13\cdot(n+1)}$ such that $L(A) = L(M)$. A nondeterministic Turing machine T can now check whether $L(A)$ is empty by proceeding as follows. It starts by checking whether the state q_+ is contained in the initial set S. In the affirmative T halts and accepts, as $\lambda \in L(A)$. In the negative it guesses a state $\alpha \in S$ by guessing a pattern which has the left sentinel $\rhd$ at position 1. This state is stored as 'current state' st_c. It then checks whether $q_+ \in \Delta(st_c, a)$, where a is the letter at position 2 of the current pattern st_c. In the affirmative T halts and accepts. Otherwise it guesses some pattern β, which it stores as 'next state' st_n, and then it checks whether $st_n \in \Delta(st_c, a)$. This is done by using the transition function δ of M and the definition of Δ. In the negative T halts without accepting, while in the affirmative it replaces st_c by st_n, in this way simulating a transitional step of A. This process continues until either T halts (accepting or not), or until $2^{13\cdot(n+1)}$ many transition steps of A have been simulated. As the NFA A has $|Q| \leq 2^{13\cdot(n+1)}$ many states, it is clear that a shortest path from its initial state to its final state has at most length $|Q|$, and so T halts without accepting after this many steps.

Now, it is rather clear that T accepts given the stl-ORWW-automaton M as input if and only if $L(M) = L(A)$ is non-empty. For performing the above computation, T needs to store two states st_c and st_n of A and a counter c that it uses to count the number of steps of A it has been simulating. Thus, we see that T just needs space $13 \cdot (n+1)$ for the counter and space $\mathcal{O}(n \cdot \log n)$ for the two states of A. Hence, the emptiness problem for stl-ORWW-automata is decidable nondeterministically within polynomial space, and therewith by Savitch's Theorem [42], it is decidable deterministically within polynomial space, that is, stl-ORWW-Emptiness $\in$ PSPACE.

Now let $A_1, A_2, \ldots, A_t$ be DFAs over a common input alphabet Σ of size k such that A_i has n_i states, $1 \leq i \leq t$. From these DFAs we can construct a stl-det-ORWW-automaton M with a tape alphabet of size $k \cdot (1+n_1+\ldots+n_{t-1})+n_t$ such that $L(M) = \bigcap_{i_1}^{t} L(A_i)$ by Corollary 3.3.7. Hence, M has at most $\mathcal{O}((k \cdot (1+n_1+\ldots+n_{t-1})+n_t)^3) = O((k \cdot \sum_{i=1}^{t} n_i)^3)$ many transitions, and so it can be computed from $A_1, A_2, \ldots, A_t$ in polynomial time.

Now $L(M) \neq \emptyset$ iff $L(A_1) \cap L(A_2) \cap \ldots \cap L(A_t) \neq \emptyset$, which shows that the above construction yields a polynomial-time reduction from the DFA-Intersection-Emptiness Problem to stl-det-ORWW-Emptiness. Since a stl-det-ORWW-automaton can also be seen as a stl-ORWW-automaton, we also have a polynomial-time reduction from the DFA-Intersection-Emptiness Problem to stl-ORWW-Emptiness.

As the former is PSPACE-complete (see, e.g., [12]), we see that the latter is PSPACE-hard. Together with the membership in PSPACE shown above, PSPACE-completeness follows. □

The following results on decision problems apply to the stateless nondeterministic as well as to the deterministic ordered restarting automata, because the conversion into an NFA is just as complex for both automata and every deterministic stl-ORWW-automaton can be seen as nondeterministic stl-ORWW-automaton. For the sake of simplicity, we only formulate our results for the deterministic model, as this is the more interesting case.

As stl-det-ORWW-automata can easily be modified to accommodate the Boolean operations (see Proposition 3.3.3, Theorem 3.3.4, and Corollary 3.3.5), the following completeness results are easily derived from Theorem 3.6.1.

Corollary 3.6.2. [24] *For stl-det-ORWW-automata, the universality problem, the finiteness problem, the inclusion problem, and the equivalence problem are PSPACE-complete.*

Proof. Universality: Let M be a stl-det-ORWW-automaton with input alphabet Σ. From M we can construct a stl-det-ORWW-automaton M^c that uses the same tape alphabet as M for the language $L(M^c) = (L(M))^c = \Sigma^* \smallsetminus L(M)$ by Proposition 3.3.3. The automaton M is universal, that is, $L(M) = \Sigma^*$, iff $L(M^c) = \emptyset$. PSPACE-completeness of the universality problem now follows from PSPACE-completeness for the emptiness problem.

Inclusion and equivalence: Let M_1 and M_2 be stl-det-ORWW-automata with alphabets of sizes n_1 and n_2, respectively. From M_1 and M_2 we can construct a stl-det-ORWW-automaton M with an alphabet of size $O(n_1 \cdot n_2)$ in polynomial time such that $L(M) = L(M_1) \cap L(M_2)^c$ (see the proof of Theorem 3.3.4). Now $L(M_1) \subseteq L(M_2)$ iff $L(M_1) \cap L(M_2)^c = \emptyset$ iff $L(M) = \emptyset$. It follows that the inclusion problem is in PSPACE, which in turn implies immediately that the equivalence problem is in PSPACE, too.

On the other hand, let M' be a stl-det-ORWW-automaton that accepts the empty set. Then $L(M) = L(M')$ iff $L(M) \subseteq L(M')$ iff $L(M) = \emptyset$. Thus, PSPACE-completeness of the inclusion and the equivalence problems follows from PSPACE-completeness for the emptiness problem,

Finiteness: Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton. We take a new symbol $\Box$, that is, $\Box \notin \Gamma$, and define a stl-det-ORWW-automaton $M' = (\Sigma', \Gamma', \rhd, \lhd, \delta', >)$ as follows:

- $\Sigma' = \Sigma \cup \{\Box\}$ and $\Gamma' = \Gamma \cup \{\Box\}$,
- the transition function δ' is obtained from δ by simply interpreting an occurrence of the symbol $\Box$ as an occurrence of the right delimiter $\lhd$.

Then $L(M') = L(M) \cup (L(M) \cdot \Box \cdot \Sigma'^*)$, which means that $L(M')$ is finite iff $L(M) = \emptyset$. PSPACE-hardness of finiteness now follows from PSPACE-hardness of the emptiness problem.

On the other hand, from a stl-(det)-ORWW-automaton M with an alphabet of size n we can construct an NFA A of size at most $2^{13\cdot(n+1)}$ such that $L(M) = L(A)$. Just like emptiness, also infiniteness is decidable for A nondeterministically in space $\log(2^{13\cdot(n+1)}) \in \mathcal{O}(n)$, and hence, it is decidable deterministically in space $\mathcal{O}(n^2)$. Thus, finiteness for stl-det-ORWW-automata is indeed in PSPACE, and so it is PSPACE-complete. $\Box$

The finite languages are a proper subclass of the regular languages. Accordingly, the finiteness problem can be seen as a special case of the following decision problem, where $\mathcal{C}$ denotes a subclass of the regular languages:

Membership problem in $\mathcal{C}$:

INSTANCE: A stl-det-ORWW-automaton M.

QUESTION: Does the language $L(M)$ belong to the class $\mathcal{C}$?

In the literature many subfamilies of the regular languages have been studied (see, e.g., [6, 10, 40]). Here we only consider some of them, beginning with the *strictly locally testable languages* of [29, 45].

For a positive integer k and a word w of length $|w| \geq k$, let $P_k(w)$ and $S_k(w)$ be the prefix and the suffix of w of length k, respectively. Further, let $I_k(w)$ be the set of all factors of w of length k except the prefix and suffix of w of length k, that is,

$$I_k(w) = \{\, u \mid |u| = k \text{ and } \exists x, y \in \Sigma^+ : w = xuy \,\}.$$

For example, $P_2(aababab) = aa$, $S_2(aababab) = ab$, and $I_2(aababab) = \{ab, ba\}$. Obviously, if $|w| \leq k+1$, then $I_k(w)$ is empty.

Definition 3.6.3. *Let k be a positive integer. A language $L \subseteq \Sigma^*$ is* strictly k-testable *if there exist finite sets $A, B, C \subseteq \Sigma^k$ such that, for all $w \in \Sigma^*$ satisfying $|w| \geq k$, we have*

$$w \in L \textit{ if and only if } P_k(w) \in A,\ S_k(w) \in B,\ \textit{ and } I_k(w) \subseteq C.$$

In this case, (A, B, C) is called a triple *for L. The language L is called* strictly locally testable *if it is strictly k-testable for some $k \geq 1$.*

Note that the definition of 'strictly k-testable' says nothing about the words of length $k-1$ or less. Hence, L is strictly k-testable if and only if

$$L \cap \Sigma^k \cdot \Sigma^* = (A \cdot \Sigma^* \cap \Sigma^* \cdot B) \smallsetminus (\Sigma^+ \cdot (\Sigma^k \smallsetminus C) \cdot \Sigma^+) \tag{3.1}$$

for some finite sets $A, B, C \subseteq \Sigma^k$. For example, the language $(a+b)^*$ is strictly 1-testable, as $(a+b)^+$ can be expressed in the form (3.1) by $(A, B, C) = (\{a, b\}, \{a, b\}, \{a, b\})$, and the language $a(baa)^+$ is strictly 3-

testable, as it can be expressed in the form (3.1) by the triple $(A, B, C) = (\{aba\}, \{baa\}, \{aba, baa, aab\})$. On the other hand, the language $(aa)^*$ is not strictly locally testable.

We denote the family of strictly k-testable languages by k-SLT and the class of strictly locally testable languages by SLT. It is known that

$$k\text{-}\mathsf{SLT} \subsetneq (k+1)\text{-}\mathsf{SLT} \subsetneq \mathsf{SLT} \subsetneq \mathsf{REG}.$$

For each $k \geq 1$, if a language L is given through a DFA, then it is decidable in polynomial time whether or not L is strictly locally k-testable. In fact, the following problem is solvable in polynomial time [9]:

INSTANCE: A DFA A.
QUESTION: Is the language $L(A)$ strictly locally testable?

Here we are interested in the corresponding variant of this problem in which the language considered is given through a stl-det-ORWW-automaton.

Theorem 3.6.4. [24] *The following problem is PSPACE-complete for each* $k \geq 1$*:*

INSTANCE: A stl-det-ORWW-automaton M*.*
QUESTION: Is the language $L(M)$ *strictly locally* k*-testable?*

Proof. Let $k \geq 1$, and let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton. The language $L = L(M)$ is strictly locally k-testable iff there are subsets $A, B, C \subseteq \Sigma^k$ such that $L \cap (\Sigma^k \cdot \Sigma^*) = (A \cdot \Sigma^* \cap \Sigma^* \cdot B) \cap (\Sigma^+ \cdot (\Sigma^k \smallsetminus C) \cdot \Sigma^+)^c$. From A, B, and C, a DFA can be constructed for the language $(A \cdot \Sigma^* \cap \Sigma^* \cdot B) \cap (\Sigma^+ \cdot (\Sigma^k \smallsetminus C) \cdot \Sigma^+)^c$, and hence, we can construct a stl-det-ORWW-automaton $M'(A, B, C)$ for the language

$$(L(M) \cap \Sigma^{\leq k-1}) \cup \left((A \cdot \Sigma^* \cap \Sigma^* \cdot B) \cap (\Sigma^+ \cdot (\Sigma^k \smallsetminus C) \cdot \Sigma^+)^c\right).$$

Now L is strictly locally k-testable iff there are $A, B, C \subseteq \Sigma^k$ such that the stl-det-ORWW-automata M and $M'(A, B, C)$ are equivalent. As the integer k is fixed, the latter can be decided in polynomial space by Corollary 3.6.2.

It remains to be shown that the above problem is PSPACE-hard. This will be done by a reduction from the emptiness problem for stl-det-ORWW-automata. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton, and let

$\Sigma' = \{a, \Box\}$ and $\Gamma' = \Sigma' \cup \{\Box_1, \Box_2\}$ be new alphabets such that Γ' and Γ are disjoint. Without loss of generality we can assume that M does not accept the empty input, that is, $\delta(\rhd\lhd) \neq \mathsf{Accept}$, and that M only accepts with its window scanning the right sentinel $\lhd$. Obviously, the language $L' = a^+$ is strictly locally 1-testable, and therewith it is strictly locally k-testable. We now construct a stl-det-ORWW-automaton $M' = (\Sigma \cup \Sigma', \Gamma \cup \Gamma', \rhd, \lhd, \delta', >')$ for the language $L(M') = L' \cup L(M) \cdot (\Box\Box)^+$ as follows:

- for $c, d \in \Gamma \cup \Gamma'$, $c >' d$ iff $c, d \in \Gamma$ and $c > d$, or $c = \Box$ and $d \in \{\Box_1, \Box_2\}$,
- and the transition function δ' is defined by

$$\begin{array}{rlcll}
(1) & \delta'(\rhd a \lhd) & = & \mathsf{Accept}, & \\
(2) & \delta'(\rhd aa) & = & \mathsf{MVR}, & \\
(3) & \delta'(aaa) & = & \mathsf{MVR}, & \\
(4) & \delta'(aa\lhd) & = & \mathsf{Accept}, & \\
(5) & \delta'(\rhd cd) & = & \delta(\rhd cd) & \text{for all } c, d \in \Gamma, \\
(6) & \delta'(cde) & = & \delta(cde) & \text{for all } c, d, e \in \Gamma, \\
(7) & \delta'(cd\Box) & = & \delta(cd\lhd), & \text{if } c \in \Gamma \cup \{\rhd\},\ d \in \Gamma, \\
 & & & & \ \text{and } \delta(cd\lhd) \neq \mathsf{Accept}, \\
(8) & \delta'(cd\Box) & = & \mathsf{MVR}, & \text{if } c \in \Gamma \cup \{\rhd\},\ d \in \Gamma, \\
 & & & & \ \text{and } \delta(cd\lhd) = \mathsf{Accept}, \\
(9) & \delta'(d\Box\Box) & = & \Box_1 & \text{for all } d \in \Gamma, \\
(10) & \delta'(cd\Box_1) & = & \mathsf{MVR} & \text{for all } c, d \in \Gamma, \\
(11) & \delta'(d\Box_1\Box) & = & \mathsf{MVR} & \text{for all } d \in \Gamma, \\
(12) & \delta'(\Box_1\Box\Box) & = & \Box_2, & \\
(13) & \delta'(\Box_1\Box\lhd) & = & \Box_2, & \\
(14) & \delta'(d\Box_1\Box_2) & = & \mathsf{MVR} & \text{for all } d \in \Gamma, \\
(15) & \delta'(\Box_1\Box_2\Box) & = & \mathsf{MVR}, & \\
(16) & \delta'(\Box_1\Box_2\lhd) & = & \mathsf{Accept}, & \\
(17) & \delta'(\Box_2\Box\Box) & = & \Box_1, & \\
(18) & \delta'(\Box_1\Box_2\Box_1) & = & \mathsf{MVR}, & \\
(19) & \delta'(\Box_2\Box_1\Box) & = & \mathsf{MVR}, & \\
(20) & \delta'(\Box_2\Box_1\Box_2) & = & \mathsf{MVR}. &
\end{array}$$

Using instructions (1) to (4), the stl-det-ORWW-automaton M' accepts all words from the language L'. Using instructions (5) to (8), M' simulates M on a tape content of the form $w\Box^i$ for $w \in \Gamma^+$ and $i \geq 1$. If M accepts the tape content w, then it does so with its window containing the last two letters of w and the sentinel $\lhd$. In the corresponding situation M' will scan the last two letters of w and the first $\Box$-letter. It will then rewrite $\Box^i$ into $\Box_1\Box_2\Box_1\ldots$ from left to right, and it will only accept if i is an even positive integer. It follows that $L(M') = L' \cup L(M) \cdot (\Box\Box)^+$. Thus, if $L(M) = \emptyset$, then $L(M') = L'$ is strictly locally k-testable, but if $L(M) \neq \emptyset$, then $L(M') = L' \cup L(M) \cdot (\Box\Box)^+$ is not strictly locally testable at all. Hence, the above is a reduction from the emptiness problem for M to the problem of deciding strictly locally k-testability for M'. As M' is easily obtained from M, this shows our intended PSPACE-hardness result. $\Box$

The construction in the proof above shows that the problem of deciding strictly locally testability is at least PSPACE-hard for stl-det-ORWW-automata, but it remains open whether this problem is in PSPACE.

Next we consider the property of being nilpotent. A language $L \subseteq \Sigma^*$ is called *nilpotent* if L is finite or if $L^c = \Sigma^* \smallsetminus L$ is finite.

Theorem 3.6.5. *The following problem is PSPACE-complete:*

INSTANCE: *A stl-det-ORWW-automaton M.*
QUESTION: *Is the language $L(M)$ nilpotent?*

Proof. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton. From M we can immediately obtain a stl-det-ORWW-automaton $M^c = (\Sigma, \Gamma, \rhd, \lhd, \delta^c, >)$ such that $L(M^c) = (L(M))^c$ (Proposition 3.3.3). Now $L(M)$ is nilpotent iff $L(M)$ is finite or $L(M^c)$ is finite. These conditions, however, can be checked in polynomial space by Corollary 3.6.2. Thus, the problem of deciding nilpotency is in PSPACE.

PSPACE-hardness will now be proved by a reduction from the emptiness problem. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton, and let $\Box$ and $\#$ be two new symbols. We define a stl-det-ORWW-automaton $M' = (\Sigma', \Gamma', \rhd, \lhd, \delta', >)$ as follows:

- $\Sigma' = \Sigma \cup \{\Box, \#\}$ and $\Gamma' = \Gamma \cup \{\Box, \#\}$,
- the transition function δ' is obtained from δ by simply interpreting an occurrence of the symbol $\Box$ as an occurrence of the right sentinel $\lhd$.

Then M' is easily obtained from M, and $L(M') = L(M) \cup (L(M) \cdot \square \cdot \Sigma'^*)$. Thus, if $L(M)$ is empty, then $L(M')$ is empty, and therewith, it is nilpotent, but if $L(M)$ is non-empty, then $L(M')$ is infinite. Further, in this case, $(L(M')^c$ is infinite as well, as it contains all words of the form $\#^i$ $(i \geq 1)$. Hence, $L(M')$ is nilpotent iff $L(M)$ is empty. It follows that the problem of deciding nilpotency is PSPACE-complete for stl-det-ORWW-automata. $\square$

A regular language $L \subseteq \Sigma^*$ is called *combinatorial* if $L = \Sigma^* \cdot H$ for some $H \subseteq \Sigma$.

Theorem 3.6.6. *The following problem is PSPACE-complete:*

INSTANCE: *A stl-det-ORWW-automaton M.*
QUESTION: *Is the language $L(M)$ combinatorial?*

Proof. Given a stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$, one can construct a DFA A_H for the language $\Sigma^* \cdot H$, where H is a subset of Σ. Now $L(M)$ is combinatorial, if and only if $L(M) = L(A_H)$ for some subset H of Σ. It follows that it is decidable in polynomial space whether or not $L(M)$ is combinatorial.

Now let M be a given stl-det-ORWW-automaton with input alphabet Σ. We take $\Sigma' = \Sigma \cup \{\square\}$, and define a stl-det-ORWW-automaton $M' = (\Sigma', \Gamma', \rhd, \lhd, \delta', >)$ that proceeds as follows:

1. First M' scans its input from left to right. If the input ends with a letter a of Σ, then it accepts immediately.

2. Otherwise, it rewrites its input from right to left, letter by letter, by replacing each letter $a \in \Sigma'$ by a specific copy $a' \in \Gamma'$.

3. Then M' simulates M, interpreting, for each $a \in \Sigma$, each occurrence of a' as M would interpret an occurrence of the letter a, and interpreting an occurrence of the letter $\square'$ as M would interpret the right sentinel $\lhd$.

4. If the simulated computation of M accepts, then M' checks that the current tape content is of the form $\alpha(\square'\square')^r$ for some $\alpha \in (\Gamma' \smallsetminus \{\square, \square'\})^*$ and some $r \geq 1$ by rewriting the suffix $(\square'\square')^r$ from left to right, letter by letter, into the word $(\square_1\square_2)^r$ (see the proof of Theorem 3.6.4). In the affirmative, M' accepts.

It follows that $L(M') = (\Sigma'^* \cdot \Sigma) \cup (L(M) \cdot (\Box\Box)^+)$. If $L(M) = \emptyset$, then $L(M') = \Sigma'^* \cdot \Sigma$ is a combinatorial language, but if $L(M)$ is nonempty, then $L(M')$ is obviously not combinatorial. Hence, the problem of deciding whether $L(M)$ is a combinatorial language is PSPACE-complete for stl-det-ORWW-automata. $\Box$

A language $L \subseteq \Sigma^*$ is called *definite* if it can be written as $L = A \cup \Sigma^* \cdot B$ for some finite sets $A, B \subset \Sigma^*$. From the above proof we see immediately that the property of being definite is PSPACE-hard for stl-det-ORWW-automata; however, it remains currently open whether this property can be checked in polynomial space.

Now we turn to the property of circularity. For a language $L \subseteq \Sigma^*$,

$$Circ(L) = \{\, z \mid \exists u, v \in \Sigma^* : z = uv \text{ and } vu \in L \,\}$$

is the set of all circular permutations of the words in L. A language L is called *circular* if $L = Circ(L)$ holds. As by definition, $L \subseteq Circ(L)$, we see that L is circular if, for each word $w \in L$, also each circular permutation of w is in L.

The language $L' = a^+$ considered in the proof of Theorem 3.6.4 is obviously circular. On the other hand, for any non-empty language $L \subseteq \Sigma^+$, where Σ does neither contain a nor the letter $\Box$, the language $L' \cup L \cdot (\Box\Box)^+$ is non-circular, as no element of this language begins with the letter $\Box$. Thus, if M is any stl-det-ORWW-automaton on Σ (that does not accept on empty input), then the stl-det-ORWW-automaton M' constructed from M in the proof of Theorem 3.6.4 has the property that $L(M')$ is circular iff $L(M)$ is empty. This shows that the property of accepting a circular language is PSPACE-hard for stl-det-ORWW-automata. In fact, we have the following result.

Theorem 3.6.7. *The following problem is PSPACE-complete:*

INSTANCE: *A stl-det-ORWW-automaton M.*
QUESTION: *Is the language $L(M)$ circular?*

Proof. It remains to show that this problem is solvable in polynomial space. Let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ be a stl-det-ORWW-automaton accepting a language $L \subseteq \Sigma^+$, and let $n = |\Gamma|$. We must check whether or not $Circ(L) \subseteq L$ holds.

