Weighted Restarting Automata

Qichao Wang

kassel university press

Qichao Wang

Weighted Restarting Automata

kassel
university
press

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. Heiko Vogler
Defense day:	13th September 2017

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

Zugl.: Kassel, Univ., Diss. 2017
ISBN 978-3-7376-0446-8 (print)
ISBN 978-3-7376-0447-5 (e-book)
DOI: http://dx.medra.org/10.19211/KUP9783737604475
URN: http://nbn-resolving.de/urn:nbn:de:0002-404474

Zusammenfassung

Restart-Automaten sind ein formales Werkzeug zur Modellierung der linguistischen Technik von Analyse durch Reduktion, die für die Analyse von natürlichen Sprachen verwendet wird. Ein Restart-Automat M is ein Sprachakzeptanzgerät, das mit einem angegebenen Eingabewort w über einem Eingabealphabet Σ entweder akzeptiert oder verwirft. Aber im Fall von Akzeptanz kann man sich für die Anzahl der akzeptierenden Rechnungen für die Eingabe w interessieren, oder für die minimale Anzahl der Schritte (oder Zyklen) in einer solchen akzeptierenden Rechnung. Um diese quantitativen Fragen zu beantworten, führen wir das Konzept der *gewichteten Restart-Automaten* ein. Ein solcher Automat ist als ein Paar (M, ω) definiert, wobei M ein Restart-Automat vom Typ X mit einem Eingabealphabet Σ ist, und ω eine Gewichtsfunktion von den Transitionen von M in einen Semiring S ist. Auf diese Weise kann jeder gewichtete Restart-Automat (M, ω) eine Funktion von Σ^* zu S repräsentieren. Wir zeigen einige syntaktische und semantische Eigenschaften dieser Funktionen wie obere Schranken (siehe [BFGM05]), Wachstumsraten und Abschlusseigenschaften. In dieser Arbeit erweitern wir auch gewichtete Restart-Automaten zu Transducern. Wir beweisen, dass für die gewichteten monotonen Restart-Automaten mit Hilfssymbolen, die sogenannten RRWW-Automaten, die nach einem Rewrite-Schritt noch weiter die Eingabe lesen können, strikt ausdrucksstärker sind als die sogenannten RWW-Automaten, die nach einem Rewrite-Schritt sofort restarten müssen, was sowohl im deterministischen als auch nichtdeterministischen Fall gilt. Es wird das erste Mal gezeigt, dass eine Version der monotonen RRWW-Automaten strikt ausdrucksstärker ist als die entsprechende Version der monotonen RWW-Automaten. Schließlich erweitern wir die gewichteten Restart-Automaten zu Sprachakzeptoren, indem eine Akzeptanzbedingung für das Gewicht der akzeptierenden Rechnungen hinzugefügt wird. Wir zeigen, dass mit einer solchen zusätzlichen Akzeptanzbedingung gewichtete Restart-Automaten von einem gewissen Typ den sogenannten *Nicht-Vergessenden Restart-Automaten* entsprechen. Des weiteren ist eine andere Klasse der von gewichteten Restart-Automaten akzeptierten Sprachen unter der Operation des Schnitts abgeschlossen. Es ist das erste Ergebnis, das zeigt, dass eine Klasse der Sprachen, die durch eine generelle Klasse der Restart-Automaten definiert ist, unter

dieser Operation abgeschlossen ist.

Abstract

Restarting automata have been introduced as a formal tool to model the analysis by reduction, which is a technique used in linguistics to analyse sentences of natural languages. A restarting automaton M is a language accepting device: Given an input word w over some input alphabet Σ, it either accepts or rejects. But in case of acceptance, one may be interested in the number of accepting computations of M on input w, or one may be interested in the least number of steps (or cycles) in such an accepting computation. For answering such quantitative questions, we introduce the concept of a *weighted restarting automaton*. Such an automaton is defined as a pair (M, ω), where M is a restarting automaton of type X on some input alphabet Σ, and ω is a weight function from the transitions of M into a semiring S. In this way, a weighted restarting automaton (M, ω) can represent a function f_ω^M from Σ^* into S. We show some syntactic and semantic properties of these functions such as upper bounds (see [BFGM05]), growth rates and closure properties. In this work, we also extend weighted restarting automata to transducers. We prove that for weighted monotone restarting automata with auxiliary symbols, the variant that may keep on reading after performing a rewrite step (the so-called RRWW-automaton) is strictly more expressive than the variant that must restart immediately after performing a rewrite step (the so-called RWW-automaton), which again holds in the deterministic as well as in the nondeterministic case. This is the first time that a version of the monotone RRWW-automaton is shown to differ in expressive power from the corresponding version of the monotone RWW-automaton. Finally, we extend weighted restarting automata to language acceptors by adding an acceptance condition on the weight of its accepting computations. We will see that using such a relative acceptance, weighted restarting automata of a certain type coincide with the so-called *non-forgetting restarting automata*. In addition, another class of languages accepted by weighted restarting automata is shown to be closed under the operation of intersection. This is the first result that shows that a class of languages defined in terms of a quite general class of restarting automata is closed under the operation of intersection.

Publications

The following refereed works have already been published, and we take them as parts of this thesis.

[WHO15] Qichao Wang, Norbert Hundeshagen, and Friedrich Otto. Weighted restarting automata and pushdown relations. In Andreas Maletti, editor, *Proceedings of the 6th International Conference on Algebraic Informatics*, volume 9270 of *Lecture Notes in Computer Science*, pages 196-207. Springer, Heidelberg, 2015.

[OW16] Friedrich Otto and Qichao Wang. Weighted Restarting Automata. *Soft Computing*, 2016. DOI: 10.1007/s00500-016-2164-4. The results of this paper have been announced at WATA 2014 in Leipzig.

[WO16a] Qichao Wang and Friedrich Otto. Weighted restarting automata and pushdown relations. *Theoretical Computer Science*, 635:1-15, 2016.

[WO16b] Qichao Wang and Friedrich Otto. Weighted restarting automata as language acceptors. In Yo-Sub Han and Kai Salomaa, editors, *Proceedings of the 21st International Conference on Implementation and Application of Automata*, volume 9705 of *Lecture Notes in Computer Science*, pages 298-309. Springer, Switzerland, 2016.

[Wan17] Qichao Wang. On the expressive power of weighted restarting automata. In Rudolf Freund, Frantisek Mráz, and Daniel Prusa, editors, *Proceedings of the Ninth Workshop on Non-Classical Models of Automata and Applications*, volume 329 of books@ocg.at, pages 227-241, Österreichische Computer Gesellschaft, 2017.

The main parts of Section 2.2.2 and 5.2 are based on the joined works with Prof. Dr. Friedrich Otto and Dr. Norbert Hundeshagen. Further, the techniques used to prove most of results in Section 4.3 and 6.2 are due to Prof. Dr. Friedrich Otto. In addition, the examples presented in Section 3.3 and 4.1 are also from Prof. Dr. Friedrich Otto.

Acknowledgements

First, I am deeply grateful to my supervisor Prof. Dr. Friedrich Otto for his continuous support and guidance in writing this thesis and for providing me with great freedom. Moreover, he introduced me into the fascinating field of weighted restarting automata and supplied me with new interesting and challenging problems and ideas. In addition, he often took time to discuss the ongoing progress with me, and his hints and corrections substantially improved my work.