From M we can construct an NFA $A = (Q, \Sigma, \delta_A, q_0, F)$ of size at most $2^{6\cdot(n+1)}$ for L. Now, for each state $q \in Q \smallsetminus \{q_0\}$, we construct an NFA B_q as follows:

- B_q consists of two disjoint copies of A, say $A_1 = (Q^{(1)}, \Sigma, \delta_A^{(1)}, q_0^{(1)}, F^{(1)})$ and $A_2 = (Q^{(2)}, \Sigma, \delta_A^{(2)}, q_0^{(2)}, F^{(2)})$, where $Q^{(1)} = \{\, q^{(1)} \mid q \in Q \,\}$ and $Q^{(2)} = \{\, q^{(2)} \mid q \in Q \,\}$.

- The initial state of B_q is the state $q^{(1)}$ in A_1.

- B_q has only a single final state, which is the state $q^{(2)}$ in A_2.

- For each transition $p' \in \delta_A(p, a)$, where $p \in Q$, $a \in \Sigma$, and $p' \in F$, we add a transition $q_0^{(2)} \in \delta_{B_q}(p^{(1)}, a)$, that is, being in state $p^{(1)}$ of A_1, which corresponds to the state p of A, and reading the letter a, B_q can either stay within A_1 by performing the transition that corresponds to the appropriate transition of A, or it can enter the state $q_0^{(2)}$ of A_2 that corresponds to the initial state of A.

It is now easily seen that w is accepted by B_q if and only if w can be factored as $w = uv$ such that $\delta_A(q, u) \cap F \neq \emptyset$ and $q \in \delta_A(q_0, v)$, that is, B_q accepts a certain subset of the language $\mathrm{C}irc(L)$. In fact, it can be seen that $\mathrm{C}irc(L) = \bigcup_{q \in Q} L(B_q)$ holds. Hence, $\mathrm{C}irc(L) \subseteq L$ holds if and only if $L(B_q) \subseteq L$ for all states q of A.

From M we also construct an NFA C of size at most $2^{6\cdot(n+1)}$ for the language L^c. Then, for each $q \in Q$, we consider the direct product $B_q \times C$ of the automata B_q and C. As $L(B_q \times C) = L(B_q) \cap L^c$, it follows that $L(B_q) \subseteq L$ iff $L(B_q \times C) = \emptyset$. Thus, we need to check emptiness for all product automata $B_q \times C$ ($q \in Q$). Each of these automata is of size $2^{\mathcal{O}(n)}$, and there are $2^{\mathcal{O}(n)}$ many such automata. By systematically going through all possible states $q \in Q$, considering only one NFA $B_q \times C$ at a time, this test can be performed in polynomial space. □

A language $L \subseteq \Sigma^*$ is called *suffix-closed* if each suffix v of each element $w \in L$ also belongs to L, and it is called *suffix-free* if $v \in L$ implies that $uv \notin L$ for all $u \in \Sigma^*$, that is, no proper suffix of any word in L belongs itself to L. The notions of *prefix-closed* language and of *prefix-free* language are defined symmetrically. Using the same proof ideas as above it can be shown that, for a given stl-det-ORWW-automaton M, the problem of deciding whether the language $L(M)$ is suffix-closed (suffix-free, prefix-closed, prefix-free) is also PSPACE-complete.

In this way we have seen that many of the decision problems for stateless deterministic restarting automata are PSPACE-complete.

3.7 Reversible Ordered Restarting Automata

This section deals with another aspect of stateless deterministic ordered restarting automata, namely reversibility. It is based for the most part on the common work "Reversible ordered restarting automata" [38].

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 [3], 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 [39]. The notion of reversibility has also been studied for other types of automata, e.g., for pushdown automata [21] and for queue automata [22].

Here, we introduce a notion of *reversibility* for a rather restricted class of restarting automata, the 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 [24], 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.

Let $M = (\Sigma, \Gamma, \triangleright, \triangleleft, \delta, >)$ 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 $(\triangleright u, abcv\triangleleft)$ to the restarting configuration $\triangleright uab'cv\triangleleft$.

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

$$\delta^R : ((\Gamma \cup \{\triangleright\}) \cdot \Gamma \cdot (\Gamma \cup \{\triangleleft\})) \rightsquigarrow \{\mathsf{MVR}\} \cup \Gamma$$

such that, for all restarting configurations $\triangleright w\triangleleft$ and $\triangleright w'\triangleleft$ that can occur within computations of M, $\triangleright w\triangleleft \vdash_M^c \triangleright w' \triangleleft$ iff $\triangleright w'\triangleleft \vdash_M^{c^R} \triangleright w \triangleleft$. Here $\vdash_M^{c^R}$ denotes a cycle that is realized by using the reverse transition function δ^R. We describe the above reversible stl-det-ORWW-automaton as $M = (\Sigma, \Gamma, \triangleright, \triangleleft, \delta, \delta^R, >)$, 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 $\triangleright w\triangleleft$ is reached from the initial configuration $\triangleright x\triangleleft$ of M. This corresponds to the way reversibility is defined for queue automata in [22].

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 3.7.2. *For each stl-det-ORWW-automaton M working on an alphabet with n letters, there exists a rev-stl-det-ORWW-automaton R with $2^{\mathcal{O}(n)}$ 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 $C : \triangleright uabcv\triangleleft \vdash_M^c \triangleright uab'cv\triangleleft$ 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 3.7.3. *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 3.7.4. *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, $|\Gamma| = n + 1$ (see Lemma 3.3.1). 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 $a, b, c, d \in \Gamma$:

$$
\begin{aligned}
\delta(\rhd a \lhd) &= \delta'(\rhd a \lhd) \text{ for } a \in \Gamma \cup \{\lambda\},\\
\delta(\underline{a} b \lhd) &= \delta'(ab \lhd),\\
\delta(\rhd ab) &= \begin{cases} c, & \text{if } \delta'(\rhd ab) = c,\\ \underline{a}, & \text{if } \delta'(\rhd ab) = \mathsf{MVR}, \end{cases}\\
\delta(\underline{a} bc) &= \begin{cases} d, & \text{if } \delta'(abc) = d,\\ \underline{b}, & \text{if } \delta'(abc) = \mathsf{MVR}, \end{cases}\\
\delta(\rhd \underline{a} b) &= \begin{cases} c, & \text{if } \delta'(\rhd ab) = c,\\ \mathsf{MVR}, & \text{if } \delta'(\rhd ab) = \mathsf{MVR}, \end{cases}\\
\delta(\underline{a}\, \underline{b} c) &= \begin{cases} d, & \text{if } \delta'(abc) = d,\\ \mathsf{MVR}, & \text{if } \delta'(abc) = \mathsf{MVR}, \end{cases}\\
\delta(\rhd \underline{a}\, \underline{b}) &= \mathsf{MVR},\\
\delta(\underline{a}\, \underline{b}\, \underline{c}) &= \mathsf{MVR}.
\end{aligned}
$$

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

$$\rhd x \lhd = \rhd x_1 x_2 x_3 \dots x_{i-1} x_i x_{i+1} \dots x_m \vdash^c_M \rhd x_1 \dots x_{i-1} a x_{i+1} \dots x_m \lhd .$$

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 $\rhd \underline{x}_1 \dots \underline{x}_{i-1} a x_{i+1} \dots x_m \lhd$. 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 $\Delta \smallsetminus \Gamma$ 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 3.7.2.

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 3.7.4 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 [36] 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 neighboring 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 neighboring field contains the l_1-th letter of its sequence W', and the right neighboring 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 $(L, W, R) = (\emptyset, (a), \emptyset)$. We could have used the same encoding as in Section 3.5, but we chose to use this more basic representation.

Further, in order to ensure that triples in neighboring fields are consistent with each other, the following notion has been introduced in [36]. 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 order(R', L) as follows:

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

Now a pair of triples $((L', W', R'), (L, W, R))$ is called *consistent*, if 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. It ensures that a correct ordering of operations can be determined.

- 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 neighboring 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:

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

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:

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

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:

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

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:

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

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. □

We now complete the proof of Theorem 3.7.2 by establishing the following claim.

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

Proof. Let $w, z \in \Delta^m$ such that $(\lambda, \rhd w \lhd) \vdash^c_R (\lambda, \rhd z \lhd)$ 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. $\delta(\rhd U_1 U_2) = \delta(U_1U_2U_3) = \dots = \delta(U_{r-1}U_rA) = \mathsf{MVR}$, 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

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

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 $(\lambda, \rhd \pi(U_1) \ldots \pi(U_{i-1})\pi((L', W', R'))\pi(U_{i+1}) \ldots \pi(V_{m-r-1} \lhd)$, $\hat{M}$ 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 3.7.2. □

Hence, we obtain the following characterization.

Corollary 3.7.5. $\mathsf{REG} = \mathcal{L}(\text{rev-stl-det-ORWW})$.

3.8 Conclusion

In this chapter we have seen that both the deterministic and the nondeterministic stl-ORWW-automata exactly classify the regular languages. We have seen that with the deterministic stl-ORWW-automata we can present some languages in a much more concise way than with a DFA. Conversely, this does not apply: For every DFA there is a det-stl-ORWW-automaton, which represents the same language equally concisely. This also applies to language operations, the mirror operation, e.g. has only a linear blowup. Finally, we have seen that patterns are well suited to describe calculations, which we ultimately used for the NFA construction. At the end, we also looked at how reversibility can be realized for deterministic stateless ordered restarting automata.

Chapter 4

Nondeterministic Ordered Restarting Automata

The nondeterministic ordered restarting automata are the most general ordered restarting automata we have looked at so far because we use states as well as nondeterminism. Unlike the restricted variants the ORWW-automata are more expressive and can describe languages that are not regular. We will see that even some languages are recognized that are not growing context sensitive. Nevertheless, there are some simple languages, like the deterministic linear language $\{a^n b^n \mid n \in \mathbb{N}\}$ that cannot be described by these automata. However this limitation allows us to derive a true pumping lemma for this language class which ultimately enables us to decide emptiness and finiteness.

As we have already given the definition in Section 3.1 we start with an example which shows that we can describe more than just regular languages.

4.1 Examples

In this section we present different languages that can be represented by ORWW-automata. For a simpler and clearer representation we use meta-instructions for RWW-automata in the following examples. A meta-instruction $(E, u \to v)$ is applicable to a restarting configuration $q_0 \rhd w \lhd$, if w can be factored as $w = w_1 u w_2$ such that $\rhd w_1 \in E$, which gives the cycle $q_0 \rhd w \lhd \vdash_M^c q_0 \rhd w_1 v w_2 \lhd$, and a meta-instruction (E, Accept) allows M to accept from any restarting configuration $q_0 \rhd w \lhd$ such that $\rhd w \lhd \in E$ (see Definition 2.3.3).

First of all we start with the linear language $L_{\geq} = \{\, b^m a^n \mid m \geq n \geq 1 \,\}$.

Example 4.1.1. *Let $\Sigma = \{a, b\}$, and let $L_{\geq} = \{\, b^m a^n \mid m \geq n \geq 1 \,\}$. We present an ORWW-automaton M for this linear language.*

Let $M_{\geq}$ be the ORWW-automaton on $\Sigma = \{a, b\}$ and $\Gamma = \{a, a_1, a_2, b, b_1, b_2\}$ that is given by the following meta-instructions using the partial ordering $a > a_1 > a_2$ and $b > b_1 > b_2$:

(1) $(\rhd \cdot ba \cdot \lhd, \mathsf{Accept})$,
(2) $(\lambda, \rhd bb \to \rhd b_1 b)$,
(3) $(\lambda, \rhd b_1 b \to \rhd b_2 b)$,
(4) $(\rhd \cdot b_2^*, b_2 bb \to b_2 b_1 b)$,
(5) $(\rhd \cdot b_2^*, b_2 b_1 b \to b_2 b_2 b)$,
(6) $(\rhd \cdot b_2^*, b_2 ba \to b_2 b_1 a)$,
(7) $(\rhd \cdot b_2^*, b_2 b_1 a_1 \to b_2 b_2 a_1)$,
(8) $(\rhd \cdot b_2^*, b_2 b a_2 \to b_2 b_1 a_2)$,
(9) $(\rhd \cdot b_2^*, b_2 b_1 a_2 \to b_2 b_2 a_2)$,
(10) $(\rhd \cdot b_1 \cdot b^+ \cdot a^*, aa\lhd \to aa_1\lhd)$,
(11) $(\rhd \cdot b_1 \cdot b^*, ba\lhd \to ba_1\lhd)$,
(12) $(\rhd \cdot b_2 \cdot b^+ \cdot a^*, aa_1\lhd \to aa_2\lhd)$,
(13) $(\rhd \cdot b_2 \cdot b^*, ba_1\lhd \to ba_2\lhd)$,
(14) $(\rhd \cdot b_2^+ \cdot b_1 \cdot b^* \cdot a^*, aaa_2 \to aa_1 a_2)$,
(15) $(\rhd \cdot b_2^* \cdot b_1 \cdot b^*, baa_2 \to ba_1 a_2)$,
(16) $(\rhd \cdot b_2^+ \cdot b_2 \cdot b^* \cdot a^*, aa_1 a_2 \to aa_2 a_2)$,
(17) $(\rhd \cdot b_2^* \cdot b_2 \cdot b^*, ba_1 a_2 \to ba_2 a_2)$,
(18) $(\rhd \cdot b_2^* \cdot b_2, b_1 aa_2 \to b_1 a_1 a_2)$,
(19) $(\rhd \cdot b_2^* \cdot b_2, b_2 a_1 a_2 \to b_2 a_2 a_2)$,
(20) $(\rhd \cdot b_2^+ \cdot a_2^+ \cdot \lhd, \mathsf{Accept})$.

Given an input of the form $b^m a^n$, $M_{\geq}$ accepts immediately if $m = 1$ and $n \leq 1$ by (1). Otherwise, we see from the rules of $M_{\geq}$ that $m \geq 2$. By rules (2) to (9), the prefix b^m is rewritten from left to right, where each letter b is first rewritten into b_1 and then into b_2. On the other hand, the suffix a^n is rewritten from right to left by rules (10) to (19), where each letter a is first rewritten into a_1 and then into a_2. However, the first occurrence of a from the right can only be rewritten into a_1, if at that moment the rightmost already rewritten b happens to be b_1, and analogously, a_1 can further be rewritten into a_2 only if at that moment the rightmost already rewritten b happens to be the letter b_2. Thus, it follows that $n \leq m$, that is, $L(M_{\geq}) = L_{\geq}$. □

Next, we look at the language L'_{copy} that is not growing context-sensitive. The construction is based on the same idea as the previous one.

$L'_{\text{copy}} = \{\, w\$u \mid w, u \in \{a, b\}^*, |w|, |u| \geq 2, u$ is a scattered subsequence of $w \,\}$.

Lemma 4.1.2. *The language L'_{copy} is not growing context-sensitive, but there exists an ORWW-automaton M such that $L(M) = L'_{\text{copy}}$.*

Proof. Let M be the ORWW-automaton on $\Sigma = \{a, b, \$\}$ and $\Gamma = \{a, a_1, a_2, b, b_1, b_2, \$\}$ that is given by the following meta-instructions using the ordering $\$ > a > b > a_1 > b_1 > a_2 > b_2$, where $c, d, e \in \{a, b\}$:

(1) $(\lambda, \rhd cd \to \rhd c_1 d)$,
(2) $(\lambda, \rhd c_1 d \to \rhd c_2 d)$,
(3) $(\rhd \cdot \{a_2, b_2\}^*, c_2 de \to c_2 d_1 e)$,
(4) $(\rhd \cdot \{a_2, b_2\}^*, c_2 d_1 e \to c_2 d_2 e)$,
(5) $(\rhd \cdot \{a_2, b_2\}^*, c_2 d\$ \to c_2 d_1 \$)$,
(6) $(\rhd \cdot \{a_2, b_2\}^*, c_2 d_1 \$ \to c_2 d_2 \$)$,
(7) $(\rhd \cdot \{a_2, b_2\}^* \cdot c_1 \cdot \{a, b\}^+, \$cd \to \$c_1 d)$,
(8) $(\rhd \cdot \{a_2, b_2\}^* \cdot c_2 \cdot \{a, b\}^+, \$c_1 d \to \$c_2 d)$,
(9) $(\rhd \cdot \{a_2, b_2\}^* \cdot c_1 \cdot \{a, b\}^+ \cdot \$ \cdot \{a_2, b_2\}^*, d_2 ce \to d_2 c_1 e)$,
(10) $(\rhd \cdot \{a_2, b_2\}^* \cdot c_2 \cdot \{a, b\}^+ \cdot \$ \cdot \{a_2, b_2\}^*, d_2 c_1 e \to d_2 c_2 e)$,
(11) $(\rhd \cdot \{a_2, b_2\}^* \cdot c_1 \cdot \{a, b\}^+ \cdot \$ \cdot \{a_2, b_2\}^*, d_2 c \lhd \to d_2 c_1 \lhd)$,
(12) $(\rhd \cdot \{a_2, b_2\}^* \cdot c_2 \cdot \{a, b\}^+ \cdot \$ \cdot \{a_2, b_2\}^*, d_2 c_1 \lhd \to d_2 c_2 \lhd)$,
(13) $(\rhd \cdot \{a_2, b_2\}^+ \cdot c_1 \cdot \$ \cdot \{a_2, b_2\}^*, d_2 c \lhd \to d_2 c_1 \lhd)$,
(14) $(\rhd \cdot \{a_2, b_2\}^+ \cdot c_2 \cdot \$ \cdot \{a_2, b_2\}^*, d_2 c_1 \lhd \to d_2 c_2 \lhd)$,
(15) $(\rhd \cdot \{a_2, b_2\}^+ \cdot \$ \cdot \{a_2, b_2\}^+ \cdot \lhd, \mathsf{Accept})$.

Given an input of the form $w\$u$, where $w, u \in \{a, b\}^*$, it is easily seen from rules (15), (1), and (7) that $|w|, |u| \geq 2$. By rules (1) to (6), the prefix w is rewritten from left to right, where each symbol is first replaced by its copy with index 1, and then this is replaced by the corresponding letter with index 2. Also the suffix u is rewritten in this way by rules (7) to (14); however, here the first letter from $\{a, b\}$ from the left, say c, can only be rewritten to c_1 if at that moment the rightmost already rewritten letter in w happens to be the letter c_1, and analogously, c_1 can further be rewritten to c_2 only if at that moment the rightmost already rewritten letter in w happens to be the letter c_2. Thus, it follows that u is a scattered subsequence of w, that is, $L(M) = L'_{\text{copy}}$.

In [8] it is shown that each growing context-sensitive language is accepted by a one-way auxiliary pushdown automaton with a logarithmic space bound, that is, the class GCSL of growing context-sensitive languages is contained in $\mathcal{L}(\mathsf{OW\text{-}auxPDA(log)})$, that is, each growing context-sensitive language is accepted by a nondeterministic Turing machine M with a one-way read-only input-tape, a pushdown tape, and an auxiliary work tape limited by endmarkers to length $\log(n)$, when M is started on an input of length n (see Definition 6.1 in [8]). On the other hand, Lautemann has shown in [28] that the language $L_{\text{copy}} = \{\, ww \mid w \in \{a,b\}^* \,\}$ is not accepted by any OW-auxPDA with a logarithmic space bound, and his argument immediately extends to the language $L^{\$}_{\text{copy}} = \{\, w\$w \mid w \in \{a,b\}^* \,\}$.

Now, assume that the language L'_{copy} is growing context-sensitive. Then there is an OW-auxPDA A that accepts this language with a logarithmic space bound. By using an extra track of the auxiliary tape to implement a binary counter that ensures that w and u have the same length, we can extend A into a OW-auxPDA B for the language $L^{\$}_{\text{copy}}$, which contradicts the statement above.

$$\begin{aligned}&\{\, w\$u \mid w,u \in \{a,b\}^*, |w|,|u| \geq 2, u \text{ is a scattered subsequence of } w, |w| = |u| \,\}\\ =&\{\, w\$u \mid w,u \in \{a,b\}^*, w = u \,\} = L^{\$}_{\text{copy}}\end{aligned}$$

Hence, L'_{copy} is not growing context-sensitive. □

As each cycle of a computation of an ORWW-automaton M 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 $>$, we see that each computation of M on an input of length n consists of at most $(|\Gamma|-1) \cdot n$ many cycles. Thus, M can be simulated by a nondeterministic single-tape Turing machine in time $\mathcal{O}(n^2)$. On the other hand, we have the following lower bound.

Theorem 4.1.3. *There exists an ORWW-automaton M such that the language $L(M)$ is NP-complete with respect to log-space reductions.*

Proof. Let $\Sigma_0 = \{\wedge, \vee\}$, $V = \{\, v_n \mid n \in \mathbb{N}_+ \,\}$, $\overline{V} = \{\, \bar{v}_n \mid n \in \mathbb{N}_+ \,\}$, and let 3SAT be the set of satisfiable propositional formulas in conjunctive normal form of degree 3 over $V \cup \overline{V} \cup \Sigma_0$, that is, each $\alpha \in$ 3SAT is of the form

$$\alpha = v_{1,1}^{\mu_{1,1}} \vee v_{1,2}^{\mu_{1,2}} \vee v_{1,3}^{\mu_{1,3}} \wedge v_{2,1}^{\mu_{2,1}} \vee v_{2,2}^{\mu_{2,2}} \vee v_{2,3}^{\mu_{2,3}} \wedge \cdots \wedge v_{m,1}^{\mu_{m,1}} \vee v_{m,2}^{\mu_{m,2}} \vee v_{m,3}^{\mu_{m,3}}$$

for some $m \geq 1$, where $v_{i,j} \in V$ and $\mu_{i,j} \in \{1, -1\}$. Here $v_{i,j}^1 = v_{i,j}$ and $v_{i,j}^{-1} = \bar{v}_{i,j}$. In addition, there exists an assignment $\varphi : V \to \{0,1\}$ such that, for each $j \in \{1, 2, \ldots, m\}$, there is an index $s_j \in \{1,2,3\}$ such that $\varphi(v_{j,s_j}^{\mu_{j,s_j}}) = 1$. It is well-known that the language 3SAT is NP-complete with respect to log-space reductions (see, e.g., [12]). By renaming the variables if necessary, we can assume that α is *normalized*, that is, it contains the variables $v_1, v_2, \ldots, v_k$ for some $k \geq 1$.

For our proof we need a particular encoding of the formulas from 3SAT. Let $\Sigma = \{x, \#, \circ, +, \neg, \wedge, \vee\}$. For $k \geq 1$, we define an encoding $c_k : (V \cup \overline{V} \cup \Sigma_0)^* \to \Sigma^*$ as follows:

$$c_k(y) = \begin{cases} \circ^{i-1} + \circ^{k-i}, & \text{if } y = v_i \in V \text{ and } 1 \leq i \leq k, \\ \circ^{i-1} \neg \circ^{k-i}, & \text{if } y = \bar{v}_i \in \overline{V} \text{ and } 1 \leq i \leq k, \\ \lambda, & \text{if } y = v_i \text{ or } y = \bar{v}_i \text{ for some } i > k, \\ y, & \text{if } y \in \Sigma_0, \end{cases}$$

and we define a function $\Psi : \left(V \cup \overline{V} \cup \Sigma_0\right)^* \to \Sigma^*$ by taking $\Psi(\alpha) := x^k \# c_k(\alpha) \#$, where k is the number of different variables occurring in α. Using Ψ we obtain the language $L_{3\mathrm{SAT}} = \{\, \Psi(\alpha) \mid \alpha \in 3\mathrm{SAT} \,\}$ from 3SAT.