Secondly, I also wish to thank my second supervisor Prof. Dr. Heiko Vogler for his carefully reading and correcting my thesis. His comments and suggestions were always helpful to improve the content of this thesis.

Furthermore, I am also truly indebted to my current and former colleagues for their various hints. In particular, I would express my profound thanks to Dr. Norbert Hundeshagen for his helpful suggestions and experiences made during writing a thesis. Most important was that he often took time to discuss recent problems on the present topic and related topics such as transducers and relations with me, and he always provided me with valuable suggestions. In addition, he also participated as a co-author in the publication [WHO15].

In some sense many people should be named here, and this thesis would not have been possible without their help and support. Finally, I would like to take this opportunity to thank everyone who helped and supported me while doing the present work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Restarting automata have been introduced as a formal model for the linguistic technique of analysis by reduction, which can be used to check the correctness of natural language sentence [JMPV95]. In order to study quantitative aspects of restarting automata, we introduce the concept of a *weighted restarting automaton*. In this thesis weighted restarting automata of various types and their computational power are studied. The main goal of this chapter is to introduce the motivation, related works, and main results of the present work. In addition, we also describe the structure of this thesis.

1.1 Motivation

Analysis by reduction is a linguistic technique used to verify the syntactic correctness of sentences of a natural language [Str00]. This analysis consists of sequences of local simplifications of a sentence, and during this process each step preserves the correctness or incorrectness of the sentence. After a finite number of steps, either a correct simple sentence is obtained, or an error is detected. In the former case the given sentence is accepted as being syntactically correct; if all possible sequences of simplifications yield errors, then the given sentence is not syntactically correct. In addition, by this process the structure of the given sentence can be analysed and information on dependencies and independencies between certain parts of the sentence can be derived. To illustrate the principles of this technique, we present the following example:

She says that the seashells she sells are on the sunshine seashore.

In order to analyse this sentence, we read it from left to right until a phrase is discovered that can be simplified. For example, we can delete the word *that* or the word *sunshine*, and then the following sentences are obtained:

(1) She says the seashells she sells are on the sunshine seashore.
(2) She says that the seashells she sells are on the seashore.

Obviously both these simplifications are correct, and so we can conclude that the phrases *that* and *sunshine* are *independent* of each other. Hence, from (1) and (2) the following sentence can be derived in the next step:

(3) She says the seashells she sells are on the seashore.

Further, it is easily seen that we cannot remove the phrase *the seashells*, as the phrase *She says* depends on the phrase *the seashells*, and such a simplification would lead to a syntactic error. In order to simplify (3) we delete the phrase *She says* or the phrase *she sells*, thus obtaining the following sentences:

(4) The seashells she sells are on the seashore.
(5) She says the seashells are on the seashore.

It is easy to observe that both these sentences derived from (3) through the simplifications are syntactically correct. It follows that the phrase *She says* and the phrase *she sells* are independent of each other. Hence, by deleting these phrases we obtain the following simple sentence:

(6) The seashells are on the seashore.

The above simple sentence is easily verified as being syntactically correct, and the analysis of the given sentence is completed with success. Thus, we can conclude that the original sentence is syntactically correct. In addition, we have seen that by this analysis some information on dependencies and independencies between certain parts of the sentence is obtained (see, e.g., [Tes59, Niv09]).

The *restarting automaton* was introduced as a formal model of analysis by reduction [JMPV95, LPS07]. As shown in Figure 1.1, such an automaton consists of a finite-state control and a flexible tape with end markers, on which a read/write window of a fixed positive size operates. Based on the state and the window content, the automaton may perform a *move-right step*, which shifts the window one position to the right and changes the state. It may also execute a *rewrite step*, which replaces the content of the window by a word that is strictly shorter, places the window immediately to the right of the newly written word, and changes the state. Finally, it may perform a *restart step*, which moves the window back to the left end of the tape and resets the automaton to its initial state, or it may make an *accept step*. Observe that a rewrite step shortens the content of the tape, and it is assumed that the length of the tape is shortened accordingly. In addition, it is required

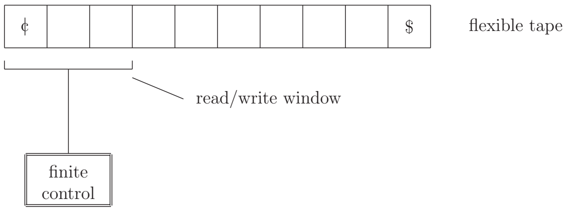

Figure 1.1: Schematic representation of a restarting automaton.

that before a restart operation can be executed, exactly one rewrite must have taken place, that is, the automaton can be seen as working in cycles, where each cycle begins with the window at the left end of the tape and the finite-state control being in the initial state, then some move-right steps are executed, then a single rewrite step is performed, then again some move-right steps may be executed, and finally the cycle is completed by a restart step. Thus, a computation consists of a finite sequence of cycles that is followed by a tail computation, which consists of a number of move-right steps, possibly a single rewrite step, and which is completed by an accept step or ends by reaching a configuration for which no further step is defined. In the latter case we say that the current computation halts without acceptance.

Many different types and variants of restarting automata have been introduced and studied since 1995 (see, e.g., [JMPV97, JMPV98, JMPV99]). In particular, many well-known classes of formal languages, like the regular languages REG, the deterministic context-free languages DCFL, the context-free languages CFL, the Church-Rosser languages CRL, and the growing context-sensitive languages GCSL have been characterized by various types of restarting automata. An overview on restarting automata is given by [Ott06].

Just as finite automata, also restarting automata accept or reject their inputs. Therefore, such an automaton can be seen as computing a Boolean function. *Weighted automata* were introduced by [Sb61]. In these automata each transition gets a quantitative value from some semiring S as a weight. These weights can model the cost involved when executing a transition such as the needed resources or time, or the probability or reliability of its successful execution. By forming the product of all weights along a computation, a weight can be assigned to that computation, and by forming the sum of all weights of all accepting computations for a given input, an element of S is associated with that input. For example, by using appropriate weights, we can determine the number of ways that a word can be accepted by a finite automaton. Weighted automata and their properties are described in detail in the recent handbook by [DKV09].

After their introduction, weighted automata have been applied in many areas like natural-language processing, speech recognition, optimization of energy consumption, and probabilistic systems (see, e.g., [CDH09, MPR02, MPR00]). Also many applications of them can be found in digital image compression and model checking (see, e.g., [IK93, Bou06]). Due to these applications, many different variants of weighted automata have been invented and studied (see, e.g., [DK17, DHV15, DK13, DM11, ICJ14]). Following this development, we introduce *weighted restarting automata* in order to study quantitative aspects of computations of restarting automata.

A weighted restarting automaton $\mathcal{M}$ is given by a pair (M, ω), where M is a restarting automaton on some input alphabet Σ, and ω is a *weight function* that assigns a weight from some semiring S to each transition of M. As outlined above, $\mathcal{M}$ defines a value $f_\omega^M(w)$ from S for each input word $w \in \Sigma^*$. Thus, $\mathcal{M}$ defines a function $f_\omega^M : \Sigma^* \to S$. By looking at different semirings S and different weight functions ω, various quantitative aspects of the behavior of M can be expressed through these functions. For example, by taking S to be the semiring of natural numbers with addition and multiplication, we can count the number of accepting computations for each input, or by using the tropical semiring, we can determine the minimal number of cycles in an accepting computation for each input.