Next we present an ORWW-automaton $M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, \delta, >)$ for a language $L(M)$ that contains $L_{3\mathrm{SAT}}$. In addition, for a normalized propositional formula α in conjunctive normal form of degree 3, it will be the case that α is satisfiable, that is, $\alpha \in 3\mathrm{SAT}$, if and only if $\Psi(\alpha) \in L(M)$. This shows that Ψ is a log-space reduction from 3SAT to $L(M)$, thus proving that $L(M)$ is NP-complete with respect to log-space reductions.

The ORWW-automaton M has tape alphabet $\Gamma = \Sigma \cup \{0_A, 1_A, 0_B, 1_B\}$, and it is described by the meta-instructions (1) to (10) below. Within these meta-instructions, we use the abbreviations

$$D_A = \{0_A, 1_A\}, \quad D_B = \{0_B, 1_B\}, \quad D = D_A \cup D_B,$$

and the functions $f_A : \{\circ, +, \neg\} \times \{0,1\} \to D_A$ and $f_B : \{\circ, +, \neg\} \times \{0,1\} \to D_B$

that are defined as follows:

$$f_A(l,t) = \begin{cases} 0_A, & \text{if } ((l=\circ \text{ or } l=+) \text{ and } t=0) \text{ or } (l=\neg \text{ and } t=1), \\ 1_A, & \text{if } (l=+ \text{ and } t=1) \text{ or } (l=\neg \text{ and } t=0), \end{cases}$$

$$f_B(l,t) = \begin{cases} 0_B, & \text{if } ((l=\circ \text{ or } l=+) \text{ and } t=0) \text{ or } (l=\neg \text{ and } t=1), \\ 1_B, & \text{if } (l=+ \text{ and } t=1) \text{ or } (l=\neg \text{ and } t=0). \end{cases}$$

The ORWW-automaton M will proceed as follows given a formula $\Psi(\alpha) = x^k\#c_k(\alpha)\#$ as input. The occurrences of the letter x are rewritten from left to right into letters from D, alternating between D_A and D_B. By choosing 0_A (or 1_A) for the first occurrence of x, the value 0 (or 1) is chosen for the variable v_1, and analogously for the other variables v_i, $i = 2, 3, \ldots, k$. When the i-th occurrence of x has been replaced, say by t_A (or t_B) for some $t \in \{0,1\}$, then the i-th letter $a_{j,i}$ of each variable encoding $c_k(v_{j,r}^{\mu_{j,r}})$ within $c_k(\alpha)$ is rewritten into the letter that is determined by $f_A(a_{j,i}, t)$ (or $f_B(a_{j,i}, t)$). From the definition of these auxiliary functions we see that $a_{j,i}$ is rewritten into the letter 1_A (or 1_B) only if $j = i$ and either $a_{j,i} = +$ and $t = 1$ or $a_{j,i} = \neg$ and $t = 0$, that is, this happens if and only if $v_{j,r}^{\mu_{j,r}} = v_i$ and $t = 1$ or if $v_{j,r}^{\mu_{j,r}} = \bar{v}_i$ and $t = 0$. Thus, when all rewrites have been completed, then the rewritten form of each clause of α contains an occurrence of the letter 1_A or 1_B if and only if the assignment chosen through the rewrites of the occurrences of the letter x is a satisfying assignment for α.

The meta-instructions (1) to (10) below are defined in such a way that it is ensured that each time a letter x is rewritten, all of the corresponding letters in $c_k(\alpha)$ are rewritten correctly, and that M satisfies the above acceptance condition. In these meta-instructions, $s, t \in \{0,1\}$, $s_A, t_A \in D_A$, $s_B, t_B \in D_B$, $z \in \{x, \#\}$, $l \in \{\circ, +, \neg\}$, $z' \in \Sigma$, and $z_1 \in \{\wedge, \vee\}$:

$$\begin{array}{ll}
(1) & (\lambda, \triangleright xz \to \triangleright t_A z), \\
(2) & (\triangleright(D_A D_B)^*, t_A xz \to t_A s_B z), \\
(3) & (\triangleright(D_A D_B)^* D_A, t_B xz \to t_B s_A z), \\
(4) & (\triangleright t_A x^*, \#lz' \to \#f_A(l,t)z'), \\
(5) & (\triangleright t_A x^* \# \Gamma^*, z_1 l z' \to z_1 f_A(l,t) z'),
\end{array}$$

(6) $(\rhd (D_A D_B)^+ t_A x^* \# (D_A D_B)^* D_A, s_B l z' \to s_B f_A(l,t) z')$,

(7) $(\rhd (D_A D_B)^+ t_A x^* \# \Gamma^* \Sigma_0 (D_A D_B)^* D_A, s_B l z' \to s_B f_A(l,t) z')$,

(8) $(\rhd (D_A D_B)^* D_A t_B x^* \# (D_A D_B)^*, s_A l z' \to s_A f_B(l,t) z')$,

(9) $(\rhd (D_A D_B)^* D_A t_B x^* \# \Gamma^* \Sigma_0 (D_A D_B)^*, s_A l z' \to s_A f_B(l,t) z')$,

(10) $(\rhd D^+ \# ((\vee | D)^* (1_A | 1_B) (\vee | D)^* (\# | \wedge))^+ \lhd, \mathsf{Accept})$.

Let α be a propositional formula in conjunctive normal form of degree 3 that contains the variables $v_1, v_2, \ldots, v_k$. Given $w = \Psi(\alpha) = x^k \# c_k(\alpha) \#$ as input, M proceeds as follows. From instruction (10) we see that all occurrences of the letters $x, \circ, +$, and $\neg$ must be replaced by symbols from D. Using instructions (1) to (3), the prefix x^k can be rewritten into a word from D^k, where the i-th occurrence of x is replaced by 0_A or 1_A for i odd, and it is replaced by 0_B or 1_B for i even. In this way an assignment of the variables $v_1, v_2, \ldots, v_k$ is chosen nondeterministically. Instructions (4) to (9) show that each syllable of the form $\circ^{j-1} s \circ^{k-j}$, $s \in \{+, \neg\}$, can also be rewritten into a word from D^k, where each symbol is replaced by either 0_A (or 0_B) with the exception that the symbol $+$ is replaced by 1_A (or 1_B), if the last occurrence of the symbol x that was rewritten prior to the current rewrite operation was replaced by 1_A (or 1_B), and the symbol $\neg$ is replaced by 1_A (or 1_B), if the last occurrence of the symbol x that was rewritten prior to the current rewrite operation was replaced by 0_A (or 0_B). As all these syllables have length k, we see that, each time an occurrence of x is rewritten, a symbol must be rewritten in each of the syllables of the form $\circ^{j-1} s \circ^{k-j}$. Finally, instruction (10) shows that in order for M to accept, there must be at least one occurrence of the symbol 1_A or 1_B in each factor that is limited by $\#$ and $\wedge$ symbols, that is, in each factor that is the encoding of a clause of α. However, this means that M can accept on input $w = \Psi(\alpha)$ if and only if there is an assignment for the variables occurring in α such that α yields the value 1, that is, if and only if the formula α is satisfiable. □

As we have now seen which languages we can express, we will now look at some closure properties.

4.2 Closure Properties

In this section we derive some closure properties for ORWW-automata.

We start with a technical lemma.

Lemma 4.2.1. *For each ORWW-automaton M, there exists an ORWW-automaton M' such that $L(M') = L(M)$, but M' only halts (accepting or non-accepting) with the left sentinel $\rhd$ in its window.*

Proof. Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORWW-automaton. Each computation of M consists of a finite sequence of cycles that is followed by a tail computation, which either ends with an accept instruction or by reaching a configuration in which M gets stuck. We now modify M into an ORWW-automaton $M' = (Q, \Sigma, \Gamma', \rhd, \lhd, q_0, \delta', >')$ by taking $\Gamma' = \Gamma \cup \{@, ®\}$, by extending the partial ordering $>$ to $>'$ by taking $x >' @$ and $x >' ®$ for all $x \in \Gamma$, and by defining the transition relation δ' based on δ with the following modifications:

(1) We replace each accept instruction $\delta(q, abc) = \mathsf{Accept}$ by the rewrite/restart instruction $\delta'(q, abc) = \{@\}$.

(2) For each $q \in Q$, $a \in \Gamma \cup \{\rhd\}, b \in \Gamma$, and $c \in \Gamma \cup \{\lhd\}$, if $\delta(q, abc) = \emptyset$, then we take $\delta'(q, abc) = \{®\}$.

(3) We introduce the transitions $\delta'(q, ab@) = \{@\}$ and $\delta'(q, ab®) = \{®\}$ for all $q \in Q$, all $a \in \Gamma \cup \{\rhd\}$, and all $b \in \Gamma$.

(4) Finally, we take $\delta'(q_0, \rhd @ b) = \mathsf{Accept}$ for all $b \in \Gamma \cup \{@, \lhd\}$.

Given an input $w \in \Sigma^+$, each computation of M ends by either an accept step or by reaching a configuration in which M has no applicable transition. In the former case, M' will produce an occurrence of the symbol @ by (1), in the latter, it will produce an occurrence of the symbol ® by (2). Observe that in the next cycle, M' can move its window all the way to the right until it detects the symbol @ or ®, and then it can rewrite all the letters to the left of this newly written symbol into the same symbol, proceeding from right to left by (3). Finally, if and when the first symbol to the right of the left sentinel has been rewritten into the symbol @, then M' halts and accepts by (4). Hence, it is immediate that $L(M) \subseteq L(M')$ and that M' only halts with the left sentinel in its window.

Notice that even after producing a letter @ or Ⓡ on the tape, M' may still apply a rewrite operation of M within the prefix of the tape content that ends with the letter @ or Ⓡ. This, however, means that also M could have executed this rewrite step instead of performing the tail computation that was simulated by M'. Hence, if this additional rewrite step leads to an accepting computation, then this corresponds to an accepting computation of M. Thus, it follows that $L(M') = L(M)$. □

Whenever the marker @ or Ⓡ appears as the first symbol to the right of the left sentinel $\rhd$, then, proceeding from left to right, M could copy this symbol to all other tape cells and then halt at the right sentinel $\lhd$. Thus, instead of requiring that M always halts at the left sentinel, we can also require that M always halts at the right sentinel, if that fits our purpose.

Now we first show that $\mathcal{L}(\mathsf{ORWW})$ is an AFL, that is, an abstract family of languages [16]. In fact, we have a slightly stronger result.

Theorem 4.2.2. $\mathcal{L}(\mathsf{ORWW})$ *is closed under union, intersection, product, Kleene star, inverse morphisms, and non-erasing morphisms.*

Proof. In Theorem 3.3.4 and its Corollary it is shown that $\mathcal{L}(\mathsf{stl\text{-}det\text{-}ORWW})$ is closed under union and intersection. Here we can use the same proof idea. Let $M_i = (Q_i, \Sigma, \Gamma_i, \rhd, \lhd, q_0^{(i)}, \delta_i, >_i)$, $i = 1, 2$, be an ORWW-automaton such that $L(M_i) = L_i$. By Lemma 4.2.1 we can assume that M_i only halts (accepting or non-accepting) with the left sentinel $\rhd$ in its window. From M_1 and M_2 we now obtain an ORWW-automaton M for $L = L_1 \cap L_2$ on the alphabet $\Delta = \Sigma \cup \{\, [a, b] \mid a \in \Gamma_1, b \in \Gamma_2 \,\}$ that works as follows:

(1) From right to left, M rewrites each input letter a into the pair $[a, a]$.

(2) Then M simulates M_1 using the first component of each letter.

(3) When M_1 accepts, which happens at the left sentinel $\rhd$, then M simulates M_2 using the second component of each letter. If and when M_2 accepts, then so does M.

In order to obtain an ORWW-automaton M' such that $L(M') = L_1 \cup L_2$, we modify the above construction of M in that, in step (3), M' halts and accepts, if M_1 accepts. On the other hand, if the computation of M_1 that is simulated by M' is non-accepting, then it still ends at the left sentinel, and then M' can continue in the same way as M.

Closure under product: Here we present an ORWW-automaton $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ for the language $L_1 \cdot L_2$, which works as follows:

1. Given a word $w \in \Sigma^*$ as input, M rewrites w from right to left, letter by letter, such that each letter of a suffix v of w is marked by an index 2, and then each letter of the corresponding prefix u is marked by an index 1. In this way $w \in \Sigma^*$ is (nondeterministically) split into $w = uv$ with the idea that $u \in L_1 = L(M_1)$ and $v \in L_2 = L(M_2)$ are to be checked.

2. After the first letter of w has been marked by an index 1, M simulates the automaton M_1 on the prefix u. During this process, the leftmost occurrence of a letter with index 2 is interpreted as the right delimiter $\lhd$.

3. When the simulated computation of M_1 on u accepts, then M realizes this with the left sentinel $\rhd$ in its window. It then moves right until it detects the first letter with index 2 or the right sentinel $\lhd$. In the latter case, it accepts if $\lambda \in L_2$, while in the former case it rewrites all the letters from the prefix u, from right to left, by the special symbol $\Box$.

4. When the first letter of w has been rewritten by the letter $\Box$, then M simulates M_2. During this process it simply ignores the prefix of $\Box$-symbols on the tape, simulating M_2 on the suffix v. Now M accepts if this computation of M_2 accepts.

5. If in step 1, all letters are marked with an index 2, that is, $v = w$ and $u = \lambda$ are chosen, then M simply simulates M_2 on v, provided $\lambda \in L_1$; otherwise, it simply halts without accepting.

Closure under Kleene star: Here the idea is essentially the same as for the operation of product. Given a word $w \in \Sigma^*$ as input, M rewrites the word from right to left, letter by letter, attaching indices 1 or 2 to these letters. In this way a factorization $w = u_1 u_2 \cdots u_m$ is chosen nondeterministically, and it remains to check that $u_1, u_2, \ldots, u_m \in L(M_1)$. This can be done as above, using two copies of the automaton M_1, where the one works on an alphabet in which all letters have index 1, and the other one works on an alphabet in which all letters have index 2.

Closure under inverse morphisms: Let $M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, \delta, >)$ be an ORWW-automaton, and let $f : \Sigma'^* \to \Sigma^*$ be a morphism. We present an ORWW-automaton $M' = (Q', \Sigma', \Gamma', \triangleright, \triangleleft, q_0', \delta', >')$ such that $L(M') = f^{-1}(L(M)) = \{\, w \in \Sigma'^* \mid f(w) \in L(M) \,\}$. Here we assume that the automaton M always halts at the right sentinel (see the remark following Lemma 4.2.1).

Basically the automaton M' proceeds as follows. Given an input $w = a_1 a_2 \cdots a_n \in \Sigma'^*$, it rewrites each letter $a_i \in \Sigma'$, proceeding from right to left, by its image $f(a_i) \in \Sigma^*$, and then it simulates a computation of M on input $f(w)$. However, there are two problems that we need to overcome. First, the length of a word $f(a_i)$ may be larger than one. Accordingly, Γ' will contain block symbols of the form $[u]$ that represent a word $u \in \Sigma^+$ of length up to $\mu = \max\{\, |f(a)| \mid a \in \Sigma' \,\}$. Secondly, it may happen that $f(a) = \lambda$ for some letters $a \in \Sigma$. In this situation, a will be rewritten into a symbol of the form $[u]^c \in \Gamma'$ that represents a copy of its right-hand neighbor $[u]$. Of course, there can be several of these copy symbols in a row. In the course of a computation they will always be updated from right to left. As these copy symbols may separate the block symbols on the tape from each other, M' must carry information on the last block symbol it has seen when moving to the right. Accordingly, we define the set of states Q' and the tape alphabet Γ' through

$$\begin{aligned} Q' &= \{q_0'\} \cup \{\, [q, x] \mid q \in Q, x \in \Gamma \cup \{\triangleright\} \,\}, \text{ and} \\ \Gamma' &= \Sigma' \cup \{\, [u], [u]^c, [u]_x, [u]_x^c \mid x \in \Gamma \cup \{\triangleright\},\ u \in \Gamma^*, 1 \leq |u| \leq \mu, \text{ or } u = \triangleleft \,\}, \end{aligned}$$

and we define the partial ordering $>'$ from the partial ordering $>$ through

$$a >' [w]^\varepsilon >' [w]_x^\varepsilon \text{ and } [v_1 \cdots v_i \cdots v_n]_x^\varepsilon >' [v_1 \cdots \hat{v}_i \cdots v_n]_y^\varepsilon \text{ if } v_i > \hat{v}_i,$$

where $a \in \Sigma'$, $\varepsilon \in \{c, \lambda\}$, $w \in \Sigma^+$, and $v_1, \ldots, v_i, \ldots, v_n, \hat{v}_i, x, y \in \Gamma \cup \{\triangleright\}$.

The transition function δ' is defined in such a way that M' proceeds as follows:

1. First we consider all words of length at most one, that is, $a \in \Sigma' \cup \{\lambda\}$,

defining $\delta'(q_0', \rhd a \lhd) = \mathsf{Accept}$, if $f(a) \in L(M)$.

2. For $w \in \Sigma'^*$, $|w| \geq 2$, we first replace each letter b by the block letter $[f(b)]$, proceeding from right to left. During this process each image $f(b) = \lambda$ is replaced by a marked copy $[u]^c$ of its right-hand neighbor $[u]^\varepsilon$, where $[u]^\varepsilon$ is used to represent $[u]$ or $[u]^c$, and analogously for $[u]_x^\varepsilon$.

3. Then, again proceeding from right to left, each symbol $[u]^\varepsilon$ is replaced by the indexed symbol $[u]_x^\varepsilon$, where $x \in \Gamma \cup \{\rhd\}$ is the next symbol of $\rhd f(w) \lhd$ to the left of the current block u.

4. Next M' simulates M, where in each rewrite step, it puts the information on the last letter x of the first proper block symbol $[ux]$ to the left of the current position, which is stored in its state, into the index of the newly written block symbol. Thus, a block symbol $[v_1 \cdots v_{i-1} v_i v_{i+1} \cdots v_k]_z$ is rewritten into the symbol $[v_1 \cdots v_{i-1} \hat{v}_i v_{i+1} \cdots v_k]_x$, if in the current situation, M would rewrite v_i into $\hat{v}_i$. Here x is the next symbol to the left of v_1 in the current word on the tape when ignoring copy symbols.

5. Whenever M' encounters a copy symbol, it simply skips it unless this symbol must be updated. In this process M' also checks that no outdated copy has been used in a rewrite step by comparing the neighboring letter stored in the current state with the letter stored in the index of the current block. Thus, if it encounters a factor $[u]_y^c [v]_z^c [w]_r^\varepsilon$ such that $v \neq w$, then $[v]_z^c$ is replaced by $[w]_r^c$, and if it encounters a factor $[u_1 \cdots u_{k-1} x]_y [v]_z^c [w]_x^\varepsilon$ such that $v \neq w$, then $[v]_z^c$ is replaced by $[w]_x^c$. Observe that in this situation it is required that the index x of the block symbol $[w]_x^\varepsilon$ coincides with the last letter in the previous proper block symbol $[u_1 \cdots u_{k-1} x]_y$. In this way it is ensured that the last symbol x within the block symbol $[u_1 \cdots u_{k-1} x]_y$ has not been rewritten since it was used as the left neighbor for a rewrite within the next proper block symbol $[w]_x$.

6. Finally, M' accepts with the right sentinel $\lhd$ in its window, if M does, and if the various block symbols currently on the tape are consistent, which M' can check while scanning its tape from left to right.

It can now be shown that M' accepts a word $w' \in \Sigma'^*$ if and only if M accepts the word $f(w')$, that is, $L(M') = f^{-1}(L(M))$.

A more detailed descriptions follows below.

The automaton M' basically replaces the letters by their images and simulates the automaton M on the blocks of letters. Empty words are replaced by a copy of their right neighbor which are updated from right to left.

The set of states Q' consists of an initial state q_0 and pairs $[q, x]$ where $q \in Q$ and $x \in \Gamma$. x contains the letter that would be to the left of the left-most letter in the current block. q represents the state in which the automaton M would be at the left-most letter in the current block.

The working alphabet Γ' consists of the input alphabet Σ' and blocks of letters that form words that are images of single letters in Γ'. These blocks can be supplemented with an indicator for being a copy and the neighboring letter that would be to the left of the left-most letter in the current block.

We always add the neighboring letter from the state whenever we rewrite a block. By doing this we verify that the automaton doesn't use outdated copies of blocks when the block has already been rewritten as we only allow an update of the copies if the neighboring letters in the state and block coincide.

The transition function Δ is defined as follows, where a block with a star $[x_1 \dots x_k]^*$ represents $[x_1 \dots x_k]$ or $[x_1 \dots x_k]^c$. All these blocks can implicitly be complemented with a neighboring letter.