1.2 Outline

Now we describe the structure of this thesis and the main results of the present work in short. We begin with some basic notions and definitions on formal languages and relations in Chapter 2. Then, we introduce the concept of the *weighted restarting automaton* that is the central notion of this work in Chapter 3.

As mentioned above, each weighted restarting automaton can represent a function $f : \Sigma^* \to S$. We are interested in the syntactic and semantic properties of these functions. In Chapter 4 we study their closure properties under various operations. If the semiring S is linearly ordered, then we can abstract this function to a function from $\mathbb{N}$ into S by taking

$$\hat{f}_\omega^M(n) = \max\{\, f_\omega^M(w) \mid w \in \Sigma^*,\ |w| = n \,\},$$

where $|w|$ denotes the length of the word w. Growth rates of and upper bounds for such functions will also be investigated in Chapter 4.

Further, if S is the semiring of formal languages over a finite alphabet Δ, then f_ω^M is a transformation from Σ^* into the languages over Δ. In fact, it is easily seen that the transformations computed by the restarting transducers introduced by [HO12] occur as a special case. In this work we extend weighted restarting automata to transducers. In Chapter 5 we will study the classes of relations that are computed by weighted restarting automata and restarting transducers, compare some of these classes to each other, and relate them to the class of *pushdown relations* and some of its subclasses. We will see that for weighted monotone restarting automata with auxiliary symbols, the variant that may keep on reading after performing a rewrite step (the so-called RRWW-automaton) is strictly more expressive than the variant that must restart immediately after performing a rewrite step (the so-called RWW-automaton), which holds in the deterministic as well as in the nondeterministic case. This is the first time that a version of the monotone RRWW-automaton is shown to differ in expressive power from the corresponding version of the monotone RWW-automaton.

Next, we extend weighted restarting automata to language acceptors by placing a relative acceptance condition on the value of the function $f_\omega^M(w)$ for a weighted restarting automaton $\mathcal{M} = (M, \omega)$ and an input $w \in \Sigma^*$. The purpose of Chapter 6 is to study the classes of languages that are accepted by weighted restarting automata relative to subsets of various semirings, to show their closure properties and membership problems, and to compare them to each other. In particular, a certain class of languages accepted by weighted restarting automata is shown to be closed under the operation of intersection. This is the first result that shows that a class of languages defined in terms of a quite general class of restarting automata is closed under the operation of intersection.

Finally, this thesis closes with a summary and some problems for future work, which are given in Chapter 7.

Chapter 2

Formal Languages and Relations

It is the goal of this chapter to present basic foundations for formal languages and relations that we will use in the forthcoming chapters of this thesis. It consists of two sections, where the first one restates some well-known classes of formal languages and corresponding automata. The second section introduces the classes of rational relations and pushdown relations. In particular, we define some restricted classes of pushdown relations and study the inclusion relations between them.

2.1 Formal Languages and Automata

We start with some basic definitions and notations of formal languages and mathematics. A finite set of symbols is called an *alphabet*, and in general we denote it by the capital Greek letter Σ. A finite string w of symbols from an alphabet Σ is called a *word* over the alphabet Σ. Throughout the paper we will use $|w|$ to denote the length of a word w, λ to denote the empty word, and $|X|$ to denote the number of elements in a set X. Further, $|w|_a$ denotes the number of occurrences of symbol a in the word w, and w^R denotes the reversal of w. By Σ^* we denote the set of all the words over the alphabet Σ, Σ^+ is used to denote the set of all the non-empty words over Σ, and for each $n \geq 0$, Σ^n is used to denote the set of all words of length n over Σ. In addition, let X be a set of words over some alphabet Σ, then $\mathbb{P}(X)$ denotes the power set of X, $\mathbb{P}_{\text{fin}}(X)$ denotes the set of all finite subsets of X, and $\overline{X}$ denotes the complement of X in Σ^*, that is, $\overline{X} = \Sigma^* \smallsetminus X$. A *(formal) language* is defined as a set of words over some alphabet Σ, that is, $L \subseteq \Sigma^*$ or $L \in \mathbb{P}(\Sigma^*)$. Finally, let $\mathbb{N}$ denote the set of natural numbers, and let $\mathbb{N}_+$ denote the set of positive natural numbers.

Regular Languages and Finite State Automata

Here we consider some well-known classes of formal languages. First, we introduce the class of regular languages (REG for short), which can be characterized by *finite state automata*.

Definition 2.1.1 ([AU72]). *A nondeterministic finite state automaton (*NFA *for short) is defined by a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to \mathbb{P}_{\text{fin}}(Q)$ [1] is a finite transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states.*

A configuration of an NFA A is described by a pair (q, w_i), where $q \in Q$ is the current state and $w_i \in \Sigma^*$ is the unread part of the input word. Further, a transition of A between configurations (q, w_i) and (q', w_{i+1}) is denoted by $(q, w_i) \vdash_A (q', w_{i+1})$, iff $q' \in \delta(q, a)$, where $a \in \Sigma$ and $w_i = aw_{i+1}$. This means that the configuration (q', w_{i+1}) can be reached from the configuration (q, w_i) by performing a single transition step. Let δ^* denote the *natural extention* of δ to words over Σ, that is, $\delta^*(q, \lambda) = \{q\}$, and $\delta^*(q, ua) = \bigcup\limits_{q' \in \delta^*(q,u)} \delta(q', a)$, where $q \in Q$, $u \in \Sigma^*$ and $a \in \Sigma$. By $\vdash_A^*$ we denote the reflexive transitive closure of this relation. After processing all the symbols of the input word, if A halts in a state $q \in F$, we say that A *accepts*. For a configuration (q, w_i), if $\delta(q, a) = \emptyset$, where $a \in \Sigma$ and $w_i = aw_{i+1}$, then A necessarily halts, and we say that A *rejects*. For an input word $w \in \Sigma^*$, a *computation* C of A is defined as a sequence of the transitions starting from the initial configuration (q_0, w) and ending in the halting configuration (q_f, w') for some $q_f \in Q$ and $w' \in \Sigma^*$, i.e., $(q_0, w) \vdash_A^* (q_f, w')$. If the computation C is completed with an accepting configuration, that is, $q_f \in F$ and $w' = \lambda$, then C is called an *accepting computation*; Otherwise, it is called a *non-accepting computation*. Obviously, for each input word $w \in \Sigma^*$, a computation of A on w is finite. The language accepted by A is defined as

$$L(A) = \{w \in \Sigma^* \mid \exists q \in F : (q_0, w) \vdash_A^* (q, \lambda)\}.$$

Finally, the automaton A is a *deterministic* finite state automaton (DFA for short), if based on a state $q \in Q$ and a symbol $a \in \Sigma$, there is at most one transition defined, i.e., $|\delta(q, a)| \leq 1$. Thus, each configuration of a DFA has at most one successor configuration, and a DFA has at most one accepting computation for any input. It is well-known that NFAs are equivalent to DFAs, and they characterize the class REG of regular languages (see, e.g., [HU79, Har78]). Now we present a simple example of a DFA.

[1]Note that there are some other variants of finite state automata such as the variant that can perform arbitrarily many λ-steps.

Example 2.1.1. *Let $A = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a\}, \delta, q_0, \{q_0, q_2, q_3, q_4\})$ be the* DFA*, where the transition function δ is defined as follows:*

$$\begin{array}{lrcllrcl} (1) & \delta(q_0, a) & = & \{q_1\}, & (4) & \delta(q_3, a) & = & \{q_4\}, \\ (2) & \delta(q_1, a) & = & \{q_2\}, & (5) & \delta(q_4, a) & = & \{q_5\}, \\ (3) & \delta(q_2, a) & = & \{q_3\}, & (6) & \delta(q_5, a) & = & \{q_0\}. \end{array}$$

It is easy to see that an input word w of length that is divisible by 2 *or* 3 *can be accepted by the* DFA *A, that is, $L(A) = \{a^n \mid n \equiv 0 \mod 2$ or $n \equiv 0 \mod 3\}$.*

Context-free Languages and Pushdown Automata

A finite state automaton can be extended to a *pushdown automaton* (PDA for short) by equipping it with a memory, the so-called *pushdown.* Each transition of a PDA is based on the current state and input symbol, and the top of the pushdown, optionally popping the top of the pushdown, and optionally pushing new symbols onto the pushdown.

Definition 2.1.2 ([AU72])**.** *A* PDA *is defined by a 7-tuple $P = (Q, \Sigma, X, q_0, Z_0, F, \delta)$, where Q is a finite set of states, Σ is an input alphabet, X is a pushdown alphabet, $q_0 \in Q$ is the initial state, $Z_0 \in X$ is the bottom marker of pushdown, $F \subseteq Q$ is the set of final states, and $\delta : Q \times (\Sigma \cup \{\lambda\}) \times X \to \mathbb{P}_{\text{fin}}(Q \times X^*)$ is a finite transition function.*

A configuration of a PDA P can be written as a 3-tuple (q, u, α), where $q \in Q$ is the current state, $u \in \Sigma^*$ is the still unread part of the input, and $\alpha \in X^*$ is the current content of the pushdown with the first letter of α at the bottom. The language accepted by P is defined as

$$L(P) = \{w \in \Sigma^* \mid \exists q \in F, \alpha \in X^* : (q_0, w, Z_0) \vdash_P^* (q, \lambda, \alpha)\}.$$

Further, the PDA P is a *deterministic* pushdown automaton (DPDA for short), if $|\delta(q, a, \alpha)| + |\delta(q, \lambda, \alpha)| \leq 1$ for each state $q \in Q, a \in \Sigma$, and $\alpha \in X \cup \{\lambda\}$. The classes of languages that are accepted by PDAs and DPDAs coincide with the classes of *context-free languages* (CFL for short) and *deterministic context-free languages* (DCFL for short), respectively. Now we give a simple example of a PDA.

Example 2.1.2. *Let $P_1 = (Q, \Sigma, X, q_0, Z_0, F, \delta)$ be the* PDA*, where $Q = \{q_0, q_1, q_2, q_3, q_4, q_e\}$ is a finite set of states, $\Sigma = \{a, b, c, d\}$ is an input alphabet, $X = \{Z_0, Z_1\}$ is a pushdown alphabet, $q_0 \in Q$ is the initial state, $Z_0 \in X$ is the bottom marker of pushdown, $F = \{q_e\}$ is the set of final states,* *and the transition function δ is defined as follows:*

$$\begin{array}{llcl@{\qquad}llcl}
(1) & \delta(q_0,a,Z_0) & = & \{(q_1,Z_0Z_1)\}, & (6) & \delta(q_3,b,Z_1) & = & \{(q_4,\lambda)\},\\
(2) & \delta(q_1,a,Z_1) & = & \{(q_1,Z_1Z_1)\}, & (7) & \delta(q_4,b,Z_1) & = & \{(q_3,Z_1)\},\\
(3) & \delta(q_1,b,Z_1) & = & \{(q_2,\lambda),(q_3,Z_1)\}, & (8) & \delta(q_4,d,Z_0) & = & \{(q_e,Z_0)\},\\
(4) & \delta(q_2,b,Z_1) & = & \{(q_2,\lambda)\}, & (9) & \delta(q_0,c,Z_0) & = & \{(q_e,Z_0)\},\\
(5) & \delta(q_2,c,Z_0) & = & \{(q_e,Z_0)\}, & (10) & \delta(q_0,d,Z_0) & = & \{(q_e,Z_0)\}.
\end{array}$$

For an input word w, when processing the prefix a^n, the PDA *P_1 pushes correspondingly many Z_1-symbols onto its pushdown. Then, it guesses whether $|w|_a = |w|_b$ or $2 \cdot |w|_a = |w|_b$. Finally, P_1 checks its guess by reading the information for the number of a-symbols from the pushdown. Thus, it is easily seen that the language accepted by P_1 is*

$$L(P_1) = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}.$$

It is important to notice that there exist some context-free languages that do not belong to DCFL, that is, DCFL $\subsetneq$ CFL. For example, the language $L(P_1)$ given in the example above cannot be accepted by any DPDA (see [Yu89]). Further, it is also clear that REG $\subseteq$ DCFL and this inclusion is proper. Obviously, the language

$$L_1 = \{a^n b^n \mid n \geq \mathbb{N}\}$$

is deterministic context-free, but not regular, since an NFA cannot remember the number of a-symbols and compare it to the number of b-symbols. Therefore, we have the following proper inclusion results.

Corollary 2.1.1. *REG $\subsetneq$ DCFL $\subsetneq$ CFL.*

Church-Rosser Languages, Growing Context-Sensitive Languages, and Two-Pushdown Automata

We continue by introducing a stronger variant of pushdown automata, the so-called *two-pushdown automata* [NO05]. A two-pushdown automaton (TPDA for short) with pushdown windows of size k is a nondeterministic automaton $T = (Q, \Sigma, \Gamma, \delta, k, q_0, Z_0, t_1, t_2, F)$, where Q is a finite of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma \setminus \Sigma$ is the bottom marker of the pushdown stores, $t_1, t_2 \in (\Gamma \setminus \Sigma)^*$ is the preassigned content of the first and second pushdown store, respectively, $F \subseteq Q$ is the set of final states, and δ is the transition function. To each triple (q, u, v), where $q \in Q$ is a state, $u \in \Gamma^k \cup \{Z_0\} \cdot \Gamma^{<k}$ is the content of the topmost part of the first pushdown, and $v \in \Gamma^k \cup \Gamma^{<k} \cdot \{Z_0\}$ is the content of the topmost part of the second pushdown, it associates a finite set of triples

from $Q \times \Gamma^* \times \Gamma^*$. The automaton T is a deterministic TPDA (DTPDA for short), if δ is a (partial) function. A configuration of a (D)TPDA is described by a triple (q, u, v), where $q \in Q$ is the current state, $u \in \Gamma^*$ is the content of the first pushdown with the first symbol of u at the bottom and the last symbol of u at the top, and the $v \in \Gamma^*$ is the content of the second pushdown with the last symbol of v at the bottom and the first symbol of v at the top. It is worth to mention that the input for a (D)TPDA is provided as a part of the initial content of the second pushdown. Therefore, in order to consume the input completely and accept, a (D)TPDA always needs to empty its pushdown stores, that is, the language accepted by a (D)TPDA T is defined as

$$L(P) = \{w \in \Sigma^* \mid \exists q \in F : (q_0, Z_0 t_1, w t_2 Z_0) \vdash_T^* (q, \lambda, \lambda)\}.$$