At first we treat all words w with $|w| \leq 1$

$$\begin{aligned} \Delta(q_0, \rhd\lhd) &= \mathsf{Accept}, \quad \text{if } \lambda \in f(L) \\ \Delta(q_0, \rhd b\lhd) &= \mathsf{Accept}, \quad \text{if } f(b) \in f(L) \end{aligned}$$

For all other cases we replace all letters by their images from right to left. Empty images are replaced by marked copies of their right neighbor:

$$\begin{aligned} \Delta(q_0, \rhd bc) &= \{(q_0, \mathsf{MVR})\} \\ \Delta(q_0, abc) &= \{(q_0, \mathsf{MVR})\} \\ \Delta(q_0, ab\lhd) &= \{[f(b)]\}, \quad \text{if } f(b) \neq \lambda \\ \Delta(q_0, ab\lhd) &= \{[\lhd]^c\}, \quad \text{if } f(b) = \lambda \\ \Delta(q_0, ab[w]^*) &= \{[f(b)]\}, \quad \text{if } f(b) \neq \lambda \\ \Delta(q_0, ab[w]^*) &= \{[w]^c\}, \quad \text{if } f(b) = \lambda \\ \Delta(q_0, \rhd b[w]^*) &= \{[f(b)]\}, \quad \text{if } f(b) \neq \lambda \\ \Delta(q_0, \rhd b[w]^*) &= \{[w]^c\}, \quad \text{if } f(b) = \lambda \end{aligned}$$

Now the automaton M is simulated on the blocks of letters and in every rewrite we write down the neighboring letter which is stored in the state at that moment.

$$\begin{aligned}
&\Delta(q_0, \rhd[v_1 \dots v_k][v_{k+1} \dots v_{k+l}]^*) \ni [v_1 \dots \hat{v}_i \dots v_k]_{\rhd},\\
&\quad \text{if } \exists z_1, \dots, z_{i-1} \in Q : \delta(z_{i-1}, v_{i-1} v_i v_{i+1}) \ni \hat{v}_i,\\
&\quad \text{and } \forall 0 \le j \le i-2 : \delta(z_j, v_j v_{j+1} v_{j+2}) \ni (z_{j+1}, \mathsf{MVR})\ , z_0 := q_0, v_0 := \rhd\\
&\Delta(q_0, \rhd[v_1 \dots v_k][v_{k+1} \dots v_{k+l}]^*) \ni ([z_k, v_k], \mathsf{MVR}),\\
&\quad \text{if } \exists z_1, \dots, z_k \in Q : \forall 0 \le j \le k-1 :\\
&\quad \delta(z_j, v_j v_{j+1} v_{j+2}) \ni (z_{j+1}, \mathsf{MVR})\ , z_0 := q_0, v_0 := \rhd\\
&\Delta([q,x], [u]^*[v_1 \dots v_k][v_{k+1} \dots v_{k+l}]^*) \ni [v_1 \dots \hat{v}_i \dots v_k]^c_x,\\
&\quad \text{if } \exists z_1, \dots, z_{i-1} \in Q : \delta(z_{i-1}, v_{i-1} v_i v_{i+1}) \ni \hat{v}_i,\\
&\quad \text{and } \forall 0 \le j \le i-2 : \delta(z_j, v_j v_{j+1} v_{j+2}) \ni (z_{j+1}, \mathsf{MVR})\ , z_0 := q, v_0 := x\\
&\Delta([q,x], [u]^*[v_1 \dots v_k][v_{k+1} \dots v_{k+l}]^*) \ni ([z_k, v_k], \mathsf{MVR}),\\
&\quad \text{if } \exists z_1, \dots, z_k \in Q : \forall 0 \le j \le k-1 :\\
&\quad \delta(z_j, v_j v_{j+1} v_{j+2}) \ni (z_{j+1}, \mathsf{MVR})\ , z_0 := q, v_0 := x
\end{aligned}$$

Whenever we encounter copies we skip them unless they need to be updated. In the same process we verify that no outdated copy has been used by comparing the neighboring letters from state and block.

$$\begin{aligned}
&\Delta(q_0, \rhd[v]^c[v]^*) \ni ([q_0, \rhd], \mathsf{MVR})\\
&\Delta([q,x], [u]^*[v]^c[v]^*) \ni ([q,x], \mathsf{MVR})\\
&\Delta(q_0, \rhd[v]^c[w]^*_{\rhd}) \ni [w]^c_{\rhd}\\
&\Delta([q,x], [u]^*[v]^c[w]^*_x) \ni [w]^c_x
\end{aligned}$$

Finally we handle the right sentinel case.

$$\begin{aligned}
&\Delta([q,x], [u]^*[v_1 \dots v_k]\lhd) \ni [v_1 \dots \hat{v}_i \dots v_k]_x,\\
&\quad \text{if } \exists z_1, \dots, z_{i-1} \in Q : \delta(z_{i-1}, v_{i-1} v_i v_{i+1}) \ni \hat{v}_i\\
&\quad \text{and } \forall 0 \le j \le i-2 : \delta(z_j, v_j v_{j+1} v_{j+2}) \ni (z_{j+1}, \mathsf{MVR})\ , z_0 := q, v_0 := x, v_{k+1} := \lhd\\
&\Delta([q,x], [u]^*[v_1 \dots v_k]\lhd) \ni \mathsf{Accept},\\
&\quad \text{if } \exists z_1, \dots, z_{i-1} \in Q : \delta(z_{k-1}, v_k v_i \lhd) \ni \mathsf{Accept}\\
&\quad \text{and } \forall 0 \le j \le k-2 : \delta(z_j, v_j v_{j+1} v_{j+2}) \ni (z_{j+1}, \mathsf{MVR}), z_0 := q, v_0 := x
\end{aligned}$$

The partial ordering $>'$ is simply deduced from the partial ordering $>$: $a > [w]^*$ and $[w_1 \dots w_k \dots w_n]^* > [w_1 \dots w_k{}' \dots w_n]^*$ if $w_k > w_k{}'$.

The alphabet we use is finite because Σ and therefore $f(\Sigma)$ is finite.

Now we have to verify that $L(M') = f^{-1}(L(M))$ holds.

Let $w' \in f^{-1}(L(M))$. Then there exists a word $w \in \Sigma^*$ such that $f(w') = w \in L(M)$. It follows that there exists a computation $\rhd w \lhd \vdash^*_M \mathsf{Accept}$.

We can now conclude that $\rhd w' \lhd \ \vdash^*_{M'} \ \rhd f'(w') \lhd \ \vdash^*_{M'} \mathsf{Accept}$ where $f(w'_1 \dots w'_n) = [f(w'_1)] \dots [f(w'_n)]$ if no images are empty. Copies are used if some images are empty. The computation can be continued in the following way: If there are no empty images the computation can be deduced straight away. For every letter that is rewritten in M we rewrite the corresponding letter in the blocks and can finally accept at the right sentinel. If we use copies we can do the same, but if we rewrite a block that possesses at least one copy we add some rewrites directly after rewriting the block to update the copies from right to left.

Now let us verify the other direction. Let $w \in L(M')$. Then there exists a computation $\rhd w \lhd \vdash_{M'}^{*} \rhd f(w) \lhd \vdash_{M'}^{*} \mathsf{Accept}$. We claim that $f(w) \in L(M)$, i.e. there is an accepting computation $\rhd f(w) \lhd \vdash_{M}^{*} \mathsf{Accept}$. If no image is empty we can use the same argumentation as for the other direction: We just do the corresponding rewrites. If some images of the morphism f are the empty word we do all corresponding rewrites except the ones for the copies. We can just skip the copies because our constructed automaton makes sure that copies are only used for rewrites if they match their original letter.

Closure under non-erasing morphisms: Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORWW-automaton, and let $f : \Sigma^* \to \Omega^*$ be a non-erasing morphism. We describe an ORWW-automaton $M' = (Q', \Omega, \Gamma', \rhd, \lhd, q_0', \Delta, >')$ for $f(L(M))$.

Given a word $w \in \Omega^*$ as input, M' first guesses a factorization $u_1 u_2 \cdots u_m$ of w such that, for each $i = 1, 2, \ldots, m$, $|u_i| \leq \mu = \max\{\, |f(a)| \mid a \in \Sigma \,\}$. This is done by marking the letters of w, one by one, from right to left, by indices 1 and 2 (see the proof for the closure under Kleene star above). Each factor u_i is a candidate for an image of a letter under the morphism f. Then, processing the factors u_i from right to left, M' checks whether $u_i = f(a)$ for some letter $a \in \Sigma$. In the negative, it halts immediately without accepting, while in the affirmative it nondeterministically chooses a letter $a_i \in \Sigma$ satisfying $f(a_i) = u_i$ and rewrites u_i into the word $[a_i]^c \cdots [a_i]^c [a_i]$, that is, the last letter of u_i is rewritten into a block symbol that encodes the letter $a_i \in \Sigma$, and all the other letters of u_i (if any) are rewritten into corresponding copy symbols. Thereafter, M' simulates a computation of M on the input $a_1 a_2 \cdots a_m$ using the technique from the proof of closure under inverse morphisms above. It follows that M' accepts on input $w \in \Omega^*$ if and only if there exists a word $u \in \Sigma^*$ such that $f(u) = w$ and $u \in L(M)$, that is, if and only if $w \in f(L(M))$.

A more detailed description including the transition function follows.

As the morphism is non-erasing, the automaton guesses the blocks of the initial letter's images. This is done by marking the letters alternatingly with

1 and 2 like we did with the Kleene Star construction. In the next phase it goes to the right border and memorizes the actual block of letters in the state. The automaton writes one of the words in the inverse image of the current position into the field unless we choose the empty word. In that case it creates a marked copy of the right content. After that the automaton M is simulated on the blocks of letters in the same way as we did for the inverse morphisms.

Let l_{max} be the constant that contains the maximal length of the words in the morphism's image of single letters:

$$l_{max} := \max_{w \in \Sigma} |f(w)|.$$

The set of states Q' consists of an initial state q_0, possible prefixes of words from the morphism's image and pairs $[q, x]$ where $q \in Q$ and $x \in \Gamma$. x contains the letter that would be to the left of the current letter in the block. q represents the state in which the automaton M would be at the current letter.

$$Q' = \{q_0\} \cup \left\{w \in \Sigma^+ \mid |w| \leq l_{max}\right\} \cup \{[q, x] \,|q \in Q, x \in \Gamma\}$$

The working alphabet Γ' consists of the input alphabet Σ and single letters in a block.

These blocks can be supplemented with an indicator for being a copy and the neighboring letter that would be to the left of the letter in the current block:

$$\Gamma' = \Sigma \cup \{[w], [w]^c, [w]_x, [w]^c_x \mid w \in \Gamma, x \in \Gamma \cup \{\triangleright\}\}.$$

The transition function Δ can be defined as follows. First, we treat the special case $|w| = 0, 1$

$$\begin{aligned} \Delta(q^*, \triangleright\triangleleft) &= \mathsf{Accept}, \quad \lambda \in f(L) \\ \Delta(q^*, \triangleright a\triangleleft) &= \mathsf{Accept}, \quad a \in f(L). \end{aligned}$$

For all longer words we mark the different areas:

$$\begin{aligned} \Delta(q_0, \triangleright ab) &= \{(q_0, \mathsf{MVR})\} \\ \Delta(q_0, abc) &= \{(q_0, \mathsf{MVR})\} \\ \Delta(q_0, ab\triangleright) &= \{b_1, b_2\} \\ \Delta(q_0, abc_i) &= \{b_1, b_2\} \\ \Delta(q_0, \triangleright bc_i) &= \{b_1, b_2\} \end{aligned}$$

Now we replace the right-most content of every separate area by a letter that could have that image and fill the rest with the corresponding copies. $a_?$ represents a_1 or a_2. s is a helper function that maps $1 \mapsto 2$ and $2 \mapsto 1$.

$$\begin{aligned}
\Delta(q_0, \rhd b_i c_i) &= \{((bc, i), \mathsf{MVR})\}, \quad |bc| \leq l_{max} \\
\Delta(q_0, \rhd b_i c_{s(i)}) &= \{(c, \mathsf{MVR})\} \\
\Delta(u, a_? b_i c_{s(i)}) &= \{(c, \mathsf{MVR})\} \\
\Delta(u, a_? b_i c_i) &= \{(uc, \mathsf{MVR})\}, \quad |uc| \leq l_{max} \\
\Delta(u, a_? b_i \lhd) &= \{[w]_i \mid w \in f^{-1}(u) \cap \Sigma\} \\
\Delta(u, a_? b_i [v]_i) &= \{[v]_i^c\} \\
\Delta(u, a_? b_i [v]_{s(i)}) &= \{[w]_i \mid w \in f^{-1}(u) \cap \Sigma\} \\
\Delta(q_0, \rhd b_i [v]_i) &= \{[v]_i^c\} \\
\Delta(q_0, \rhd b_i [v]_{s(i)}) &= \{[w]_i \mid w \in f^{-1}(b) \cap \Sigma\}
\end{aligned}$$

Finally we are in the same situation as we were when we were handling inverse morphisms. We just do not have to treat blocks of letters. The alternating numbering isn't needed anymore and is ignored from now on.

$$\begin{aligned}
&\Delta(q_0, \rhd [v_1][v_2]^*) \ni [\hat{v}_1]_{\rhd}, \quad \text{if } \delta(q_0, \rhd v_1 v_2) \ni \hat{v}_1 \\
&\Delta(q_0, \rhd [v_1][v_2]^*) \ni ([z_1, v_1], \mathsf{MVR}), \quad \text{if } \delta(q_0, \rhd v_1 v_2) \ni (z_1, \mathsf{MVR})
\end{aligned}$$

If we encounter copies we skip them if they don't need to be updated.

$$\begin{aligned}
\Delta(q_0, \rhd [v]^c [v]^*) &\ni ([q_0, \rhd], \mathsf{MVR}) \\
\Delta([q, x], [u]^* [v]^c [v]^*) &\ni ([q, x], \mathsf{MVR})
\end{aligned}$$

$$\begin{aligned}
&\Delta([q, x], [u]^* [v_1][v_2]^*) \ni [\hat{v}_1]_x, \quad \text{if } \delta(q, x v_1 v_2) \ni \hat{v}_1 \\
&\Delta([q, x], [u]^* [v_1][v_2]^*) \ni ([z_1, x], \mathsf{MVR}), \quad \text{if } \exists z_1 \in Q : \delta(q, x v_1 v_2) \ni (z_1, \mathsf{MVR})
\end{aligned}$$

We update the copies when we need to and verify that we have not used a copied letter when the original letter has already been rewritten.

$$\begin{aligned}
\Delta(q_0, \rhd [v]^c [w]^*_{\rhd}) &\ni [w]^c_{\rhd} \\
\Delta([q, x], [u]^* [v]^c [w]^*_x) &\ni [w]^c_x
\end{aligned}$$

Finally we treat the right sentinel.

$$\begin{aligned}
&\Delta([q, x], [u]^* [v] \lhd) \ni [\hat{v}]_x, \quad \text{if } \delta(q, x v \lhd) \ni \hat{v} \\
&\Delta([q, x], [u]^* [v] \lhd) \ni \mathsf{Accept}, \quad \text{if } \delta(q, x v \lhd) \ni \mathsf{Accept}
\end{aligned}$$

The ordering $>'$ is defined as follows

$$\forall a \in \Sigma' \forall u, v, w \in \Gamma : a >' [w]^* \text{ and } [u]^* >' [v]^* \text{ iff } u > v$$

Now it follows with the same argumentation that $L(M') = f(L)$ holds. For every valid computation $\triangleright w \triangleleft \vdash_M^* \mathsf{Accept}$ there is a computation $\triangleright f(w) \triangleleft \vdash_{M'}^* \triangleright [w] \triangleleft \vdash_{M'}^* \mathsf{Accept}$, and for every computation $\triangleright w' \triangleleft \vdash_{M'}^* \triangleright [\tilde{w}] \triangleleft \vdash_{M'}^* \mathsf{Accept}$, there is a computation $\triangleright \tilde{w} \triangleleft \vdash_M^* \mathsf{Accept}$ with $f(\tilde{w}) = w'$. That concludes the proof.

□

It remains open whether $\mathcal{L}(\mathsf{ORWW})$ is a *full* AFL, that is, whether it is closed under arbitrary morphisms.

In order to deduce a few non-closure properties, we present a pumping lemma for ordered restarting automata.

4.3 A Pumping Lemma

In this section we derive a pumping lemma for ordered restarting automata. It is actually separated into a Cut-and-Paste-Lemma and a Pumping-Lemma. For a given ORWW-automaton M a part from the end of a sufficiently long accepted word can be omitted such that the resulting word is recognized by M. This fact is covered by the Cut-and-Paste Lemma. The Pumping-Lemma covers the fact that a part from the beginning of a sufficient long word can be repeated arbitrarily many times.

Although we can derive the Cut-and-Paste Lemma with the tools we use to derive the Pumping-Lemma we prove it on its own with a slightly simpler technique in order to get more familiar with manipulating computation cycles.

Theorem 4.3.1. (Cut-and-Paste Lemma)
For each ORWW-automaton M, there exists a constant $N(M) > 0$ such that each word $w \in L(M)$, $|w| \geq N(M)$, has a factorization $w = xyz$ satisfying all of the following conditions:

$$\text{(a) } |yz| \leq N(M), \text{ (b) } |y| > 0, \textit{ and } \text{(c) } xz \in L(M).$$

Proof. Let $M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, \delta, >)$ be an ORWW-automaton, where $Q = \{q_0, q_1, \ldots, q_k\}$ and $\Gamma = \{s_1, s_2, \ldots, s_n\}$. Without loss of generality we may assume that M accepts at the left end of its tape, that is, it accepts in state q_0 with its window containing the left sentinel $\triangleright$ and the first two symbols of the proper tape inscription. The transition relation δ of M can be represented by a finite set of five-tuples of the form (q, a, b, c, r), where $q \in Q$, $a \in \Gamma \cup \{\rhd\}$, $b \in \Gamma$, $c \in \Gamma \cup \{\lhd\}$, and $r \in Q \cup \Gamma \cup \{\mathsf{Accept}\}$. Here a five-tuple (q, a, b, c, q') with $q' \in Q$ represents the move-right operation $(q', \mathsf{MVR}) \in \delta(q, abc)$, and a five-tuple (q, a, b, c, d) with $d \in \Gamma$ represents the rewrite/restart operation $d \in \delta(q, abc)$, and analogously, for $(q, a, b, c, \mathsf{Accept})$. As $|Q|= k + 1$ and $|\Gamma|= n$, we see that this set consists of $K \leq (k + 1) \cdot n \cdot (n + 1)^2 \cdot (k + n + 2)$ five-tuples. We introduce a new alphabet $\Omega = \{t_1, t_2, \ldots, t_K\}$ the symbols of which are in 1-to-1 correspondence to these five-tuples.

We now consider a shortest accepting computation C of M on an input $w_m \in \Sigma^m$, where m is sufficiently large. To each number $j = 1, 2, \ldots, m - 1$, we associate a word $x_j \in \Omega^*$ such that x_j describes the sequence of operations that are performed within the computation C at position $m + 1 - j$. Thus, we obtain a sequence of words $X_C = (x_1, x_2, \ldots, x_{m-1})$ over Ω.

Claim. $|x_1|\leq n - 1$, and for all $j = 2, \ldots, m - 1$, $|x_{j-1}|\leq |x_j|\leq j \cdot (n - 1)$.

Proof. We proceed by induction on j. For $j = 1$, x_j is the sequence of operations that are performed within C at the right-most position. Hence, x_1 only consists of rewrite/restart operations, and as $|\Gamma|= n$, it follows that $|x_1|\leq (n - 1)$.

Now, assume that $|x_{j-1}|\leq (j - 1) \cdot (n - 1)$ has been established for some $j \geq 2$. We consider the sequence of operations that is described by the word x_j. This word describes all the rewrite/restart operations and all the move-right operations that are performed within C at position $m+1-j$. Each move-right operation executed at this position leads to an operation that is performed at its right neighbor, that is, at position $m + 2 - j$. Hence, the number of these move-right operations is exactly $|x_{j-1}|$, which implies that $|x_{j-1}|\leq |x_j|\leq |x_{j-1}|+(n - 1) \leq j \cdot (n - 1)$. □

Finally, we extend each word x_j into $a_j x_j s_j$, where a_j is the input letter at position $m + 1 - j$ and s_j is the letter from Γ into which the input symbol a_j is being rewritten by the sequence of operations x_j. Now we consider the sequence $(a_1x_1s_1, a_2x_2s_2, \ldots, a_{m-1}x_{m-1}s_{m-1})$ over $\Omega \cup \Gamma$.

To determine the constant $N(M)$ we use Higman's theorem [15] and the corresponding *Length function H* from [20] (see also [43]). Higman's Lemma states that an infinite sequence $w_0, w_1, \ldots$ of finite words over a finite alphabet contains a pair of words $w_i \sqsubseteq w_j$ with $i < j$ where the words are partially ordered by the subsequence relation. The Length function takes a growth function as an argument and calculates the length of a longest bad sequence, that is, a sequence where we do not find an increasing pair of words. Let $H(2, n+1, \Omega \cup \Gamma)$ be the maximal positive integer N such that there exists a sequence $\sigma_1, \sigma_2, \ldots, \sigma_N$ of words over $\Omega \cup \Gamma$ such that $|\sigma_j| \leq j \cdot (n+1)$ for all $j \geq 1$ and σ_{j_1} is not a scattered subsequence of σ_{j_2} for any indices $1 \leq j_1 < j_2 \leq N$. It is shown in [20] that H is a total recursive function.

Now, we choose $N(M) = H(2, n+1, \Omega \cup \Gamma) + 2$. We see from Claim 1 that $|a_j x_j s_j| = |x_j| + 2 \leq j \cdot (n-1) + 2 \leq j \cdot (n+1)$ for all $j \geq 1$. If $m \geq N(M)$, then we see from the definition of the function H that there are indices $1 \leq j_1 < j_2 \leq m-1$ such that $a_{j_1} x_{j_1} s_{j_1}$ is a scattered subsequence of $a_{j_2} x_{j_2} s_{j_2}$. Thus, $a_{j_1} = a_{j_2}$, $s_{j_1} = s_{j_2}$, and if $x_{j_1} = t_1 t_2 \ldots t_r$ for some $r \geq 1$ and $t_1, t_2, \ldots, t_r \in \Omega$, then x_{j_2} can be written as $x_{j_2} = y_0 t_1 y_1 t_2 y_2 \ldots y_{r-1} t_r y_r$ for some $y_0, y_1, y_2, \ldots, y_r \in \Omega^*$.

The subsequence of rewrite operations of x_{j_1} rewrites the input letter a_{j_1} into the letter s_{j_1}, and the subsequence of rewrite operations of x_{j_2} rewrites the input letter $a_{j_2} = a_{j_1}$ into the letter $s_{j_2} = s_{j_1}$. Hence, as the former is a scattered subsequence of the latter, it follows that actually the same rewrite operations occur in x_{j_2} and in x_{j_1}, and they occur in the same order. In particular, this means that the factors $y_0, y_1, y_2, \ldots, y_r$ only consist of move-right operations.

Finally we take x to be the prefix of w_m up to position $m + 1 - j_2$, y to be the factor of w_m from positions $m + 2 - j_2$ to $m + 1 - j_1$, and z to be the remaining suffix of w_m. Then $w_m = xyz$, $|yz| \leq N(M)$ and $|y| = j_2 - j_1 > 0$. Finally, let $w' = xz$. We will show that M has an accepting computation C' for input w'. This computation is obtained from the computation C as follows.

Let $(C_1, C_2, \ldots, C_\mu)$ be the sequence of cycles of the computation C. For $j = 1, 2, \ldots, \mu$, let C_j be the cycle currently considered.

1. If the rewrite step in C_j is performed on the prefix of length $m + 1 - j_2$ of the current tape, then we append C_j to C'. This includes in particular all those cycles that include a rewrite operation at position $m + 1 - j_2$, that is, the rewrite operations encoded within the word x_{j_2}.

2. If C_j includes a move-right step at position $m + 1 - j_2$ that contributes a letter to one of the factors $y_0, y_1, \ldots, y_r$ of x_{j_2}, then C_j is not appended

to C'.

3. Finally, if C_j includes a move-right operation at position $m+1-j_2$ that corresponds to a letter t_l of the word x_{j_2} for some $1 \leq l \leq r$, then we combine the initial part of this cycle, up to the point where the operation t_l is executed at position $m+1-j_2$, with the final part of the cycle which starts with this very operation at position $m+1-j_1$. The resulting cycle C'_j is appended to C'. As $x_{j_1} = t_1 t_2 \dots t_r$ is a subsequence of x_{j_2}, C'_j is indeed a valid cycle of M.

The computation C' is completed by appending the accepting tail of C to it. Then it is easily checked that C' is indeed an accepting computation of M on input w'. This completes the proof of the Cut-and-Paste Lemma. □

A corollary of the Cut-and-Paste Lemma is that the deterministic linear language

$$L = \{a^n b^n \mid n \geq 1\}$$

cannot be accepted by an ORWW-automaton.

Proposition 4.3.2. *Let M be an ORWW-automaton and let*

$$L = \{a^n b^n \mid n \geq 1\}.$$

Then $L(M) \neq L$.

Proof. We prove this proposition by contradiction. Let us assume that M is an ORWW-automaton that accepts the language L. Now, let $N_C(M) =: K$ be the constant from the Cut-and-Paste Lemma for M. If we look at the word $w = a^K b^K$, we see that it satisfies the condition $|w| \geq N_C(M)$. Hence, there must exist a factorization $w = xyz$ such that $y = b^j$ with $1 \leq j \leq N_C(M)$. According to the Cut-and-Paste Lemma the word $a^n b^{n-j}$ must be accepted by M. Therefore, there cannot be an automaton M that accepts L. □

In order to establish some non-closure properties we consider the example language $L_{\leq} = \{\, a^m b^n \mid 1 \leq m \leq n \,\}$. Clearly, $L_{\leq}$ is a linear language [16].

Proposition 4.3.3. *$L_{\leq} \notin \mathcal{L}(\mathsf{ORWW})$.*

Proof. Assume to the contrary that there exists an ORWW-automaton $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ such that $L(M) = L_{\leq}$, and let $N(M)$ be the corresponding constant from the Cut-and-Paste Lemma (Theorem 4.3.1). We consider the word $w = a^{N(M)}b^{N(M)} \in L_{\leq}$. Then $w = xyz$ such that $|yz| \leq N(M)$, $|y| > 0$, and $w' = xz \in L_{\leq}$. From the two factorizations of w we see that $y = b^i$ for some $i > 0$, which implies that $w' = a^{N(M)}b^{N(M)-i} \notin L_{\leq}$, a contradiction. Thus, it follows that the language $L_{\leq}$ is not accepted by any ORWW-automaton. $\square$

As $\mathcal{L}(\mathsf{ORWW})$ contains a language that is not growing-context-senstive (see Lemma 4.1.2), but does not contain a language that is linear (see Proposition 4.3.2) we obtain the following result.

Corollary 4.3.4. *The language class $\mathcal{L}(\mathsf{ORWW})$ is incomparable to the language classes LIN, CFL, and GCSL with respect to inclusion.*

Finally, Proposition 4.3.3 allows us to derive the following non-closure properties.

Theorem 4.3.5. *The language class $\mathcal{L}(\mathsf{ORWW})$ is neither closed under the operation of reversal nor under complementation.*

Proof. In Example 4.1.1 it has been shown that $L_{\geq} = \{\, b^m a^n \mid m \geq n \geq 1 \,\}$ is accepted by some ORWW-automaton. As $L_{\leq} = L_{\geq}^R$, Proposition 4.3.3 implies that $\mathcal{L}(\mathsf{ORWW})$ is not closed under the operation of reversal.

Obviously, also the language $L'_{\geq} = \{\, a^m b^n \mid m \geq n \geq 1 \,\}$ is accepted by some ORWW-automaton. Assume that its complement $(L'_{\geq})^c$ is accepted by some ORWW-automaton. Then also the language $(L'_{\geq})^c \cap (a^+ \cdot b^+)$ is accepted by some ORWW-automaton, since all regular languages are accepted by these automata and $\mathcal{L}(\mathsf{ORWW})$ is closed under intersection. However, $(L'_{\geq})^c \cap (a^+ \cdot b^+) = \{\, a^m b^n \mid 1 \leq m < n \,\}$, and in analogy to the proof of Proposition 4.3.3 it can be shown that this language is not accepted by any ORWW-automaton, either. Thus, it follows that $\mathcal{L}(\mathsf{ORWW})$ is not closed under complementation. $\square$

Although $\mathcal{L}(\mathsf{ORWW})$ is an abstract family of languages that is incomparable to LIN, CFL, and GCSL, we have the following decidability result.

Theorem 4.3.6. *The emptiness problem for ORWW-automata is decidable.*

Proof. Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORWW-automaton, and let $N(M)$ be the corresponding constant from the Cut-and-Paste Lemma (Theorem 4.3.1). As shown in the proof of that lemma, $N(M) = H(2, n+1, \Omega \cup \Gamma) + 2$, where H is the Length function corresponding to Higman's theorem, which is a recursive function [20]. It now follows that $L(M) \neq \emptyset$ iff $L(M)$ contains a word of length at most $N(M)$. □

Before we finally derive the Pumping-Lemma we need some more notation for ORWW-automata.

Definition 4.3.7. *Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORWW-automaton, $w \in L(M)$, and C be an accepting computation of M on input w. With each integer, $1 \leq i \leq |w|$, we associate the sequence of operations σ_i^C like we did in the proof of Theorem 4.3.1.*

Now the pattern $\tau_i^C \in \Omega^*$ *is the word that is obtained from σ_i^C by squeezing consecutive identical letters into a single letter.*

Observe that it is only MVR operations that may be squeezed when forming a pattern as a rewrite operation changes the tape content which makes it impossible to execute the same rewrite operation at this position again.

Example 4.3.8. *Let M be an ORWW-automaton that can execute the following accepting computation:*

$$\begin{aligned} q_0\rhd aaa\lhd \ &\vdash_M \ q_0\rhd a_1aa\lhd \ \vdash_M \ \rhd q_0a_1aa\lhd \ \vdash_M \ \rhd a_1q_0aa\lhd \\ &\vdash_M \ q_0\rhd a_1aa_1\lhd \ \vdash_M \ \rhd q_0a_1aa_1\lhd \ \vdash_M \ \rhd a_1q_0aa_1\lhd \ \vdash_M \ \mathsf{Accept}. \end{aligned}$$

The computation C consists of two cycles and an accepting tail that are described by the following sequences of operations:

$$\begin{aligned} c_1 &= (q_0, \rhd, a, a, a_1), \\ c_2 &= (q_0, \rhd, a_1, a, q_0), (q_0, a_1, a, a, q_0), (q_0, a, a, \lhd, a_1), \\ c_3 &= (q_0, \rhd, a_1, a, q_0), (q_0, a_1, a, a_1, q_0), (q_0, a, a_1, \lhd, \mathsf{Accept}). \end{aligned}$$

For the first position, we therefore get the sequence of operations

$$\sigma_1^C = (q_0, \rhd, a, a, a_1)(q_0, \rhd, a_1, a, q_0)(q_0, \rhd, a_1, a, q_0),$$

which yields the pattern $\tau_1^C = (q_0, \rhd, a, a, a_1)(q_0, \rhd, a_1, a, q_0)$*, while for the second position we get the sequence of operations*

$$\sigma_2^C = (q_0, a_1, a, a, q_0)(q_0, a_1, a, a_1, q_0) = \tau_2^C.$$

For the third position we have $\sigma_3^C = (q_0, a, a, \lhd, a_1)(q_0, a, a_1, \lhd, \mathsf{Accept}) = \tau_3^C$*.*

For two pattern τ_1^C and τ_2^C, we write $\tau_1^C \sqsubseteq \tau_2^C$ if τ_1^C is a *scattered subword* of τ_2^C, that is, if $\tau_1^C = \omega_1\omega_2 \dots \omega_m$ for some $\omega_1, \omega_2, \dots, \omega_m \in \Omega$, then there are words $y_0, y_1, \dots, y_m \in \Omega^*$ such that $\tau_2^C = y_0\omega_1 y_1 \omega_2 y_2 \dots y_{m-1}\omega_m y_m$. The following technical lemma is the main step towards the proof of the Pumping Lemma.

Lemma 4.3.9. *Let* $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ *be an ORWW-automaton that accepts at the left sentinel, let* C_{xz} *be an accepting computation of* M *for the input* xz*, and let* C_{uv} *be an accepting computation of* M *for the input* uv*. If the pattern* $\tau_{|u|}^{C_{uv}}$ *of the computation* C_{uv} *at position* $|u|$ *is a scattered subword of the pattern* $\tau_{|x|}^{C_{xz}}$ *of the computation* C_{xz} *at position* $|x|$*, that is,* $\tau_{|u|}^{C_{uv}} \sqsubseteq \tau_{|x|}^{C_{xz}}$*, and if these two patterns contain the same rewrite operations, then* $xv \in L(M)$*.*

Proof. We construct an accepting computation C' for the input xv from the given computations C_{xz} and C_{uv}. The sequences of cycles $(C_1, C_2, \dots, C_m)$ of C_{xz} and $(D_1, D_2, \dots, D_n)$ of C_{uv} are considered as working lists that are used for constructing the cycles of C' that have their rewrite operations in the prefix x or in the suffix v of the input xv, respectively. As $\tau_{|u|}^{C_{uv}} \sqsubseteq \tau_{|x|}^{C_{xz}}$, these patterns can be written as $\tau_{|u|}^{C_{uv}} = t_1 t_2 \dots t_r$ with $t_1, t_2, \dots, t_r \in \Omega$ and $\tau_{|x|}^{C_{xz}} = y_0 t_1 y_1 \dots y_{r-1} t_r y_r$ for some $y_0, y_1, \dots, y_r \in \Omega^*$ (see Fig. 4.1). As both patterns contain the same rewrite operations, the factors $y_0, y_1, \dots y_r$ only consist of MVR operations.

For constructing the computation C' on input xv, we start by taking C' to be the empty sequence of cycles. Now we consider the cycles of C_{xz} one after another (see Fig. 4.2).

Let C_i be the cycle currently considered.

- If C_i is a *short cycle*, that is, a cycle that executes a rewrite step within a proper prefix of x, then we just append it to C' (see the cycle c_2 in Fig. 4.2).

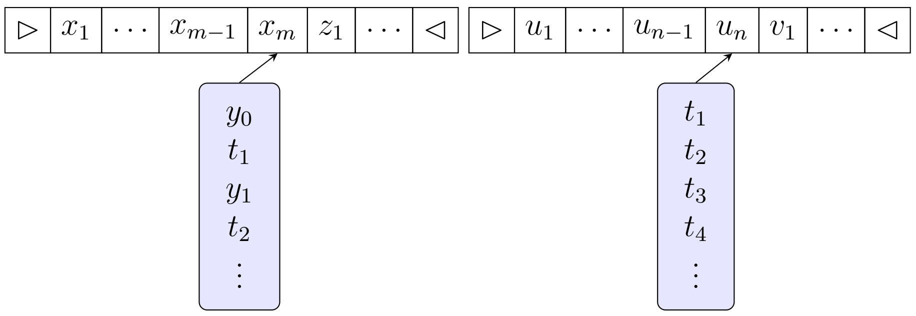
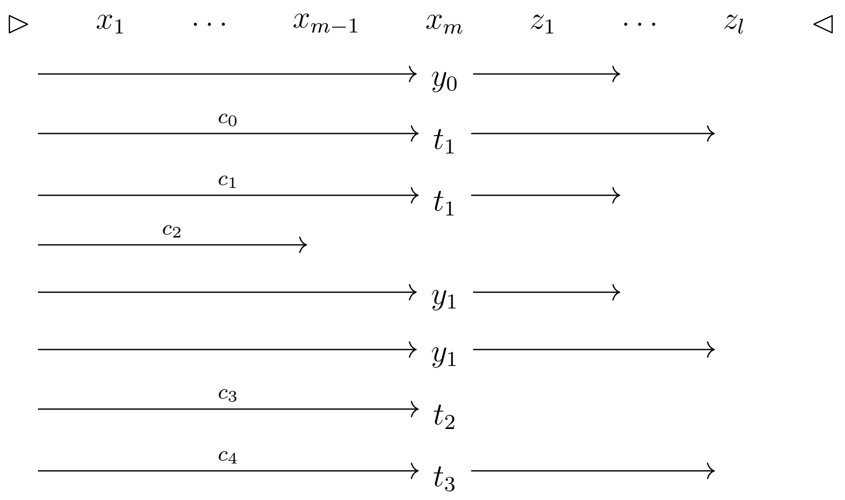


Figure 4.1: The inputs xz and uv with the patterns $\tau_{|x|}^{C_{xz}}$ (left) and $\tau_{|u|}^{C_{uv}}$ (right)



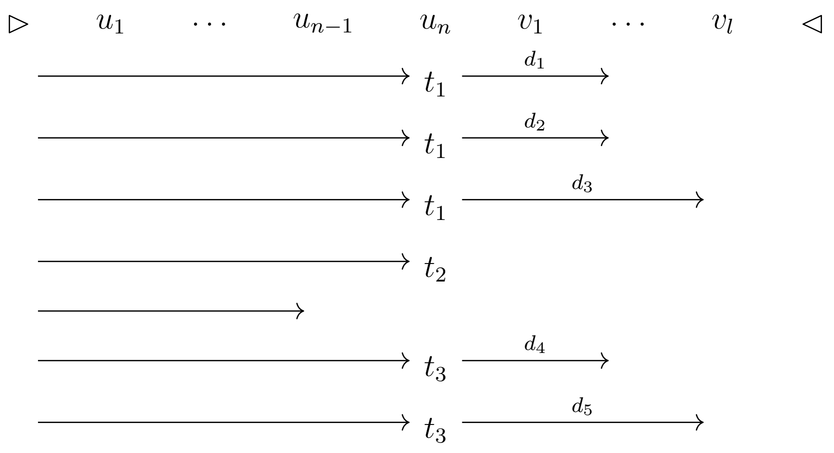


Figure 4.2: The cycles of the computations C_{xz} (top) and C_{uv} (bottom). Each arrow represents a (partial) cycle.

- If C_i contains a rewrite operation at position $|x|$, then this operation corresponds to the letter t_l for some $1 \leq l \leq r$. Again we append this cycle to C' (see the cycle c_3). As both patterns contain the same rewrite operations, which must occur in the same relative order in both patterns, we see that the rewrite operation t_l can also be executed at this point in the computation C'.

- If C_i is a cycle that executes a rewrite step within the suffix z of xz, then this cycle contains a MVR operation at position $|x|$. If this operation does not correspond to one of the letters $t_1, t_2, \ldots, t_r$ in the pattern $\tau_{|x|}^{C_{xz}}$, we skip this cycle without appending it to C'.

- Finally, let C_i be a cycle that executes a rewrite step within the suffix z of xz, but the MVR operation executed at position $|x|$ corresponds to the letter t_l for some $1 \leq l \leq r$. By c_0 we denote the prefix of the cycle C_i up to position $|x|-1$. Further, let $D_{i_1}, D_{i_2}, \ldots, D_{i_\nu}$ be all those cycles of C_{uv} that contain the MVR operation t_l at position $|u|$, and for all $1 \leq j \leq \nu$, let d_j be the suffix of the cycle D_{i_j} that starts with the operation t_l at position $|u|$. We now combine the prefix c_0 of C_i with the suffix d_j of D_{i_j} for all $1 \leq j \leq \nu$ (see c_0 and d_1, d_2, d_3 in Fig. 4.2). As the same operation t_l is applied in the cycle C_i at position $|x|$ as in the cycles $D_{i_1}, D_{i_2}, \ldots, D_{i_\nu}$ at position $|u|$, we see that $c_0d_1, c_0d_2, \ldots, c_0d_\nu$ is a sequence of possible cycles of M. We can append this sequence of cycles to C'.

- Any further cycle C_{i+s}, $s \geq 1$, that also executes a MVR operation at position $|x|$ which corresponds to the letter t_l of the pattern $\tau_{|x|}^{C_{xz}}$, is skipped (see c_1 in Fig. 4.2).

Fig. 4.3 illustrates the above construction. Finally, the computation C' is completed by attaching the accepting tail computation from C_{xz} to it. Recall that M accepts with the left sentinel in its window. It is now easily seen that C' is an accepting computation of M for the input xv. $\square$

Next, we consider a special case of the above lemma.

Lemma 4.3.10. *Let $M = (Q, \Sigma, \Gamma, \triangleright, \triangleleft, q_0, \delta, >)$ be an ORWW-automaton that accepts at the left sentinel, let $w \in L(M)$, let C be an accepting computation*

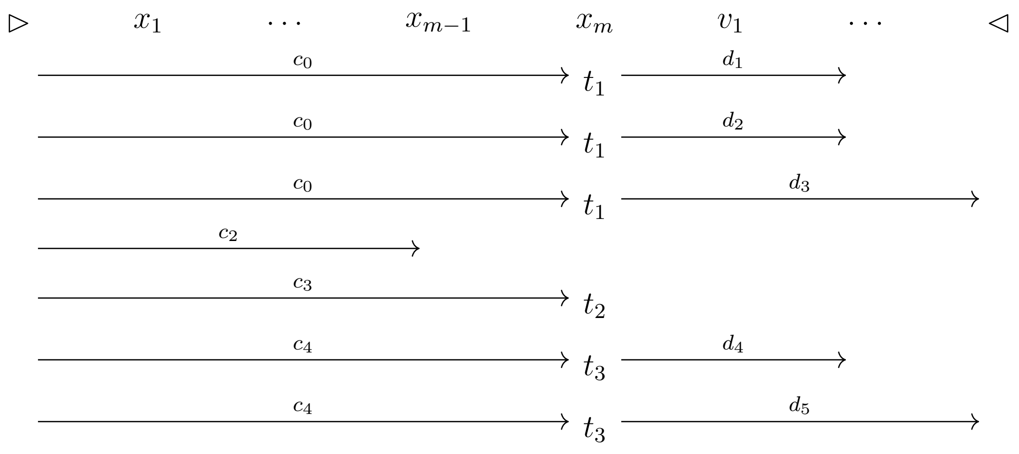


Figure 4.3: The computation C' for input xv

of M for the input w, and let $1 \leq i < j \leq |w|$ be indices such that $\tau_i^C(w) \sqsubseteq \tau_j^C(w)$ and these two patterns contain the same rewrite operations. Then w can be factored as $w = xyz$, where $|x| = i$ and $|y| = j - i$, such that $xyyz \in L(M)$. In fact, there exists an accepting computation C' for $xyyz$ satisfying $\tau_i^{C'}(xyyz) = \tau_j^{C'}(xyyz)$.

Proof. If we choose $x_1 = xy$, $y_1 = z$, $u_1 = x$, and $v_1 = yz$, we can apply Lemma 4.3.9 to the factorizations $w = xyz = x_1y_1$ and $w = xyz = u_1v_1$. Thus, we obtain an accepting computation C' of M for the input $x_1v_1 = xyyz$. From the construction of C' in the proof of the above lemma we see that the patterns $\tau_i^{C'}(xyyz)$ and $\tau_j^{C'}(xyyz)$ coincide. □

Finally, we need the following notion that has already been considered in [32] under the name of *det-MVR$_1$-form* for general restarting automata.

Definition 4.3.11. *An ORWW-automaton $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ is said to have* deterministic MVR operations *if, for all $q \in Q$ and all $a, b, c \in \Gamma \cup \{\rhd, \lhd\}$, $\delta(q, abc)$ contains at most a single MVR operation.*

Lemma 4.3.12. *For each ORWW-automaton $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, there exists an ORWW automaton M' with deterministic MVR operations that accepts the same language as M. If M accepts at the left sentinel, then so does M'.*

Proof. Using a variant of the well-known powerset construction, the ORWW-automaton M' can be defined as $M' = (2^Q, \Sigma, \Gamma, \rhd, \lhd, \{q_0\}, \delta', >)$, where, for all $\emptyset \neq S \subseteq Q$ and all $a, b, c \in \Gamma \cup \{\rhd, \lhd\}$,

$$T_{(S,abc)} = \{\, (q, \mathsf{MVR}) \mid q \in Q, \exists s \in S : (q, \mathsf{MVR}) \in \delta(s, abc) \,\},$$

and

$$\delta'(S, abc) = \begin{cases} \mathsf{Accept} & \text{if } \exists s \in S : \delta(s, abc) = \mathsf{Accept}, \\ \left(\bigcup_{s \in S} \delta(s, abc) \cap \Gamma\right) \cup T_{(S,abc)} & \text{otherwise.} \end{cases}$$

□

The next lemma is the second technical main result.

Lemma 4.3.13. *Let M be an ORWW-automaton with deterministic MVR operations that accepts at the left sentinel. From M a constant $N(M) > 0$ can be computed such that, for each $w \in L(M)$ satisfying $|w| \geq N(M)$ and each accepting computation C of M on input w, there are indices $1 \leq i < j \leq |w|$ such that $\tau_i^C(w) \sqsubseteq \tau_j^C(w)$ and these patterns contain the same rewrite operations.*

Proof. Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORWW-automaton with deterministic MVR operations that accepts at the left sentinel, and let $n = |\Gamma|$. Further, let $w \in L(M)$ and let C be an accepting computation of M on input w. The MVR operations executed at a position $1 \leq k \leq |w| - 1$ only depend on the prefix of length $k + 1$ of w. As M has deterministic MVR operations, the MVR operation that can be executed at position k is uniquely determined by that prefix, if it exists at all. For this reason a different MVR operation can become applicable at position k only if that prefix has been modified by a rewrite operation. This, however, can happen at most $(k + 1) \cdot (n - 1)$ times. Therefore, the pattern $\tau_k^C(w)$ contains at most $(k + 1) \cdot (n - 1) + 1$ MVR operations. Additionally, it contains at most $n - 1$ rewrite operations. Therefore, $\tau_k^C(w)$ has length at most $(k+1) \cdot (n-1) + n + 1 = k \cdot (n-1) + 2n$. Finally, we extend each pattern $\tau_k^C(w)$ into the word $\eta_k^C(w) = a_k \tau_k^C(w) s_k$ where a_k is the input letter at position k and s_k is the final letter produced by C at position k. Higman's theorem [15] (see, also [20] and [43]) implies that there exists a computable constant $N(M)$ such that, if $|w| \geq N(M)$, then there are indices $1 \leq i < j \leq N(M)$ such that $\eta_i^C(w)$ is a scattered subsequence of

$\eta_j^C(w)$. This means that $a_i = a_j$ and $s_i = s_j$, and that $\tau_i^C(w)$ is a scattered subsequence of $\tau_j^C(w)$. As in both positions the letter $a_i = a_j$ is rewritten into the letter $s_i = s_j$, and as each rewrite operation at position i occurs in $\tau_i^C(w)$ and therewith also in $\tau_j^C(w)$, we see that $\tau_i^C(w)$ and $\tau_j^C(w)$ contain exactly the same rewrite operations. □

Now we can state and prove the announced Pumping Lemma.