It is well-known that TPDAs have the same computational power as *Turing Machines*, and the class of *recursively enumerable languages* (RE for short) is characterized by the latter model [HMU01]. Now we introduce some restricted variants of TPDAs given in [BO98]. A (D)TPDA is called *shrinking* if there exists a weight function $\varphi : (Q \cup \Gamma)^* \to \mathbb{N}$ such that, for all transitions $(q', u', v') \in \delta(q, u, v)$, $\varphi(q'u'v') < \varphi(quv)$. A (D)TPDA is called *length-reducing* if $|u'v'| < |uv|$ holds for all transitions $(q', u', v') \in \delta(q, u, v)$. Obviously, the length-reducing TPDA is a special case of the shrinking TPDA. For these restricted variants of TPDAs, there are some characterizations given in [BO98, NO05, DW86, Nie03]. A language is a *Church-Rosser language* (CRL for short), if and only if it is accepted by a shrinking DTPDA, if and only if it is accepted by a length-reducing DTPDA. A language is a *growing context-sensitive language* (GCSL for short), if and only if it is accepted by a shrinking TPDA, if and only if it is accepted by a length-reducing TPDA. It is easy to see that DCFL $\subsetneq$ CRL $\subsetneq$ GCSL and CFL $\subsetneq$ GCSL. Further, we consider the languages

$$L_2 = \{\, ww \mid w \in \{a, b\}^* \,\}$$

and

$$L_3 = \{a^{2^n} \mid n \geq 0\}.$$

It is known that the language L_2 is not context-free, while $\overline{L_2} = \{a, b\}^* \smallsetminus L_2$ is a context-free language. On the other hand, $\overline{L_2}$ does not belong to the language class CRL, and it follows that $\overline{L_2} \in \mathsf{CFL} \setminus \mathsf{CRL}$. Together with the fact that $L_3 \in \mathsf{CRL} \setminus \mathsf{CFL}$, we can conclude that the language classes CFL and CRL are incomparable under inclusion (see, e.g., [BO98, MNO88, Nar84]). Of course, GCSL is a proper subset of the class of context-sensitive languages (CSL for short). Finally, the language class RE contains some languages that are not context-sensitive [HU79]. The inclusion relations between the above language classes are summarized in the diagram in Figure 2.1.

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

Figure 2.1: Hierarchy of some well-known classes of formal languages. An arrow denotes a proper inclusion, and classes that are not connected through a sequence of arrows are incomparable with respect to inclusion.

2.2 Relations and Transducers

We have seen that the automata introduced in the previous section can only accept or reject its input, that is, such an automaton simply determines whether the input belongs to the language that it accepts. Now we turn to *transducers* that are the devices realizing transductions and relations. A *(binary) relation* R is defined as a subset of the Cartesian product of two sets of words, i.e., $R \subseteq \Sigma^* \times \Delta^*$ for some alphabets Σ and Δ. The aim of this section is to present some well-known classes of relations and the corresponding transducers.

2.2.1 Rational Relations and Finite State Transducers

In this section we will shortly recall the class of *rational relations* (RAT for short), which is described by *finite state transducers* (see, e.g., [Ber79, Eil74]).

Definition 2.2.1 ([Ber79])**.** *A finite state transducer (*FST *for short) T is defined by a 6-tuple $T = (Q, \Sigma, \Delta, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, Δ is a finite output alphabet, $\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathbb{P}_{\text{fin}}(Q \times \Delta^*)$ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a finite set of final states.*

We see that an FST is obtained from an NFA by adding an output word to each (λ-)transition. The output produced during a computation is then simply the concatenation of all output words produced during that computation. Therefore, each FST represents a connection between an input language and an output language. A configuration of an FST T is a triple (q, u, v), where $q \in Q$ is the current state, $u \in \Sigma^*$ is the unread part of the input, and $v \in \Delta^*$ is the output produced so far. The transduction computed by T is defined as

$$T(u) = \{v \in \Delta^* \mid \exists q \in F : (q_0, u, \lambda) \vdash_T^* (q, \lambda, v)\}$$

for all $u \in \Sigma^*$. Let L be the language accepted by the underlying NFA [2] of an FST T, i.e., the input language (denoted by $L(T)$). Then the output language of T is defined as

$$T(L) = \{v \in T(u) \mid u \in L\}.$$

Theorem 2.2.1 ([Ber79])**.** *Let $T = (Q, \Sigma, \Delta, \delta, q_0, F)$ be an* FST*, and let $L \subseteq \Sigma^*$ be the input language of T. Then the output language $T(L) \subseteq \Delta^*$ is regular.*

The relation computed by an FST T is the set of pairs consisting of input and output sequences. Formally, it is defined as

$$Rel(T) = \{(u, v) \in \Sigma^* \times \Delta^* \mid v \in T(u)\}.$$

Now we present a simple example of an FST.

Example 2.2.1. *Let $T_1 = (\{q_0, q_1\}, \{a\}, \{b\}, \delta, q_0, \{q_0\})$ be an* FST*, where δ contains the following transitions:*

$$\delta(q_0, a) = \{(q_1, \lambda)\} \text{ and } \delta(q_1, a) = \{(q_0, b)\}.$$

It is easily seen that the underlying DFA *of T_1 accepts the input words w of even length, and for each input word $w = a^{2n}$, the output is b^n. Hence, the relation computed by T_1 is $Rel(T_1) = \{(a^{2n}, b^n) \mid n \geq 0\}$.*

It is well-known that the class RAT of rational relations can be characterized by FSTs [Ber79]. The applications of FSTs can be found in many areas like e.g., language and speech processing (see [Moh97]). Additionally, some transducer models based on FSTs are developed specially for the application in electronic commerce (see, e.g., [AVFY00, Spi00]).

[2]Here we consider the variant of finite state automata that can perform λ-steps. It is well-known that such an automaton A can be simulated by a finite state automaton A' that does not perform λ-steps, i.e., $L(A) = L(A')$ [HU79].

2.2.2 Pushdown Relations and Pushdown Transducers

In this section we recall the notion of *pushdown transducer* (PDT for short). In analogy to FST, a PDT is a PDA that produces an output word in each step, and the output for an accepting computation is the concatenation of all the output words that are produced during this computation.

Definition 2.2.2 ([CC83]). *A PDT is defined by an 8-tuple $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$, where Q is a finite set of states, Σ is an input alphabet, Δ is an output alphabet, X is a pushdown alphabet, $q_0 \in Q$ is the initial state, $Z_0 \in X$ is the bottom marker of the pushdown, $F \subseteq Q$ is the set of final states, and $E : Q \times (\Sigma \cup \{\lambda\}) \times X \to \mathbb{P}_{\text{fin}}(Q \times X^* \times \Delta^*)$ is a transition function that produces a (possible empty) output word in each step.*

A configuration of a PDT T is written as a 4-tuple (q, u, α, v), where $q \in Q$ is the current state, $u \in \Sigma^*$ is the still unread part of the input, $\alpha \in X^*$ is the current content of the pushdown with the first letter of α at the bottom, and $v \in \Delta^*$ is the output produced so far. The language accepted by the underlying PDA of T is called the *input language* of T and denoted by $L(T)$. The relation computed by T is defined as

$$Rel(T) = \{\, (u, v) \in \Sigma^* \times \Delta^* \mid \exists q \in F, \alpha \in X^* : (q_0, u, Z_0, \lambda) \vdash_T^* (q, \lambda, \alpha, v) \,\}.$$

Concerning the output language of a PDT the following result is known.