Theorem 4.3.14 (Pumping Lemma)**.** *For each ORWW-automaton M there exists a computable constant $N_p(M) > 0$ such that each word $w \in L(M)$, $|w| \geq N_p(M)$, has a factorization $w = xyz$ satisfying all of the following*

$$\text{(a) } |xy| \leq N_p(M), \text{ (b) } |y| > 0, \textit{ and } \text{(c) } xy^m z \in L(M) \textit{ for all } m \geq 1.$$

Proof. Let M be an ORWW automaton. By Lemma 3.4.3 we may assume that M only accepts at the left sentinel. Further, by Lemma 4.3.12, we can convert M into an equivalent ORWW-automaton M_1 that is MVR-deterministic and that only accepts at the left sentinel. Then Lemma 4.3.13 implies that a constant $N_p(M)$ can be computed such that, for each $w \in L(M_1) = L(M)$ satisfying $|w| \geq N_p(M)$, and each accepting computation C of M_1 on input w, there are indices $1 \leq i < j \leq N_p(M)$ such that $\tau_i^C(w) \sqsubseteq \tau_j^C(w)$ and these patterns contain the same rewrite operations. Hence, by Lemma 4.3.10, w can be factored as $w = xyz$ such that $|xy| \leq N_p(M)$, $|y| > 0$, $xyyz \in L(M_1) = L(M)$, and $\tau_{|x|}^{C'}(xyyz) = \tau_{|xy|}^{C'}(xyyz)$, where C' is the accepting computation of M_1 for input $xyyz$ that is obtained from the computation C. Using Lemma 4.3.10 repeatedly we obtain that $xy^m z \in L(M_1) = L(M)$ holds for all $m \geq 1$. □

4.4 Applications of the Pumping Lemma

In Theorem 4.3.6 we have used the Cut-and-Paste Lemma to prove that emptiness is decidable for ORWW-automata. Here we show that also finiteness is decidable for ORWW-automata using both, the Cut-and-Paste Lemma and the Pumping Lemma.

Theorem 4.4.1. *The following* finiteness problem *is decidable:*

INSTANCE: *An ORWW-automaton M.*

QUESTION: *Is the language $L(M)$ finite?*

Proof. Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORWW-automaton, let $N_c(M)$ be the corresponding constant from the Cut-and-Paste Lemma for M, and let $N_p(M)$ be the corresponding constant from the Pumping Lemma for M. We claim that $L(M)$ is finite if and only if it does not contain any word w such that $N_p(M) \leq |w| \leq N_p(M) + N_c(M)$.

Indeed, if $L(M)$ contains a word w such that $N_p(M) \leq |w| \leq N_p(M)+N_c(M)$, then the Pumping Lemma tells us that $L(M)$ is infinite. Conversely, if $L(M)$ is infinite, then it contains a word w of length at least $N_p(M)$. Assume that w is the shortest word with these properties. If $|w| \leq N_p(M) + N_c(M)$, then there is nothing to prove. On the other hand, if $|w| > N_p(M) + N_c(M)$, then we can apply the Cut-and-Paste Lemma to w, which yields a factorization $w = xyz$ such that $|yz| \leq N_c(M)$, $|y| > 0$, and $xz \in L(M)$. Thus, $|w| > |xz| = |w| - |y| \geq |w| - N_c(M) > N_p(M)$, which contradicts our choice of w. Hence, we see that $L(M)$ is infinite iff it contains a word w such that $N_p(M) \leq |w| \leq N_p(M) + N_c(M)$. □

Unfortunately, it is still an open question whether universality is decidable for ORWW-automata in general, but for the case of ORWW-automata with a unary input alphabet we have a decision algorithm.

Theorem 4.4.2. *The following problem is decidable:*

INSTANCE: *An ORWW-automaton M with a unary alphabet $\Sigma = \{a\}$.*

QUESTION: *Is $L(M) = \Sigma^*$?*

Proof. Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORWW-automaton, let $N_p(M)$ be the corresponding constant from the Pumping Lemma for M. We claim that $L(M)$ is universal if and only if $a^n \in L(M)$ for all $n < N_p(M) + N_p(M)!$.

If $L(M) = \Sigma^*$, M obviously satisfies our universality condition. If $L(M) \neq \Sigma^*$, then there exists a number $k \in \mathbb{N}$ such that $a^k \notin L(M)$. Assume that k is minimal. If we have the case that $k < N_p(M) + N_p(M)!$, M does not satisfy our universality condition. Now, let us assume that $k \geq N_p(M) + N_p(M)!$. Then $a^{k-N_p(M)!} \in L(M)$ as k is minimal. As $|a^{k-N_p(M)!}| = k - N_p(M)! \geq N_p(M) + N_p(M)! - N_p(M)! \geq N_p(M)$, we can apply the pumping lemma. Thus, there exists a constant $1 \leq c \leq N_p(M)$ such that $a^{k-N_c(M)!+c\cdot i} \in L(M)$ for all $i \in \mathbb{N}$.

Choose $i = \frac{N_p(M)!}{c}$. Then $a^{k-N_p(M)!+c\cdot\frac{N_p(M)!}{c}} = a^{k-N_p(M)!+N_p(M)!} = a^k \in L(M)$ which is obviously a contradiction. $\square$

Theorem 4.4.3. *The following problem is decidable:*

INSTANCE: *An ORWW-automaton M with input alphabet Σ, and three word $u, v, x \in \Sigma^*$.*

QUESTION: *Is the language $L = \{ux^iv \mid i \geq 1\}$ contained in $L(M)$?*

Proof. We use some closure properties to reduce this problem to the known unary universality problem.

Let $\varphi : \{a, b, c\} \to \Sigma^*$ with $\varphi(a) = u, \varphi(b) = x$ and $\varphi(c) = v$. Then let $L_1 = \{ab^ic \mid i \geq 1\}$, which is a regular language, $\psi : \{a, b, c\} \to \{a\}$ with $\psi(a) = a, \psi(b) = a$ and $\psi(c) = a$ be another morphism, and let $L_2 = \{\lambda, a, aa\}$ be another regular language. Then we can determine an ORWW-automaton M' for the unary language $L' = \psi(\varphi^{-1}(L(M)) \cap L_1) \cup L_2$. Now, we claim that L is contained in $L(M)$ if and only if L' is universal. If L is contained in $L(M)$, then $L' = \{a^n \mid n \in \mathbb{N}\}$ by construction.

Now if L' is universal, then $\psi(\varphi^{-1}(L(M)) \cap L_1) \supseteq \{a^i \mid i \geq 3\}$. Thus $\{ab^ic \mid i \geq 1\} \subseteq \varphi^{-1}(L(M)$, which implies that $L = \{ux^iv \mid i \geq 1\} \subseteq L(M)$.

Therefore, this decision problem is decidable, as it can be reduced to the unary universality problem. $\square$

The next result, which is also derived from the Pumping Lemma, shows that ORWW-automata only accept unary languages that are regular.

Theorem 4.4.4. *For each ORWW-automaton M, if the language $L(M)$ is unary, then it is already regular.*

Proof. Let M be an ORWW-automaton with input alphabet $\Sigma = \{a\}$, and let $\alpha = N_p(M)$ be the constant from the Pumping Lemma for M. For all integers d satisfying $0 \leq d < \alpha!$, we let $S_d \subseteq \mathbb{N}$ be defined as follows:

$$S_d := \{\, n \geq \alpha \mid n \equiv d \mod \alpha!\,, a^n \in L(M)\,\}.$$

By definition $\{\, a^n \mid n \in S_d \,\} \subseteq L(M)$ for all $0 \leq d < \alpha!$. On the other hand, if $a^n \in L(M)$ for some $n \geq \alpha$, then there exists an integer d, $0 \leq d < \alpha!$, such that $n \equiv d \bmod \alpha!$. By the Pumping Lemma there also exists an integer c, $0 < c \leq \alpha$, such that $a^{n+c\cdot i} \in L(M)$ for all $i \in \mathbb{N}$. It follows that $a^{n+\alpha!\cdot i} \in L(M)$

for all $i \in \mathbb{N}$, as c is a proper divisor of $\alpha!$. Hence, it follows that $n \in S_d$. Moreover, it follows that $n + i \cdot \alpha! \in S_d$ for all $i \in \mathbb{N}$.

If $S_d \neq \emptyset$, it can be represented as the linear set $S_d = \{\min(S_d) + i \cdot \alpha! \mid i \in \mathbb{N}\}$. Therefore, if $\psi : \Sigma^* \to \mathbb{N}$ denotes the Parikh mapping defined by $a^n \mapsto n$ $(n \geq 0)$, then

$$\psi(L(M)) = \{\, n < \alpha \mid a^n \in L(M) \,\} \cup \bigcup_{d=0}^{\alpha!-1} S_d,$$

which shows that $\psi(L(M))$ is a semi-linear subset of $\mathbb{N}$. Thus, it follows that $L(M)$ is indeed a regular language. □

Remark 4.4.5. *Actually, it can be shown that a regular expression can be determined for the language $L(M)$ of an ORWW-automaton M that has a unary input alphabet.*

Proof. The reason why we cannot easily determine the regular language with the help of the proof above is that we do not know whether S_d is empty or not. We can decide this by constructing the ORWW-automaton M_d for the language $L(M) \cap T_d$, where T_d is the regular language

$$T_d = \{a^n \mid n > \alpha, n \equiv d \mod \alpha! \,\}.$$

We can then check $L(M_d)$ for emptiness and S_d is empty if and only if $L(M_d)$ is empty. We know from the Cut-and-Paste Lemma that $\min(S_d)$ is smaller than $N_c(M_d)$ if S_d is not empty. This can be easily checked. Thereby, a regular expression for $L(M)$ can be determined. □

As unary ORWW languages are regular and we cannot imagine an ORWW language that is not semi-linear, we may conjecture that all ORWW languages are semi-linear.

One promising idea seems to use the definition of letter-equivalence to a regular language. If we assume that every tape field content is changed at most once, we could rearrange the letters in their rewrite order which would lead to a regular language. Unfortunately, this idea was unrewarding so far as we were not able to formalize and generalize it.

4.5 Conclusion

The nondeterministic ordered restarting automata form a very interesting model with an unusual language class, which still forms an abstract family of languages. We can decide emptiness and finiteness by means of our special pumping lemma. Unfortunately, it is not yet known whether universality is decidable or not. In order to prove the decidability of the universality problem it is unlikely that using the Pumping Lemma or the Cut-and-Paste Lemma will be sufficient as also the context-free languages satisfy some kind of pumping lemma. Their universality problem is known to be undecidable. One possible approach would be to work directly with the patterns and perhaps argue with antichains. But perhaps universality is not at all decidable. The question of semi-linearity is also still open and we still have not even touched the topic of transducers. Thus, there are still enough interesting unsolved problems.

Chapter 5

Ordered RRWW-Automata

As we have seen ORWW-automata have very nice properties especially regarding decidability. They can recognize all regular languages and even accept some languages that are not growing context-sensitive. Nevertheless, these automata do not accept some languages we would like them to accept, e.g. the very simple deterministic linear language $L = \{a^n b^n \mid n \in \mathbb{N}\}$. One reason for this restriction is the strong emphasis on the left sentinel.

After exploring different variations that accept this language, we opted for the classic variation, the ORRWW-automaton. As the name suggests, this is an ORWW-automaton with separate rewrite and restart operations instead of the combined rewrite/restart operation. This means that we do not have to execute a restart operation immediately after a rewrite, but we can execute additional MVR operations in between a rewrite and the subsequent restart.

Like before, we start with the definition of the generic automaton, the nondeterministic automaton with states. Then we look at how we realize limitations and how this affects the language class.

5.1 Definition

An *ordered RRWW-automaton* (ORRWW-automaton) is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet such that $\Sigma \subseteq \Gamma$, the letters $\rhd, \lhd \notin \Gamma$ serve as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $>$ is a strict *partial* *ordering* on Γ, and

$$\delta : (Q \times ((\Gamma \cup \{\rhd\})^{\leq 1} \cdot \Gamma \cdot (\Gamma \cup \{\lhd\}))) \cup \{(q_0, \rhd\lhd)\} \\ \to 2^{(Q\times(\{\mathsf{MVR}\}\cup\Gamma))\cup\{\mathsf{Restart}\}} \cup \{\mathsf{Accept}\}$$

is the *transition relation*, which describes four different types of transition steps:

(1) A *move-right step* has the form $(q', \mathsf{MVR}) \in \delta(q, a_1a_2a_3)$, where $q, q' \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$ and $a_2, a_3 \in \Gamma$. It causes M to shift the window one position to the right and to change to state q'. Observe that no move-right step is possible, if the window contains the right delimiter $\lhd$.

(2) A *rewrite step* has the form $(q', b) \in \delta(q, a_1a_2a_3)$, where $q, q' \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2, b \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$ such that $a_2 > b$ holds. It causes M to replace the letter a_2 in the middle of its window by the letter b, to move the window one position to the right, and to change to state q'.

(3) A *restart step* has the form $\mathsf{Restart} \in \delta(q, a_1a_2a_3)$, where $q \in Q$, $a_1, a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$, or $a_1 \in \Gamma$, $a_2 = \lhd$, and $a_3 = \lambda$ (the empty word). It causes M to restart, that is, the window is moved back to the left end of the tape, and M is reset to the initial state q_0.

(4) An *accept step* has the form $\delta(q, a_1a_2a_3) = \mathsf{Accept}$, where $a_1 \in \Gamma \cup \{\rhd\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$, or $a_1 \in \Gamma$, $a_2 = \lhd$, and $a_3 = \lambda$. It causes M to halt and accept. In addition, we allow an accept step of the form $\delta(q_0, \rhd\lhd) = \mathsf{Accept}$.

If $\delta(q, u) = \emptyset$ for some pair (q, u), then M halts, when it is in state q with u in its window, and we say that M *rejects* in this situation. The letters in $\Gamma \smallsetminus \Sigma$ are called *auxiliary letters*. If $|\delta(q, u)| \leq 1$ for all pairs (q, u), then M is a *deterministic* ORRWW-automaton (det-ORRWW-automaton). In that case we also use the partial transition function

$$\delta : (Q \times ((\Gamma \cup \{\rhd\})^{\leq 1} \cdot \Gamma \cdot (\Gamma \cup \{\lhd\}))) \cup \{(q_0, \rhd\lhd)\} \\ \rightsquigarrow (Q \times (\{\mathsf{MVR}\} \cup \Gamma)) \cup \{\mathsf{Restart}, \mathsf{Accept}\}$$

Observe that for general RRWW-automata, a rewrite operation $(q', v) \in \delta(q, u)$ replaces the factor u by the word v, changes the state to q', and moves the window immediately to the right of v. In our case this would mean that a rewrite operation $(q', b) \in \delta(q, abc)$ should move the window *three* steps to the right, as it rewrites the factor abc into the word $ab'c$. However, for the stateless variant (that is, q_0 is the only state) this would mean that after a rewrite no information on the new letter would be available to the automaton, and therefore we have chosen the above interpretation for the rewrite step.

A *configuration* of an ORRWW-automaton M is a word of the form $\alpha q \beta$, where $q \in Q$ is a state, and $\alpha\beta \in \{\rhd\} \cdot \Gamma^* \cdot \{\lhd\}$ such that $|\beta| \geq 2$, and either $\alpha = \lambda$ and $\beta \in \{\rhd\} \cdot \Gamma^+ \cdot \{\lhd\}$ or $\alpha \in \{\rhd\} \cdot \Gamma^*$ and $\beta \in \Gamma^+ \cdot \{\lhd\}$; here $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first three letters of β or all of β, if $|\beta| < 3$. In addition, we admit the configuration $q_0 \rhd \lhd$. A *restarting configuration* has the form $q_0 \rhd w \lhd$; if $w \in \Sigma^*$, then $q_0 \rhd w \lhd$ is also called an *initial configuration.* Further, we use Accept to denote the *accepting configurations*, which are those configurations that M reaches by an accept step. A configuration of the form $\alpha q \beta$ such that $\delta(q, \beta_1) = \emptyset$, where β_1 is the current content of the window, 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 the set of configurations, and the *computation relation* $\vdash_M^*$ of M is the reflexive and transitive closure of $\vdash_M$.

Any computation of an ORRWW-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 step is performed, which replaces a letter by a smaller one. After that further MVR steps may follow until, finally, a restart step is executed and thus, a new restarting configuration is reached. If no further restart operation is performed, any computation necessarily finishes in a halting configuration – such a phase is called a *tail.* It is required that each cycle contains *exactly* one rewrite step, and a tail may contain at most a single rewrite step. By $\vdash_M^c$ we denote the execution of a complete cycle, and $\vdash_M^{c^*}$ is the reflexive transitive closure of this relation. It can be seen as the *rewrite relation* that is realized by M on the set of restarting configurations.

An input $w \in \Sigma^*$ is accepted by M, if there is a computation of M which starts with the initial configuration $q_0 \rhd w \lhd$ and ends with an accept step. By $L(M)$ we denote the language $L(M) = \{\, w \in \Sigma^* \mid q_0 \rhd w \lhd \vdash_M^* \mathsf{Accept} \,\}$.

As each cycle contains a rewrite operation, which replaces a letter a by a letter b that is strictly smaller than a with respect to the given ordering $>$, we see that each computation of M on an input of length n consists of at most $(|\Gamma|-1) \cdot n$ many cycles. Thus, M can be simulated by a nondeterministic single-tape Turing machine in time $O(n^2)$. The following example illustrates the way in which a deterministic ORRWW-automaton works.

Example 5.1.1. *For $m \geq 1$, let $L_{\mathrm{ch},m}$ be the following language:*

$$L_{\mathrm{ch},m} = \{\, w_1 w_2 \dots w_n \in \{a,b\}^n \mid n \geq 2m \text{ and } w_m = w_{n+1-m} = w_n \,\},$$

that is, a word w of length $n \geq 2m$ belongs to this language iff the m-th letter and the m-th last letter both coincide with the last letter of w. We define a det-ORRWW-automaton $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ by taking $Q = \{q_0, q_1, \dots, q_{m-1}, q_a, q_b, q_r\}$, $\Sigma = \{a, b\}$ and $\Gamma = \Sigma \cup \{a_1, b_1, x_2, x_3, \dots, x_{m-1}\}$, by defining the partial ordering $>$ through $a > a_1 > x_i$ and $b > b_1 > x_i$ for all $i = 2, 3, \dots, m-1$, and by specifying the transition function through the following table, where $c, d, e, f \in \Sigma$:

$$\begin{array}{lcll}
\delta(q_0, \rhd cd) & = & (q_1, \mathsf{MVR}), & \\
\delta(q_i, cde) & = & (q_{i+1}, \mathsf{MVR}) & \textit{for all } i = 1, 2, \dots, m-2, \\
\delta(q_{m-1}, dce) & = & (q_c, \mathsf{MVR}), & \\
\delta(q_c, def) & = & (q_c, \mathsf{MVR}), & \\
\delta(q_c, de\lhd) & = & (q_r, e_1), & \\
\delta(q_c, def_1) & = & (q_r, x_2), & \\
\delta(q_c, dex_i) & = & (q_r, x_{i+1}) & \textit{for all } i = 2, 3, \dots, m-2, \\
\delta(q_c, dcx_{m-1}) & = & (q_c, \mathsf{MVR}), & \\
\delta(q_c, cx_{m-1}x_{m-2}) & = & (q_c, \mathsf{MVR}), & \\
\delta(q_c, x_{i+2}x_{i+1}x_i) & = & (q_c, \mathsf{MVR}) & \textit{for all } i = 2, 3, \dots, m-3, \\
\delta(q_c, x_3x_2c_1) & = & (q_c, \mathsf{MVR}), & \\
\delta(q_c, x_2c_1\lhd) & = & \mathsf{Accept}, & \\
\delta(q_r, x_{i+2}x_{i+1}x_i) & = & (q_r, \mathsf{MVR}) & \textit{for all } i = 2, 3, \dots, m-3, \\
\delta(q_r, x_3x_2c_1) & = & (q_r, \mathsf{MVR}), & \\
\delta(q_r, x_2c_1\lhd) & = & \mathsf{Restart}, & \\
\delta(q_r, c_1\lhd) & = & \mathsf{Restart}. &
\end{array}$$

Using its states M counts from left to right until it sees the m-th letter, say c, which it then remembers in its state. Then it rewrites the last $m-1$ letters from right to left, rewriting the last letter, say $w_n = d$, into d_1, and the letters $w_{n-1}, w_{n-2}, \ldots, w_{n+2-m}$ into $x_2, x_3, \ldots, x_{m-1}$. This is possible because the m-th last letter of w is not to the left of the m-th letter. Finally, it checks whether w_{n+1-m}, which is the letter immediately before x_{m-1}, coincides with $w_m = c$. In the affirmative, M moves to the right, where it compare $w_m = w_{n+1-m} = c$ to the last letter d (or rather its encoding d_1). If a positive result is returned, then M accepts. It is easily seen that $L(M) = L_{\mathrm{ch},m}$ holds.

The ORWW-automaton studied in the previous chapters differs from the ORRWW-automaton in that the rewrite and restart operations are combined into a joint operation, that is, such an automaton restarts immediately after executing a rewrite step. Obviously, (deterministic) ORWW-automata can be simulated by (deterministic) ORRWW-automata. Thus, it follows that all regular languages are accepted by det-ORRWW-automata. However, in the deterministic case also the converse holds and deterministic ORRWW-automata can be simulated by deterministic ORWW-automata.

Lemma 5.1.2. *Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be an ORRWW-automaton. Then there exists an ORRWW-automaton M' that accepts at the right sentinel.*

Proof. The construction and proof is the same as for the ORWW-automaton (see the remark after Lemma 4.2.1). ☐

Theorem 5.1.3. $\mathcal{L}(\mathsf{det\text{-}ORRWW}) = \mathsf{REG}$.

Proof. Let $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ be a det-ORRWW-automaton, and let $L = L(M)$. Without loss of generality we can assume that M performs restart and accept operations only at the right delimiter $\lhd$. We present a det-ORWW-automaton $M' = (Q', \Sigma, \Gamma', \rhd, \lhd, q_0, \delta', >')$ that simulates M. Then $L(M') = L(M) = L$, which implies that L is a regular language. Each cycle of a computation of M is of the following form, where $u, v \in \Gamma^*$, $a, b, b', c, d, e \in \Gamma$, and $q_1, q_2, q_3 \in Q$:

$$\begin{aligned} q_0 \rhd uabcvde\lhd &\vdash^{1+|u|}_{\mathrm{MVR}} \rhd uq_1abcvde\lhd \vdash_{\mathrm{Rewrite}} \rhd uaq_2b'cvde\lhd \\ &\vdash^{|v|+2}_{\mathrm{MVR}} \rhd uab'cvq_3de\lhd \vdash_{\mathrm{Restart}} q_0 \rhd uab'cvde\lhd, \end{aligned}$$

and in the next cycle M moves its window at least until it contains the newly written letter b' before the next rewrite step can be executed. In order for the det-ORWW-automaton M' to be able to correctly simulate the above cycle, M' must ensure (or verify) in some way that after the above rewrite operation M will eventually perform a restart. For this we let M' perform some kind of preprocessing during which it encodes certain additional information on its tape.