Theorem 2.2.2 ([Eve63]). *If $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$ is a PDT, and if L is the input language of T, then the output language $T(L) \subseteq \Delta^*$ is context-free.*

Now we present a simple example of a PDT.

Example 2.2.2. *Let $T_2 = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$ be the PDT, where $Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}\}$, $\Sigma = \{a, b\}$, $\Delta = \{c, d\}$, $X = \{Z_0, Z_1\}$, $F = \{q_{acc}\}$, and δ is defined as follows:*

$$\begin{array}{lrcl}
(1) & \delta(q_0, a, Z_0) & = & \{(q_1, Z_0Z_1, \lambda)\}, \\
(2) & \delta(q_1, a, Z_1) & = & \{(q_1, Z_1Z_1, \lambda)\}, \\
(3) & \delta(q_0, \lambda, Z_0) & = & \{(q_{acc}, Z_0, \lambda)\}, \\
(4) & \delta(q_1, b, Z_1) & = & \{(q_2, \lambda, c), (q_3, \lambda, d)\}, \\
(5) & \delta(q_2, b, Z_1) & = & \{(q_2, \lambda, c)\}, \\
(6) & \delta(q_2, \lambda, Z_0) & = & \{(q_{acc}, Z_0, \lambda)\}, \\
(7) & \delta(q_3, b, Z_1) & = & \{(q_4, Z_1, \lambda)\}, \\
(8) & \delta(q_4, b, Z_1) & = & \{(q_3, \lambda, d)\}, \\
(9) & \delta(q_4, \lambda, Z_0) & = & \{(q_{acc}, Z_0, \lambda)\}.
\end{array}$$

It is easily seen that the input language

$$L(T_2) = \{a^n b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}.$$

On an input of the form $w = a^n b^n$ for $n \geq 0$, the output c^n is produced during the computation, and on an input of the form $w = a^n b^{2n}$ for $n \geq 0$, the corresponding output is d^n. We see that all pairs in the relation $Rel(T_2)$ are of the form $(a^n b^n, c^n)$ or $(a^n b^{2n}, d^n)$, that is,

$$Rel(T_2) = \{(a^n b^n, c^n) \mid n \geq 0\} \cup \{(a^n b^{2n}, d^n) \mid n \geq 0\}.$$

A relation $R \subseteq \Sigma^* \times \Delta^*$ is called a *pushdown relation*, if $R = Rel(T)$ for some PDT T. For example, the above relation $Rel(T_2)$ presented in Example 2.2.2 is a pushdown relation. By PDR we denote the class of all pushdown relations. In [WO16a] we have introduced some restricted types of pushdown relations. If T is a deterministic PDT (DPDT for short), then $Rel(T)$ is called a *deterministic pushdown relation*. By DPDR we denote the class of all deterministic pushdown relations. A pushdown relation R is called *linearly bounded*, if there exists a constant $c \in \mathbb{N}$ such that $|v| \leq c \cdot |u|$ holds for all pairs $(u, v) \in R$, $u \neq \lambda$. By lbPDR we denote the class of all linearly bounded pushdown relations. A pushdown relation R is called *realtime*, if it is computed by a PDT $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$ that does not perform any λ-step, that is, the transition function δ is of the form $\delta : Q \times \Sigma \times X \to \mathbb{P}_{\text{fin}}(Q \times X^* \times \Delta^*)$. By rtPDR we denote the class of all realtime pushdown relations. Finally, a pushdown relation R is called *almost-realtime* [3] if it is computed by a PDT $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$ for which each λ-step pops a symbol from the pushdown, that is, if T has a λ-transition $(q', x', v) \in \delta(q, \lambda, x)$, then $x' = \lambda$. By artPDR we denote the class of all almost-realtime pushdown relations.

Now we study the inclusion relations between the various types of pushdown relations. First, we consider the relation

$$R_{uu^R} = \{\, (u, uu^R) \mid u \in \{a, b\}^* \,\},$$

taken from [WO16a] and give the following negative result.

Lemma 2.2.1 ([WO16a])**.** $R_{uu^R} \notin \mathsf{rtPDR}$.

Proof. We assume that $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$ is a realtime PDT such that $Rel(T) = R_{uu^R}$, where $Q = \{q_0, q_1, \ldots, q_m\}$ is a finite set of states, $\Sigma = \{a, b\}$ is the input alphabet, $\Delta = \{a, b\}$ is the output alphabet, $X = \{Z_0, Z_1, \ldots, Z_k\}$ is the pushdown alphabet containing the bottom marker Z_0, q_0 is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \times X \to \mathbb{P}_{\text{fin}}(Q \times X^* \times \Delta^*)$ is the transition function. By the above definition

$$\begin{aligned} Rel(T) &= \{\, (u, v) \in \Sigma^* \times \Delta^* \mid \exists q \in F, \alpha \in X^* : (q_0, u, Z_0, \lambda) \vdash_T^* (q, \lambda, \alpha, v) \,\} \\ &= \{\, (u, uu^R) \mid u \in \{a, b\}^* \,\} \;=\; R_{uu^R}. \end{aligned}$$

[3]In our paper [WHO15] such a relation is called a *quasi-realtime* pushdown relation. However, the class of quasi-realtime languages has been introduced in [BG70], where this notion expresses the fact that between any two reading steps only a bounded number of λ-transitions are possible. To not confuse them, this class is renamed in [WO16a].

It follows that for each input word $u = a_1a_2\ldots a_n$, where $a_1, a_2, \ldots, a_n \in \Sigma$, T has an accepting computation of the form

$$\begin{aligned}(q_0, u, Z_0, \lambda) &\vdash_T (q_{i_1}, a_2a_3\ldots a_n, \alpha_1, v_1)\\ &\vdash_T (q_{i_2}, a_3a_4\ldots a_n, \alpha_2, v_1v_2)\\ &\vdash_T \ldots\\ &\vdash_T (q_{i_{n-1}}, a_n, \alpha_{n-1}, v_1v_2\ldots v_{n-1})\\ &\vdash_T (q_{i_n}, \lambda, \alpha_n, v_1v_2\ldots v_{n-1}v_n)\\ &= (q_{i_n}, \lambda, \alpha_n, uu^R),\end{aligned}$$

where $q_{i_1}, q_{i_2}, \ldots, q_{i_{n-1}} \in Q$, $q_{i_n} \in F$, $\alpha_1, \alpha_2, \ldots, \alpha_n \in X^*$, and $v_1, v_2, \ldots, v_n \in \Delta^*$.