For each word $w \in \Gamma^+$, $|w| \geq 2$, and each letter $a \in \Gamma$, we define two sets $Q^{(a)}_{\text{rs}}(w)$ and $Q^{(a)}_{+}(w)$ as follows, where $w = w_1bc$ for $b, c \in \Gamma$:

$$\begin{aligned} Q^{(a)}_{\text{rs}}(w) &= \{\, p \in Q \mid \rhd paw\lhd \vdash^{|w|-1}_{\text{MVR}} \rhd aw_1p'bc\lhd \vdash_{\text{Restart}} q_0 \rhd aw\lhd \,\} \text{ and} \\ Q^{(a)}_{+}(w) &= \{\, p \in Q \mid \rhd paw\lhd \vdash^{|w|-1}_{\text{MVR}} \rhd aw_1p'bc\lhd \vdash_{\text{Accept}} \mathsf{Accept} \,\}. \end{aligned}$$

Now if the det-ORWW-automaton M' is to simulate the above cycle of M, then from the fact that $q_2 \in Q^{(b')}_{\text{rs}}(cvde)$ it sees that M will actually restart at the right end of the tape, and hence, it can safely perform the same rewrite operation and restart. Accordingly, we define a precomputation for M' that assigns, from right to left, the collection of sets $(Q^{(a)}_{\text{rs}}(z), Q^{(a)}_{+}(z))_{a\in\Gamma}$ to the first letter z_1 of each suffix z of the given input w. Thus, we define

$$\Gamma' = \Sigma \cup \{\, (A, (Q^{(a)}_1, Q^{(a)}_2)_{a\in\Gamma}) \mid A \in \Gamma, Q^{(a)}_1, Q^{(a)}_2 \subseteq Q \,\},$$

take $Q' = Q \cup \{q_C\}$, define $>'$ by taking $A >' (A, (Q^{(a)}_1, Q^{(a)}_2)_{a\in\Gamma})$ for all $A \in \Gamma$ and all $Q^{(a)}_1, Q^{(a)}_2 \subseteq Q$ and $(A, (Q^{(a)}_1, Q^{(a)}_2)_{a\in\Gamma}) >' (B, (P^{(a)}_1, P^{(a)}_2)_{a\in\Gamma})$ if $A > B$.

The transition function can now be defined in such a way that M' first encodes the information on the sets $(Q_{\text{rs}}, Q_+)_{a\in\Gamma}$ proceeding from right to left until it detects the position, say i, at which the next rewrite operation of M is to be simulated. Based on the information from the encoded sets of states, it then simulates this rewrite step, updating also the information on the stored sets of states of M at the current position. For this, it can extract the information on the corresponding sets from the symbol stored at position $i+1$. Observe that in the next cycle, M cannot execute a rewrite step until it has the newly written symbol in its window, that is, not to the left of position $i-1$.

It can be shown that in this way M' can simulate M correctly, implying that $L(M') = L(M)$. Thus, $\mathcal{L}(\mathsf{det\text{-}ORRWW}) = \mathcal{L}(\mathsf{det\text{-}ORWW})$, which implies that $\mathcal{L}$(det-ORRWW) coincides with the class REG of regular languages. $\square$

5.2 Stateless ORRWW-Automata

For restarting automata in general, each RR-variant is at least as powerful as the corresponding R-variant, but for stateless automata the situation is not that obvious. The feature of continuing to read the tape after a rewrite step has been executed is problematic for these automata, as they cannot distinguish between the phase of a cycle *before* the rewrite step and the phase *after* the rewrite step on their own. Clearly, this distinction is important, since no rewrite steps may appear in the latter phase. For general restarting automata, this is avoided by using states, but how to deal with this situation for stateless RR-automata?

In [23] this problem has been addressed for various types of deterministic restarting automata, and two options for dealing with it have been proposed. First, one can interpret any additional rewrite step within a cycle as a reject. Also a cycle without a rewrite step is regarded as a rejecting computation. However, this approach amounts to an *external* supervisor that aborts the computation in these unwanted situations. Here we rather follow the second option presented in [23] in which two *phases* of each cycle are distinguished: the first phase, which ends with the execution of a rewrite operation, and the second phase, which starts after the execution of a rewrite operation and ends with either a restart or an accept step. These two phases are realized by providing two separate transition functions. In [23] the corresponding stateless restarting automata are called *two-phase* restarting automata, but as we will only deal with this type of stateless ORRWW-automata, we just call them *stateless ORRWW-automata* (stl-ORRWW-automata). Formally these automata are defined as follows.

Definition 5.2.1. *A* stl-ORRWW-automaton *is described by a 7-tuple* $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_1, \delta_2, >)$*, where* Σ*,* Γ*,* $\rhd$*,* $\lhd$*, and* $>$ *are defined as for ORRWW-automata, and*

$$\delta_1 : ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \cup \{\rhd\lhd\} \to 2^{\Gamma \cup \{\mathsf{MVR}\}} \cup \{\mathsf{Accept}\}$$

and

$$\delta_2 : (\Gamma^{\leq 2} \cdot (\Gamma \cup \{\lhd\})) \to 2^{\{\mathsf{MVR}\}\cup\{\mathsf{Restart}\}} \cup \{\mathsf{Accept}\}$$

are the transition relations. *Here it is required that $b > b'$ holds for each rewrite instruction $b' \in \delta_1(abc)$.*

Based on our previous definitions we do not allow any other operations if we can execute an Accept operation.

A configuration of M is written as a pair (α, β), where $\alpha\beta$ is the current content of the tape and the window contains the prefix of β. On input a word $w \in \Sigma^+$, the computation starts with the initial configuration $(\lambda, \rhd w \lhd)$. First the transition relation δ_1 is used until either an accept instruction is reached, a rewrite instruction $b' \in \Gamma$ is reached, or the window contains a word for which δ_1 is undefined. In the first case, M accepts, in the second case the letter in the middle of the window is replaced by the letter b', the window is moved one step to the right, and the computation is continued using the transition relation δ_2. Finally, in the third case M simply halts without accepting. The transition relation δ_2, which is used in the second phase of a cycle *after* the execution of a rewrite step, shifts the window to the right until either an accept instruction is executed, and then M accepts, until a restart instruction is executed, which resets the window to the left end of the tape and starts the next cycle, or until a window content is reached for which δ_2 is undefined. In the latter case M halts without accepting. For $w = \lambda$, there either is no applicable operation for the configuration $(\lambda, \rhd\lhd)$, or $\delta_1(\rhd\lhd) = \mathsf{Accept}$. By $L(M)$ we denote again the language consisting of all input words that M accepts.

Theorem 5.2.2. $\mathcal{L}(\mathsf{stl\text{-}det\text{-}ORRWW}) = \mathcal{L}(\mathsf{stl\text{-}ORRWW}) = \mathsf{REG}$.

Proof. If L is a regular language, then there exists a stl-det-ORWW-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ for L. From M we obtain an equivalent stl-det-ORRWW-automaton $M' = (\Sigma, \Gamma, \rhd, \lhd, \delta_1, \delta_2, >)$ by defining $\delta_1 = \delta$ and $\delta_2(abc) = \mathsf{MVR}$ and $\delta_2(de\lhd) = \{\mathsf{Restart}\}$ for all $a, b, c \in \Gamma$ and $de \in \Gamma^{\leq 2}$. Hence, $\mathsf{REG} \subseteq \mathcal{L}(\mathsf{stl\text{-}det\text{-}ORRWW}) \subseteq \mathcal{L}(\mathsf{stl\text{-}ORRWW})$ follows.

Conversely, let $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_1, \delta_2, >)$ be a stl-ORRWW-automaton. We will prove that $L(M)$ is a regular language. Let $w \in L(M)$, let m be an integer such that $1 \leq m \leq |w|$, and let C be an accepting computation of

M on input w. During this computation M executes certain operations at position m, that is, when position m is in the middle of the window of M. These operations may include rewrite steps $b' \in \delta_1(abc)$, move-right steps $\mathsf{MVR} \in \delta_1(abc)$ or $\mathsf{MVR} \in \delta_2(abc)$, restart steps $\mathsf{Restart} \in \delta_2(abc)$, and accept steps $\delta_1(abc) = \mathsf{Accept}$ or $\delta_2(abc) = \mathsf{Accept}$. As by a rewrite step, the letter in the middle of the window is replaced by a smaller letter with respect to $>$, the rewrite steps that occur at position m are obviously ordered. Before the first rewrite step, and between two successive rewrite steps, a sequence of move-right steps may occur at position m. Here again, an ordering is induced by $>$, if one of the letters at position $m-1$ or $m+1$ is rewritten during this part of the computation. What remains is the case that a sequence of move-right steps occurs at position m that all have the same left-hand side, that is, they are all applied to the same window content abc. Some of them may be δ_1-steps, while others may be δ_2-steps. By associating the number 1 with a positive number of δ_1-steps and the number 2 with a positive number of δ_2-steps, we can assign a *type* $t \in (1 \cdot (2 \cdot 1)^* \cdot \{2, \lambda\}) \cup (2 \cdot (1 \cdot 2)^* \cdot \{1, \lambda\})$ to this sequence.

We claim that by rearranging the corresponding cycles of the computation C, this computation can be transformed into a computation C' for which the type of any sequence of move-right steps with the same window content at position m is of length at most three, that is, only the six types from the set $T = \{1, 2, 1 \cdot 2, 2 \cdot 1, 1 \cdot 2 \cdot 1, 2 \cdot 1 \cdot 2\}$ occur. Additionally, we demand that all cycles associated to a single number 1 appear back to back in the computation. We call this computation normalized.

But let us first look at the proof of the claim. Let $o_1, o_2, \ldots, o_k$ be a sequence of move-right operations that occur within an accepting computation C of M on input xz at the position $|x|$, where all these operations are applied to the same window content abc. Some of these operations may be δ_1-operations, which then belong to cycles of C that perform their rewrite steps within the suffix z, and others may be δ_2-operations, which belong to cycles of C that perform their rewrite steps within the prefix x. By assigning the letter 1 to each maximal subsequence that only consists of δ_1-steps, and by assigning the letter 2 to each maximal subsequence that only consists of δ_2-steps, we obtain the *type* $t = t(o_1, o_2, \ldots, o_k) \in (1 \cdot (2 \cdot 1)^* \cdot \{2, \lambda\}) \cup (2 \cdot (1 \cdot 2)^* \cdot \{1, \lambda\})$ of this sequence. We claim that by rearranging the corresponding cycles of C, we obtain an equivalent computation C' such that the corresponding sequence of move-right operations is of a type $t' \in T$.

This can be proven as follows. Assume that $t \notin T$. Then $t = 1 \cdot 2 \cdot 1 \cdot 2 \cdot \hat{t}$ or $t = 2 \cdot 1 \cdot 2 \cdot 1 \cdot \hat{t}$ for some type $\hat{t}$. Let us first assume the former. Then the sequence of cycles of C that correspond to the prefix of type $1 \cdot 2 \cdot 1 \cdot 2$ of the sequence $o_1, o_2, \ldots, o_k$ of move-right operations performs the following actions on x_1z_1, where x_1 and z_1 are the successors of x and z that are produced by the cycles proceeding the ones containing these move-right operations, and where we overline the part that is being rewritten by the corresponding cycles:

$$x_1\overline{z_1} \quad \to^+_{\mathrm{MVR}_{\delta_1}} \quad \overline{x_1}z_2 \quad \to^+_{\mathrm{MVR}_{\delta_2}} \quad x_2\overline{z_2} \quad \to^+_{\mathrm{MVR}_{\delta_1}} \quad \overline{x_2}z_3 \quad \to^+_{\mathrm{MVR}_{\delta_2}} \quad x_3z_3.$$

During the first group of cycles containing the move-right steps of δ_1, M reads across the prefix x_1 and rewrites z_1 into z_2, while in the second group of cycles containing the move-right steps of δ_1, M reads across the prefix x_2 and rewrites z_2 into z_3. Obviously, these cycles can be combined such that M reads across x_1 and rewrites z_1 all the way into z_3. During the first group of cycles containing the move-right steps of δ_2, M rewrites x_1 into x_2, and it moves across the suffix z_2, while in the second group of cycles containing the move-right steps of δ_2, M rewrites x_2 into x_3, and it moves across the suffix z_3. Again these cycles can be combined such that M rewrites x_1 all the way to x_3, while moving right across the suffix z_3, that is, we obtain the following partial computation:

$$x_1\overline{z_1} \quad \to^+_{\mathrm{MVR}_{\delta_1}} \quad x_1\overline{z_2} \quad \to^+_{\mathrm{MVR}_{\delta_1}} \quad \overline{x_1}z_3 \quad \to^+_{\mathrm{MVR}_{\delta_2}} \quad \overline{x_2}z_3 \quad \to^+_{\mathrm{MVR}_{\delta_2}} \quad x_3z_3.$$

The sequence of move-right steps at position $|x|$ of this computation has type $1 \cdot 2$ only, that is, we can replace C by an equivalent computation C' such that the corresponding sequence of move-right steps has the type $1 \cdot 2 \cdot \hat{t}$.

For the case $t = 2 \cdot 1 \cdot 2 \cdot 1 \cdot \hat{t}$, it can be shown analogously that there exists an equivalent computation such that the corresponding sequence of move-right steps has the type $2 \cdot 1 \cdot \hat{t}$. Iteratively it follows that there exists an equivalent computation for which the corresponding sequence of cycles has a type $t' \in T$.

Now let Ω be the following extended set of operations of M, where $a \in$

$\Gamma \cup \{\rhd\}$, $b, b' \in \Gamma$ and $c \in \Gamma \cup \{\lhd\}$:

$$\Omega = \{\, (a,b,c,b') \mid b' \in \delta_1(abc) \,\} \cup \{\, (a,b,c,t) \mid t \in T \,\} \cup \\ \{\, (a,b,c,-) \mid \delta_2(abc) \ni \mathsf{Restart} \,\} \cup \{\, (a,b,c,+_i) \mid i = 1,2,\ \delta_i(abc) = \mathsf{Accept} \,\}.$$

Then the sequence of operations that are executed by M during the computation C' at position m can be described by a word $A_m^{C'}(w) \in \Omega^*$. In fact, as at most $|\Gamma|-1$ rewrite steps can occur in this sequence, $A_m^{C'}(w)$ is of length $\mathcal{O}(|\Gamma|)$.

With each word $w \in \Sigma^+$, we now associate the sets

$$S_1(w) = \left\{ A_{|w|}^{C}(wz) \mid z \in \Sigma^*, C \text{ a valid comp. for } wz \text{ accepting at a pos.} \le |w| \right\}$$

and

$$S_2(w) = \left\{ A_{|w|}^{C}(wz) \mid z \in \Sigma^*, C \text{ a valid comp. for } wz \text{ accepting at a pos.} > |w| \right\}.$$

For proving the regularity of $L(M)$, we use the Myhill-Nerode Theorem. We will show that there are no distinguishing extensions for $x, y \in \Sigma^+$ if $S_i(x) = S_i(y)$ for $i = 1,2$. Accordingly, let $z \in \Sigma^*$ such that $xz \in L(M)$. Then there exists a number $i \in \{1,2\}$ and a normalized accepting computation C_{xz} of M such that $A_{|x|}^{C_{xz}}(xz) = A \in S_i(x)$. As $S_i(y) = S_i(x)$, there exists a word $u \in \Sigma^*$ and a normalized accepting computation C_{yu} of M such that $A_{|y|}^{C_{yu}}(yu) = A$. We claim that there is also an accepting computation C' of M for the word yz. We consider the sequences of cycles of C_{yu} and C_{xz} which serve as working lists for constructing the cycles of C' that have their rewrite operations in the y-part and the z-part, respectively. We construct the computation C' for the word yz as follows. We divide the cycles into groups according to the different types of MVR-patterns. All consecutive cycles that contribute to the same number 1 in one pattern form a group of type 1. All consecutive cycles from the first to the last cycle that contribute to a single number 2 belong to a group of type 2. Note that such a group may include short cycles that do not have a δ_2-MVR operation at the border. Additionally, each cycle that does a rewrite at the border forms a rewrite group. We see that we have the same groups in C_{yu} and C_{xz}. There may be some short cycles that do not belong to a group.

We consider the group of cycles or ungrouped short cycles of C_{yu} one after another. If we have a short cycle, we just append it to C' as the u-part is not involved. If we have a cycle of the rewrite group, we take the cycle up to position $|y|$ and complement it with the second part of the corresponding cycle of C_{xz}. If we have a group of type 1, we take the part up to position $|y|-1$ of the first cycle and call it c_0. Then, we take all last parts starting at position $|x|$ of the cycles of the corresponding group of C_{xz} and call them $c_1, \dots, c_k$. Finally, we append the cycles $c_0c_1, \dots, c_0c_k$ to C'. The computation C' stays valid, as these new cycles do not make any changes in the y-part. Therefore, it is possible to execute $c_0c_1, \dots, c_0c_k$ in that order. If we have a group of type 2, we take the last part of the first cycle of the corresponding group of C_{xz} starting at position $|x|$ and call it c_t. Then, we replace the last part of each cycle of the current group of C_{yu} starting at position $|y|$ by c_t if the cycle has a length $> |y|$. Finally, we append all these cycles to C'. The computation C' stays valid, as there are no changes in the $|z|$-part.

This construction ends as soon as an accepting tail is encountered, which happens eventually as the computations C_{xz} and C_{yz} either both accept in the z-part or they both accept to the left of this part.

As there are only finitely many words $A_m^{C'}(w) \in \Omega^*$ that can occur as descriptions of sequences of operations of M, there are only finitely many different sets $S_1(x)$ and $S_2(x)$. Hence, the Myhill Nerode relation of $L(M)$ is of finite index, which means that $L(M)$ is indeed a regular language. $\square$

We have seen that the stateless ORRWW automata again characterize the regular languages. This may come as a surprise at first, since there is no obvious direct simulation of a valid computation using stateless ORWW automata or NFA. This is only possible by normalizing the computation cycles.

Now, however, we turn our attention to a variant that can obviously represent more than the regular languages, namely the non-deterministic ORRWW automata with states.

5.3 Nondeterministic ORRWW-Automata

In Section 4.2 it was shown that the class $\mathcal{L}(\mathsf{ORWW})$ of languages that are accepted by ORWW-automata is an abstract family of languages that is in addition closed under intersection, but that is not closed under reversal and complementation. In addition, this class contains a language that is not even growing context-sensitive, while at the same time it does not even contain the deterministic linear language $\{ a^n b^n \mid n \geq 1 \}$. Thus, this class is incomparable to the (deterministic) linear languages, the (deterministic) context-free languages, and the growing context-sensitive languages. However, the following inclusion obviously holds.

Lemma 5.3.1. $\mathcal{L}(\mathsf{ORWW}) \subseteq \mathcal{L}(\mathsf{ORRWW})$.

Actually, this inclusion is proper. While the context-free language $L = \{ a^n b^n \mid n \geq 1 \}$ is not accepted by any ORWW-automaton, all context-free languages are accepted by ORRWW-automata.

Theorem 5.3.2. $\mathsf{CFL} \subsetneq \mathcal{L}(\mathsf{ORRWW})$.

Proof. Let $L \subseteq \Sigma^+$ be a context-free language that does not contain the empty word. Then there exists a grammar $G = (N, \Sigma, P, S)$ in quadratic Greibach normal form that generates L (see, e.g., [2]), that is, each production of P is of the form $A \to a$, $A \to aB$, or $A \to aBC$, where $a \in \Sigma$ and $A, B, C \in N$.

The language L is accepted by the ORRWW-automaton $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$ that works as follows. Given an input $w \in \Sigma^+$, M guesses a leftmost derivation of w in G. In each step of this derivation the next symbol a of w must be produced by applying a corresponding production. Thus, the symbol a must be marked as having been read, and the nonterminals produced by that step are written in reverse order on the tape, where the rightmost of these nonterminals is marked as being 'active.' Now in each phase a production $A \to aT$ is chosen that has the active nonterminal A as its left-hand side and that produces the next symbol a of w on its right-hand side. Then the symbol a is replaced by an encoding of T^R, which is tagged. After that the marked nonterminal is deleted, the tag is being removed, and the rightmost of the remaining nonterminals is again marked as 'active.'

Formally the automaton M is defined as follows. We take the tape alphabet

$$\Gamma = \Sigma \cup \{[\lambda], [B], [BC], \overline{[B]}, \overline{[BC]}, \widetilde{[\lambda]}, \widetilde{[B]}, \widetilde{[BC]} \mid B, C \in N\}$$

with the partial ordering

$$a > \widetilde{[CB]} > \widetilde{[C]} > \widetilde{[\lambda]} > [CB] > \overline{[CB]} > [C] > \overline{[C]} > [\lambda].$$

The set of states Q and the transition relation δ are defined in such a way that, in each cycle, M scans its tape from left to right and executes one of the following steps depending on the form of the word on the tape. In the following we have $a \in \Sigma$, $B, C, D, E \in N$, and we use R to denote the set $R = \{\, [\lambda], [A], [AB] \mid A, B \in N \,\}$:

1. If the word on the tape is of the form $\{a\} \cdot \Sigma^*$, then M can replace a by $[\lambda]$, if there is a production $S \to a$, by $[B]$, if there is a production $S \to aB$, or by $[CB]$, if there is a production $S \to aBC$.

2. If the word on the tape is of the form $R^* \cdot \{[B], [CB]\} \cdot \{[\lambda]\}^* \cdot \Sigma^*$, then M replaces $[B]$ (or $[CB]$) by $\overline{[B]}$ (or $\overline{[CB]}$).

3. If the word on the tape is of the form $R^* \cdot \{\overline{[B]}, \overline{[CB]}\} \cdot \{[\lambda]\}^* \cdot \{a\} \cdot \Sigma^*$, then M can replace a by $\widetilde{[\lambda]}$, if there is a production $B \to a$, by $\widetilde{D}$, if there is a production $B \to aD$, or by $\widetilde{ED}$, if there is a production $B \to aDE$.

4. If the word on the tape is of the form
$$R^* \cdot \{\overline{[B]}, \overline{[CB]}\} \cdot \{[\lambda]\}^* \cdot \{\widetilde{[\lambda]}, \widetilde{[D]}, \widetilde{[ED]}\} \cdot \Sigma^*,$$
then M replaces $\overline{[B]}$ (or $\overline{[CB]}$) by $[\lambda]$ (or $[C]$).

5. If the word on the tape is of the form $R^* \cdot \{[\lambda]\}^* \cdot \{\widetilde{[\lambda]}, \widetilde{[D]}, \widetilde{[ED]}\} \cdot \Sigma^*$, then M removes the tilde from $\widetilde{[\lambda]}$, $\widetilde{[D]}$, or $\widetilde{[ED]}$.