Let $c = \max\{\, |v| \mid \exists q, a, x, q', x' : \delta(q, a, x) = (q', x', v)\,\}$, that is, c is the maximal length of an output word that T can produce in a single step. Observe that $c \geq 2$, as T produces an output of length $2n$ from an input of length n. In order to output the suffix u^R of the output, T needs at least $\lceil \frac{n}{c} \rceil$ steps. Let the number $l \geq \lfloor \frac{n}{c} \rfloor$ such that T outputs the prefix u of the output (possibly with the first few symbols of u^R) during the first $n - l$ steps, and it outputs the major part of u^R during the last l steps. It is easy to see that $n - \lfloor \frac{n}{c} \rfloor \geq n - l \geq \lceil \frac{n}{c} \rceil$. Next, we assume that T needs at least s steps to produce the prefix $a_1a_2\ldots a_{n-l}$ as output. Then $\lceil \frac{n-l}{c} \rceil \leq s \leq n - l - \lceil \frac{l+1-c}{c} \rceil$, as in steps $s+1$ to $n-l$, the suffix $a_{n-l+c-1}a_{n-l+c}\ldots a_n$ of u is produced. Thus, it follows that

$$|a_{s+1}a_{s+2}\ldots a_{n-l}| = n - l - s \geq \left\lceil \frac{l+1-c}{c} \right\rceil \geq \frac{l+1-c}{c} \in \Omega\left(\frac{n}{c^2}\right),$$

that is, while the infix $a_{s+1}a_{s+2}\ldots a_{n-l}$ of u of length $\Omega\left(\frac{n}{c^2}\right)$ is being read, the suffix $a_{n-l+c-1}a_{n-l+c}\ldots a_n$ of length $l - c + 1 \geq \lfloor \frac{n}{c} \rfloor - c + 1$ of the prefix u of the output is produced. Further, there is an index $m \geq n - l + \Omega\left(\frac{n}{c^3}\right)$ such that T outputs the prefix $a_na_{n-1}\ldots a_{s+1}$ of u^R while processing the factor $a_{n-l+1}a_{n-l+2}\ldots a_m$ of the input. Since the suffix $a_sa_{s-1}\ldots a_1$ of the output must be produced while the rest $a_{m+1}a_{m+2}\ldots a_n$ of the input is being read, we obtain that $n - m \geq \lfloor \frac{s}{c} \rfloor \geq \lfloor \frac{n-l}{c^2} \rfloor$. It follows that T produces the factor

$$a_{n-l+c-1}a_{n-l+c}\ldots a_na_n\ldots a_{s+2}a_{s+1}$$

of the output while reading the factor $a_{s+1}a_{s+2}\ldots a_{n-l}a_{n-l+1}\ldots a_m$ of the input. Let $s' = \max\{s+1, n-l+c-1\}$, then the factor $a_{s'}a_{s'+1}\ldots a_na_n\ldots a_{s'+1}a_{s'}$ of the output uu^R is produced while the factor $a_{s+1}a_{s+2}\ldots a_{n-l}a_{n-l+1}\ldots a_m$ of the input is being read.

However, T has to produce the prefix $a_na_{n-1}\ldots a_{s'+1}a_{s'}$ of the output u^R that matches the previous output $a_{s'}a_{s'+1}\ldots a_{n-1}a_n$. For this purpose, when processing the factor $a_{s+1}a_{s+2}\ldots a_{n-l}$ of the input and producing the output

$a_{s'}a_{s'+1}\dots a_n$, T must store the word $a_{s'}a_{s'+1}\dots a_n$ (possibly in an encoded form of this word) on its pushdown. Then, when it processes the input symbols $a_{n-l+1}a_{n-l+2}\dots a_m$, it is to produce the output $a_na_{n-1}\dots a_{s'+1}a_{s'}$, corresponding to the previous output $a_{s'}a_{s'+1}\dots a_{n-1}a_n$ that is stored on the pushdown. To do this, T has to take (and hence, also pop) this word from the pushdown. However, now T cannot remember the suffix $a_{s'}a_{s'+1}\dots a_n$ of the output produced, and it has not yet seen the complete suffix $a_{m+1}a_{m+2}\dots a_n$ of the input. Hence, T is not able to determine whether the suffix $a_{s'}a_{s'+1}\dots a_n$ of the output produced is correct, and it can only compare the suffix $a_{m+1}a_{m+2}\dots a_n$ of the input to the output word $a_{s'}a_{s'+1}\dots a_n$ of length $n-m$. As $n-m \geq \lfloor \frac{n-l}{c^2} \rfloor \in \Omega(\frac{n}{c^3})$, T cannot store this information in its finite-state control. Accordingly, it is not able to correctly compare the remaining part of the input to the corresponding part of the output already produced. Thus, R_{uu^R} cannot be computed by any realtime PDT. □

By Lemma 2.2.1 the following inclusion result can be easily established.

Proposition 2.2.1 ([WO16a]). rtPDR $\subsetneq$ artPDR.

Proof. From the above definitions, we see that each realtime pushdown relation is also an almost-realtime relation. Thus, the relation class rtPDR is a subset of the relation class artPDR, i.e., rtPDR $\subseteq$ artPDR. To show the properness of this inclusion, we consider the relation R_{uu^R} defined above. We can construct a PDT T such that $Rel(T) = R_{uu^R}$, where T proceeds as follows. For an input word $u = a_1a_2\dots a_n$, first T reads all the input symbols and outputs them, also pushing them onto the pushdown. This means that the output $u = a_1a_2\dots a_n$ is produced. In each reading step, T guesses whether the current symbol is the end of the input, and it empties its pushdown letter by letter, producing the output $u^R = a_na_{n-1}\dots a_1$. Obviously, T is an almost-realtime PDT, which completes the current proof. □

We continue by studying the inclusion relation between the relation classes artPDR and lbPDR. Let $R_{a^mb^mc^n} \subseteq \{a,b,c\}^* \times \{a,b,c\}^*$ be the relation

$$R_{a^mb^mc^n} = \{\, (a^mb^mc^n, c^na^mb^m) \mid m,n \geq 1 \,\}.$$

Lemma 2.2.2 ([WO16a]). $R_{a^mb^mc^n} \notin$ artPDR.

Proof. Assume that $T = (Q,\Sigma,\Delta,X,q_0,Z_0,F,\delta)$ is an almost-realtime PDT such that $Rel(T) = R_{a^mb^mc^n}$. Here $Q = \{q_0,q_1,\dots,q_m\}$ is a finite set of states, $\Sigma = \{a,b,c\}$ is the input alphabet, $\Delta = \{a,b,c\}$ is the output alphabet, $X = \{Z_0,Z_1,\dots,Z_k\}$ is the pushdown alphabet containing the bottom marker Z_0, q_0 is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times (\Sigma \cup \{\lambda\}) \times X \to \mathbb{P}_{\text{fin}}(Q \times X^* \times \Delta^*)$ is the transition function, where $\delta(p,\lambda,x) = (q,x',v)$ implies that $x' = \lambda$, that is, a λ-step must pop from the pushdown.

Let $c = \max\{\, |v| \mid \exists p, a, x, q, x' : \delta(p, a, x) = (q, x', v) \,\}$, that is, c is the maximal length of an output word that T can produce in a single step, and let $d = \max\{\, |x'| - 1 \mid \exists p, a, x, q, v : \delta(p, a, x) = (q, x', v) \in \delta \,\}$, that is, d is the maximal size increase of the pushdown that T can realize in a single step.