6. Finally, M halts and accepts, if the tape contains a word from $\{[\lambda]\}^*$.

It can easily be seen that $L(M) = L$, as the transitions of M are in close correspondence to the productions of G.

First, we show that $L(G) \subseteq L(M)$. We claim that, for each derivation $S \to^i \alpha\beta$, where $\alpha \in \Sigma^+$ and $\beta \in N^*$, and every word $v \in \Sigma^*$, there is a valid computation $q_0 \rhd \alpha v \lhd \vdash_M^{c^*} q_0 \rhd \gamma v \lhd$ of M, where $\gamma \in R^*$ and $\beta = f(\gamma)^R$. Here $f : R^* \to V^*$ denotes the morphism that is defined as follows:

$$f(x) = \begin{cases} B, & \text{if } x = [B], \\ BC, & \text{if } x = [BC], \\ \lambda, & \text{otherwise.} \end{cases}$$

We proceed by induction on the length i of the derivation. For $i = 1$, we have the derivation $S \to a$, $S \to aB$ or $S \to aBC$, which leads to the computation $q_0 \rhd av\lhd \vdash_M^c q_0 \rhd[\lambda]v\lhd$, $q_0 \rhd av\lhd \vdash_M^c q_0 \rhd[B]v\lhd$, or $q_0 \rhd av\lhd \vdash_M^c q_0 \rhd[CB]v\lhd$.

For $i > 1$, the derivation is of the form $S \to^{i-1} \alpha A\beta \to \alpha aT\beta$, where $T \in \{\lambda, B, BC\}$. By the induction hypothesis, we get the computation

$$q_0 \rhd \alpha av\lhd \vdash_M^{c^*} q_0 \rhd \gamma' av\lhd,$$

where $f(\gamma')^R = A\beta$. From the definition of the transitions of M, we see that

$$q_0 \rhd\gamma' av\lhd \vdash_M^c q_0 \rhd\overline{\gamma'}av\lhd \vdash_M^c q_0 \rhd\overline{\gamma'}[\widetilde{T^R}]v\lhd \vdash_M^c q_0 \rhd\gamma[\widetilde{T^R}]v\lhd \vdash_M^c q_0 \rhd\gamma[T^R]v\lhd,$$

where $\overline{\gamma'}$ is the word that is created from γ' by overlining the last letter that is not $[\lambda]$, and γ is created from $\overline{\gamma'}$ by deleting the overlined letter, where $f(\gamma)^R = \beta$ holds. This gives our result, as $f(\gamma[T^R])^R = (f(\gamma) \cdot f([T^R]))^R = T\beta$. For each $w \in L(G)$, we have $S \to^* w$, and therefore, we have a valid computation $q_0 \rhd w\lhd \vdash_M^{c^*} q_0 \rhd\gamma\lhd$ such that $f(\gamma)^R = \lambda$. This can only be the case if $\gamma \in \{[\lambda]\}^*$, which leads to the accepting computation

$$q_0 \rhd w\lhd = q_0 \rhd w_1 \dots w_n\lhd \vdash_M^{c^*} q_0 \rhd [\lambda]^n\lhd \vdash_M^c \mathsf{Accept}.$$

Next, we show that $L(M) \subseteq L(G)$, which can be shown similarly. By induction on k, we show that, for every computation

$$q_0 \rhd w_1 \dots w_n\lhd \vdash_M^{c^*} q_0 \rhd \gamma w_{n-k+1} \dots w_n\lhd,$$

where $\gamma \in R^*$, there is a leftmost derivation $S \to^* w_1 \dots w_k\beta$ such that $f(\gamma) = \beta^R$. The case $k = 1$ follows directly from the definition of the automaton. For $k > 1$, we have the valid computation

$$q_0 \rhd w_1 \dots w_n\lhd \vdash_M^{c^*} q_0 \rhd \gamma' w_{n-k} \dots w_n\lhd \vdash_M^{c^*} q_0 \rhd \gamma w_{n-k+1} \dots w_n\lhd,$$

where $f(\gamma') = n_1 \dots n_l A$ and $f(\gamma) = n_1 \dots n_l W$ for a production $A \to w_k W$. This leads to the derivation $S \to^* w_1 \dots w_{k-1}\beta' \to w_1 \dots w_k\beta$, where $\beta'^R = f(\gamma') = n_1 \dots n_l A$. We conclude that $\beta^R = n_1 \dots n_l W$, which proves our claim.

For each $w \in L(M)$, we have

$$q_0 \rhd w \lhd = q_0 \rhd w_1 \dots w_n \lhd \vdash_M^{c^*} q_0 \rhd [\lambda]^n \lhd \vdash_M^c \mathsf{Accept}.$$

Therefore, there exists a derivation $S \to^* w_1 \dots w_n = w$, which shows that $w \in L(G)$. $\Box$

If the language L contains the empty word, we apply the above construction to the language $L \smallsetminus \{\lambda\}$ and then add the transition $\delta(q_0, \rhd\lhd) = \mathsf{Accept}$. $\Box$

Since $L_{\text{copy}} = \{ww \mid w \in \{a,b\}^*\}$ is contained in $\mathcal{L}(\mathsf{ORRWW})$, but it is neither contained in $\mathcal{L}(\mathsf{ORWW})$ nor in CFL, we get the following corollary.

Corollary 5.3.3. $\mathcal{L}(\mathsf{ORWW}) \cup \mathsf{CFL} \subsetneq \mathcal{L}(\mathsf{ORRWW})$.

In Section 4.2 it is shown that $\mathcal{L}(\mathsf{ORWW})$ is an abstract family of languages that is closed under intersection. The same proof can be used to show the following.

Theorem 5.3.4. $\mathcal{L}(\mathsf{ORRWW})$ *is closed under union, intersection, product, Kleene star, inverse morphisms, and non-erasing morphisms.*

In contrast to the situation for ORWW-automata, we know that the language class described by the ORRWW-automata is not closed under deleting morphisms. Since for every Turing machine the language of valid computations is included in $\mathcal{L}(\mathsf{ORRWW})$, all recursively enumerable languages would also be contained if we had closure under deleting morphisms.

Additionally, in contrast to the situation for ORWW-automata, the class $\mathcal{L}(\mathsf{ORRWW})$ is closed under the operation of reversal.

Proposition 5.3.5. *For each ORRWW-automaton M, there exists an ORRWW-automaton M' such that $L(M') = L(M)^R$.*

Proof. Let M be an ORRWW-automaton. As for general RRWW-automata, the behavior of M can be described by finitely many rewriting meta instructions of the form $(E_1, abc \to adc, E_2)$ and accepting meta-instructions of the form (E, Accept), where E, E_1, E_2 are regular expressions (see Definition 2.3.3). By replacing the former by $(E_2^R, cba \to cda, E_1^R)$ and the latter by (E^R, Accept), we obtain an ORRWW-automaton M' for the language $L(M)^R$. $\Box$

Finally, the following result shows that ORRWW-automata can even accept some unary languages that are not context-free.

Proposition 5.3.6. *The unary language $L = \{ a^n \mid \exists p, q > 1 : n = p \cdot q \}$ is accepted by an ORRWW-automaton.*

Proof. The languages $L_1 = \{ a^n b^n \mid n > 1 \}$, $L_2 = \{ b^n a^n \mid n > 1 \}$, $L_3 = \{ a^n \mid n \geq 1 \}$, and $L_4 = \{ b^n \mid n \geq 1 \}$ are all context-free and, therefore, they are accepted by ORRWW-automata. Now let $\varphi : \{a, b\}^* \to \{a\}^*$ denote the morphism that is defined by $\varphi(a) = \varphi(b) = a$. It is easily seen that

$$L = \varphi\left(\left(L_1^+ \cup L_1^+ \cdot L_3\right) \cap \left(L_3 \cdot L_2^* \cup L_3 \cdot L_2^* \cdot L_4\right)\right).$$

Hence, by Theorem 5.3.4, L is accepted by an ORRWW-automaton. □

Proposition 5.3.7. *The unary language $L_{expo} = \{ a^k \mid \exists i \in \mathbb{N} : k = 2^i \}$ is accepted by an ORRWW-automaton.*

Proof. The automaton works as follows. In the first phase, it marks every second letter for deletion, and in the second phase all those letters are marked as deleted. By using this method we ensure that we really halve the number of letters each time. The corresponding transition function can be realized with the following modified meta instructions.

$$(\rhd a c^* \lhd, \mathsf{Accept})$$
$$(\rhd a(c^* b c^* a)^* c^*, a \to b, (a|c)^* \lhd)$$
$$(\rhd (a|c)^*, b \to c, a(c^* b c^* a)^* c^* \lhd)$$

The automaton M accepts the word a^8 with the following rewrites: In the first cycle the word *aaaaaaaa* is rewritten into *abaaaaaa* which becomes *abababab* after another three cycles which end the first phase of marking every second letter for deletion. In the second phase each letter b is replaced by the letter c from left to right. It is important that we have alternating letters a and b right to the rewrite position. This leads to the word *acacacac*. In the next phase we get *acbcacbc* which changes into *acccaccc* which becomes *acccbccc* and finally *accccccc*. This word is finally accepted. □

Although the automaton is nondeterministic there is only one unique computation. Even more, there is only one valid choice of a rewrite position for each step. Therefore, we have a unique sequence of tape contents.

This allows us the construction of an ORRWW-automaton for the complementary language.

Proposition 5.3.8. *The unary language* $\overline{L_{expo}} = \{\, a^k \mid \nexists i \in \mathbb{N} : k = 2^i \,\}$ *is accepted by an ORRWW-automaton.*

Proof. As our ORRWW-automaton has a unique sequence of tape contents for each word, we can interchange Accept and Reject conditions in order to construct the automaton for the complement. We obviously reject if we get an odd number of letters a after a phase of halving the number of letters.

$$\begin{gathered}(\rhd a(c^*ac^*a)^+\lhd, \mathsf{Accept})\\ (\rhd a(c^*bc^*a)^*c^*, a \to b, (a|c)^*\lhd)\\ (\rhd(a|c)^*, b \to c, a(c^*bc^*a)^*c^*\lhd)\end{gathered}$$

□

We conjecture that we can only construct a complementary ORRWW-automaton if there is an ORRWW automaton with an unique sequence of tape contents.

We complete this section by shortly looking at some decision problems for ORRWW-automata. It has been shown in Section 4.3 that the emptiness and finiteness problems are decidable for ORWW-automata. However, as each context-free language is accepted by an ORRWW-automaton, and as the language class $\mathcal{L}(\mathsf{ORRWW})$ is closed under intersection, we obtain the following undecidability result from the undecidability of the intersection-emptiness problem for context-free languages.

Corollary 5.3.9. *The emptiness problem for ORRWW-automata is undecidable.*

From an ORRWW-automaton M, we can construct an ORRWW-automaton M' for the language $L(M) \cdot \Sigma^+$, as the proof of Theorem 5.3.4 is actually constructive. Now, $L(M')$ is finite, iff $L(M)$ is empty. Thus, from the undecidability of the emptiness problem, we immediately get the following.

Corollary 5.3.10. *The finiteness problem for ORRWW-automata is undecidable.*

Finally, from the corresponding results for context-free languages, it follows that also for ORRWW-automata, universality, regularity, inclusion and equivalence are all undecidable as every context-free languages can be represented by an ORRWW-automaton.

5.4 Concluding Remarks about ORRWW-automata

ORRWW automata clearly have more expressive power than ORWW-automata and we can represent many languages with them. The downside of the greater power is the undecidability of all decision problems of interest.

The ORRWW-automaton can recognize all context free languages. Like the ORWW-automaton it can recognize some languages that are not even in GCSL, but also some infinite languages that do not contain an infinite semi-linear subset like the unary language $\{a^{2^n} \mid n \geq 0\}$.

It is still an open problem whether the language class is a proper subset of the context-sensitive languages.

A next obvious point of research would be to look at other extensions of the context-free languages and see if they can be represented by our automata. Positive range concatenation (see, e.g. [19]) and indexed languages (see, e.g. [1]) would be suitable candidates.

Chapter 6

Conclusion and Closing Remarks

We have seen that ordered restarting automata are interesting for their own sake. Only the most general variant, that is the nondeterministic automaton with states, can express non-regular languages. But that does not mean the restricted variants are not interesting. Especially the simplest variant, the deterministic stateless ORWW-automaton, can be used to present regular languages concisely and in a simple way, which can also be used in teaching as a supplement to the DFA. As a size measurement we simply use the number of symbols instead of states. With this measurement in mind some language operations can be realized at least as efficient as with DFAs and some languages and language operations can be realized exponentially more succinctly. The general ORWW- and ORRWW-automata are interesting in a different way, although the former are probably the more interesting ones. The ORRWW automata are very powerful. They can accept all context-free languages and many more interesting languages, but all decision problems of interest are undecidable for them. Unfortunately, we still do not know whether they are properly included in the class of context sensitive languages which would be an interesting point to investigate further.

Nevertheless, it seems to be of greater interest to focus on the ORWW-automata. Especially since even a small extension of the ORWW-automata makes emptiness and finiteness undecidable, as we can see from the ORRWW-automaton, it is interesting to investigate this borderline case in more detail. We thus have a language class that properly contains the regular languages and forms an abstract family of languages. In addition, emptiness and finiteness can be decided, but we cannot represent the linear language $\{a^n b^n \mid n \in \mathbb{N}\}$ although we can represent even some languages that are not growing context sensitive.

Finally, here is a small inclusion diagram of the different language classes we have studied. You can see that it is still unknown whether $\mathcal{L}(\mathsf{ORRWW})$ is properly included in CSL. Furthermore we do not know whether the language classes GCSL or even CRL are included in $\mathcal{L}(\mathsf{ORRWW})$.

$$\begin{tikzcd}
 & \mathsf{CSL} & & \\
 & \mathsf{GCSL} \arrow[u] \arrow[r, dotted, "?"] & \mathsf{ORRWW} \arrow[ul, "?"'] & \\
\mathsf{CRL} \arrow[ur] \arrow[urr, dotted, "?"'] & \mathsf{CFL} \arrow[u] \arrow[ur] & \mathsf{ORWW} \arrow[u] & \\
\mathsf{DCFL} \arrow[u] \arrow[ur] & \mathsf{LIN} \arrow[u] & & \\
 & \mathsf{DLIN} \arrow[ul] \arrow[u] & \mathsf{det\text{-}ORWW} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & \mathsf{det\text{-}ORRWW} \arrow[d, leftrightarrow] \\
\mathsf{ORWW} \cap 2^{\{a\}^*} \arrow[r, hook] & \mathsf{REG} \arrow[u] \arrow[uuur] \arrow[r, leftrightarrow] & \mathsf{stl\text{-}ORWW} \arrow[r, leftrightarrow] & \mathsf{stl\text{-}ORRWW}
\end{tikzcd}$$

Figure 6.1: The Hierarchy of Language Class Inclusions. Solid lines indicate proper inclusions, dotted lines indicate inclusions. Question marks indicate unknown inclusions.

Some of the many open problems are the following. First of all there are still some open problems regarding the descriptional complexity of some language operations for languages given by stl-det-ORWW-automaton. Are there any efficient constructions for product and Kleene Star? What is the complexity of these operations? The current constructions involve converting the stl-det-ORWW-automaton into a NFA with an exponential blowup, using the constructions for the NFA, then using the powerset construction to get a DFA. This DFA is then converted into a stl-det-ORWW-automaton. Obviously, this construction leads to a huge blowup in size.

As our restricted variants all describe regular languages, we have asked how they can be converted into each other efficiently and how succinct their representations are. We still do not know how we can efficiently simulate a stl-

ORWW-automaton by a stl-det-ORWW-automaton, a stl-ORRWW-automaton by a stl-ORWW-automaton, or a stl-det-ORRWW-automaton by a stl-det-ORWW-automaton. How succinctly can we represent regular languages by stl-ORRWW-automata? Can we use the additional MVR steps for more succinctness?

Finally, there are still some open problems regarding ORWW-automata. Is universality or containment of a regular set decidable? Lastly, we have still the conjecture that the language class $\mathcal{L}(\mathsf{ORWW})$ only consists of semi-linear languages.

Bibliography

[1] Alfred V. Aho. Indexed grammars — an extension of context-free grammars. *J. ACM*, 15(4):647–671, October 1968.

[2] J.M. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 111–174. Springer, Berlin, Heidelberg, 1997.

[3] C.H. Bennett. Logical reversibiliy of computation. *IBM J. Res. Dev.*, 17:525–532, 1973.

[4] J.-C. Birget. Intersection and union of regular languages and state complexity. *Inform. Proc. Letters*, 43:185–190, 1992.

[5] J.-C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Math. Systems Theory*, 26:237–269, 1993.

[6] H. Bordihn, M. Holzer, and M. Kutrib. Determination of finite automata accepting subregular languages. *Theor. Comp. Sci.*, 410:3209–3222, 2009.

[7] J. Brzozowski. In search of the most complex regular languages. In N. Moreira and R. Reis, editors, *CIAA 2012, Proc.*, *LNCS 7381*, pages 5–24. Springer, Heidelberg, 2012.

[8] G. Buntrock and F. Otto. Growing context-sensitive languages and Church-Rosser languages. *Inform. Comp.*, 141:1–36, 1998.

[9] P. Caron. Families of locally testable languages. *Theor. Comp. Sci.*, 242:361–376, 2000.

[10] J. Dassow. Subregular restrictions for some language generating devices. In R. Freund, M. Holzer, B. Truthe, and U. Ultes-Nitsche, editors, *Fourth Workshop on Non-Classical Models for Automata and Applications (NCMA 2012), Proc.*, books@ocg.at, Band 290, pages 11–26. Oesterreichische Computer Gesellschaft, Wien, 2012.

[11] Y. Gao, N. Moreira, R. Reis, and Sh. Yu. A Survey on Operational State Complexity. arXiv:1509.03254v1, Sept. 10, 2015.

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[13] S. Ginsburg and E. H. Spanier. Finite-turn pushdown automata. *SIAM Journal on Control*, 4(3):429–453, 1966.

[14] I. Glaister and J. Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59(2):75 – 77, 1996.

[15] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.

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

[17] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In H. Reichel, editor, *FCT'95, Proc.*, *LNCS 965*, pages 283–292. Springer, Heidelberg, 1995.

[18] N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975.

[19] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars.* Springer Berlin Heidelberg, 2010.

[20] P. Karandikar and Ph. Schnoebelen. Generalized Post embedding problems. *Theory Comput. Syst.*, 56:697–716, 2015.

[21] M. Kutrib and A. Malcher. Reversible pushdown automata. *J. Comput. System Sci.*, 78:1814–1827, 2012.

[22] M. Kutrib, A. Malcher, and M. Wendlandt. Reversible queue automata. In S. Bensch, R. Freund, and F. Otto, editors, *Sixth workshop on non-classical models of automata and applications (NCMA 2014), Proc.*, books@ocg.at, Band 304, pages 163–178. Oesterreichische Computer Gesellschaft, Wien, 2014.

[23] M. Kutrib, H. Messerschmidt, and F. Otto. On stateless deterministic restarting automata. *Acta Inform.*, 47:391–412, 2010.

[24] K. Kwee and F. Otto. On some decision problems for stateless deterministic ordered restarting automata. In J. Shallit and A. Okhotin, editors, *DCFS 2015, Proc.*, *LNCS 9118*, pages 165–176. Springer, Heidelberg, 2015.

[25] K. Kwee and F. Otto. On Ordered RRWW-Automata. In Srečko Brlek and Christophe Reutenauer, editors, *Developments in Language Theory, LNCS 9840*, pages 268–279, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[26] K. Kwee and F. Otto. On the effects of nondeterminism on ordered restarting automata. In R.M. et al. Freivalds, editor, *SOFSEM 2016, Proc.*, *LNCS 9587*, pages 369–380. Springer, Heidelberg, 2016.

[27] K. Kwee and F. Otto. A Pumping Lemma for Ordered Restarting Automata. In G. Pighizzini and C. Câmpeanu, editors, *DCFS 2017, Proc.*, *LNCS 10316*, pages 226 – 237. Springer, Heidelberg, 2017.

[28] C. Lautemann. One pushdown and a small tape. In K.W. Wagner, editor, *Dirk Siefkes zum 50. Geburtstag*, pages 42–47. Technische Universität Berlin and Universität Augsburg, 1988.

[29] R. McNaughton. Algebraic decision procedures for local testability. *Math. Systems Theory*, 8:60–76, 1974.

[30] Hartmut Messerschmidt. CD-Systems of Restarting Automata. *Doctoral dissertation, Fachbereich Elektrotechnik/Informatik, Universität Kassel*, 2008.

[31] B.G. Mirkin. On dual automata. *Cybernetics*, 2:6–9, 1966.

[32] F. Mráz, M. Plátek, and Procházka. M. On special forms of restarting automata. *Grammars*, 2:223–233, 1999.

[33] F. Otto. Restarting Automata and Their Relations to the Chomsky Hierarchy. In Zoltán Ésik and Zoltán Fülöp, editors, *Developments in Language Theory*, *LNCS 2710*, pages 55–74, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[34] F. Otto. On the descriptional complexity of deterministic ordered restarting automata. In H. Jürgensen, J. Karhumäki, and A. Okhotin, editors, *DCFS 2014, Proc.*, *LNCS 8614*, pages 318–329. Springer, Heidelberg, 2014.

[35] F. Otto and K. Kwee. Deterministic ordered restarting automata that compute functions. In I. Potapov, editor, *DLT 2015, Proc.*, *LNCS 9168*, pages 401–412. Springer, Heidelberg, 2015.

[36] F. Otto and K. Kwee. On the descriptional complexity of stateless deterministic ordered restarting automata. *Information and Computation*, `https://doi.org/10.1016/j.ic.2017.09.006`, 2017.

[37] F. Otto and F. Mráz. Deterministic ordered restarting automata for picture languages. *Acta Inf.*, 52(7-8):593–623, November 2015.

[38] F. Otto, M. Wendlandt, and K. Kwee. Reversible ordered restarting automata. In J. Krevine and J.-B.. Stefani, editors, *RC 2015, Proc.*, *LNCS 9138*, pages 60–75. Springer, Heidelberg, 2015.

[39] J.-E. Pin. On reversible automata. In Simon. I., editor, *LATIN'92, Proc.*, *LNCS 583*, pages 401–416. Springer, Heidelberg, 1992.

[40] J.-E. Pin. Syntactic semigroups. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 1*, pages 679–746. Springer, Heidelberg, 1997.

[41] J.-E. Pin. Mathematical foundations of automata theory, `http://www.liafa.jussieu.fr/~jep/PDF/MPRI/MPRI.pdf`, November 2016.

[42] J.E. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[43] S. Schmitz and Ph. Schnoebelen. Multiply-recursive upper bounds with Higman's lemma. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP 2011, Proc., Part II*, *LNCS 6756*, pages 441–452. Springer, Heidelberg, 2011.

[44] S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comp. Sci.*, 125:315–328, 1994.

[45] Y. Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6:151–167, 1972.