As $Rel(T) = R_{a^m b^m c^n}$, for each input word of the form $a^m b^m c^n$, T has an accepting computation that produces the output $c^n a^m b^m$. To do this, T must first guess the number of c-symbols and then produce this many c-symbols as the prefix of the output. While processing the prefix $a^m b^m$ of the input, T can push at most $2 \cdot m \cdot d$ symbols onto its pushdown, and so it can execute at most this many λ-steps. Thus, this initial part of the computation consists of at most $2 \cdot m \cdot (d + 1)$ steps, and accordingly, during this phase at most $2 \cdot m \cdot (d + 1) \cdot c$ symbols can be produced as output. Thus, if we choose n to be a constant such that $n > 2 \cdot m \cdot (d + 1) \cdot c$, then only c-symbols have been produced as output while consuming all the a- and b-symbols. Let r be the number of the c-symbols that are produced so far, then the output c^{n-r} is still to be produced. Now T is in some internal state, and it has some finite string on its pushdown. If n is sufficiently large, T has to consume even some symbols of the suffix c^n of the input to produce the remaining part c^{n-r} of the prefix of the output. After that, T must compare the input syllable b^m to the prefix a^m and produce the suffix $a^m b^m$ of the output, while reading the suffix c^n of the input. For this purpose, it must store the syllable a^m (possibly in an encoded form of it) on its pushdown while reading it, and then it must read (and thereby pop) this information from the pushdown when reading b^m. However, the information of the number of a- and b-symbols is lost when producing the syllable c^n. In addition, T cannot store this information in its finite-state control as the number m is an arbitrary positive integer. Therefore, T is not able to remember the correct value of m, and it can only output just some suffix of the form $a^{m'} b^{m'}$, where $m = m'$ cannot be verified. Thus, $Rel(T) \neq R_{a^m b^m c^n}$, which shows that $R_{a^m b^m c^n}$ does not belong to the relation class artPDR. $\square$

By Lemma 2.2.2 the following inclusion result is easily obtained.

Proposition 2.2.2 ([WO16a]). $\mathsf{artPDR} \subsetneq \mathsf{lbPDR}$.

Proof. Let R be the relation that is computed by an almost-realtime PDT T, and let $c \geq 1$ be the maximal length of a word that T can push on its pushdown in a single step. During a computation of T on an input word u, T can push at most $c \cdot |u|$ symbols onto its pushdown, and so it can excute at most this many λ-transitions. Therefore, for the input word u, the computation consists of at most $(c + 1) \cdot |u|$ steps. Let v be the output for u, and let d be the maximal length of any output string produced by T in a single step. It follows that the output v produced during this computation satisfies the inequality $|v| \leq d \cdot (c+1) \cdot |u|$. Thus, R is also linearly bounded, and so the class artPDR is a subset of the class lbPDR, i.e., $\mathsf{artPDR} \subseteq \mathsf{lbPDR}$.

In Lemma 2.2.2 we have seen that $R_{a^m b^m c^n} \notin \mathsf{artPDR}$. In order to prove the properness of the above inclusion, for the relation $R_{a^m b^m c^n}$ we construct a linearly bounded PDT T that proceeds as follows. First, let T guess the value of n and output this many c-symbols, pushing them onto its pushdown. Next, T compares the syllable a^m to the syllable b^m and outputs $a^m b^m$. Finally, T takes c-symbols from its pushdown in order to check its guess for the value of n. Hence, we see that $R_{a^m b^m c^n}$ is linearly bounded, which completes our proof. □

Next, we study the inclusion relation between the relation classes lbPDR and PDR.

Proposition 2.2.3. $\mathsf{lbPDR} \subsetneq \mathsf{PDR}$.

Proof. Clearly, the class lbPDR is a subset of the class of PDR, i.e., $\mathsf{lbPDR} \subseteq \mathsf{PDR}$. In order to prove the properness of this inclusion, we consider the relation

$$R_{+} = \{\, (a^m, b^n a^m b^n) \mid m, n \geq 1 \,\}.$$

For the relation R_{+} we construct a PDT T that proceeds as follows. First, T performs some λ-transitions, and in each step it outputs a b-symbol and pushes it onto its pushdown. Then, T reads the input a^m and produces an a-symbol in each reading step without changing the pushdown. Finally, it outputs the suffix b^n by reading the information on the number n from its pushdown. If follows that $Rel(T) = R_{+}$, and thus $R_{+} \in \mathsf{PDR}$. Obviously, R_{+} is not linearly bounded, which completes the current proof. □

From the inclusion results above, now we can obtain the following consequence for the inclusion relations between the above relation classes.

Theorem 2.2.3. $\mathsf{rtPDR} \subsetneq \mathsf{artPDR} \subsetneq \mathsf{lbPDR} \subsetneq \mathsf{PDR}$.

The class of pushdown relations can be characterized in terms of context-free languages and morphisms. For that we recall the following concept.

Definition 2.2.3 ([AU72]). *A language $L \subseteq \Gamma^*$* characterizes *a relation $R \subseteq \Sigma^* \times \Delta^*$ if there exist two morphisms $h_1 : \Gamma^* \to \Sigma^*$ and $h_2 : \Gamma^* \to \Delta^*$ such that $R = \{\, (h_1(w), h_2(w)) \mid w \in L \,\}$.*

In Chapter 3 of [AU72] it was shown that the pushdown relations are characterized by the context-free languages. We now introduce a stronger notion.

Definition 2.2.4 ([AU72]). *A language $L \subseteq (\Sigma \cup \Delta')^*$* strongly characterizes *a relation $R \subseteq \Sigma^* \times \Delta^*$ if*

1. $\Sigma \cap \Delta' = \emptyset$ and

2. there exist two morphisms $h_1 : (\Sigma \cup \Delta')^* \to \Sigma^*$ and $h_2 : (\Sigma \cup \Delta')^* \to \Delta^*$ such that $R = \{(h_1(w), h_2(w)) \mid w \in L\}$, where

(2.1) $h_1(a) = a$ for all $a \in \Sigma$ and $h_1(b) = \lambda$ for all $b \in \Delta'$,

(2.2) $h_2(a) = \lambda$ for all $a \in \Sigma$ and h_2 is a copy isomorphism between Δ and Δ', that is, $h_2(b) \in \Delta$ for all $b \in \Delta'$ and $h_2(b) = h_2(b')$ implies that $b = b'$.

In terms of [AU72] this is expressed by saying that a certain subclass of pushdown relations are strongly characterized by the context-free languages. In the following we extend this result to lbPDR.

Proposition 2.2.4 ([WHO15])**.** *Every linearly bounded pushdown relation is strongly characterized by a context-free language.*

Proof. Let $R \subseteq \Sigma^* \times \Delta^*$ be an lbPDR, and let c be a constant such that $|v| \leq c \cdot |u|$ for all $(u, v) \in R$. From Definition 2.2.3 it follows that R is characterized by a context-free language $L \subseteq \Gamma^*$ and two morphisms $h_1 : \Gamma^* \to \Sigma^*$ and $h_2 : \Gamma^* \to \Delta^*$. Thus, for each pair $(u, v) \in R$, there is a word $w \in L$ such that $h_1(w) = u$ and $h_2(w) = v$. Now a strong characterization would put the additional restriction $|w| \leq |u| + |v| \leq (c+1) \cdot |u|$ on the length of w, which is not necessarily the case for the above characterization in terms of L.

To simplify the discussion, we assume that Γ, Σ, and Δ are pairwise disjoint. We introduce an additional alphabet $\Gamma' = \{\, x' \mid x \in \Gamma, h_2(x) \neq \lambda \,\}$ and take $\Gamma_0 = \Gamma \cup \Gamma'$. Further, we define a morphism $h : \Gamma^* \to \Gamma_0^*$, where $x \in \Gamma$:

$$h(x) = \begin{cases} xx', & \text{if } h_1(x) \neq \lambda \text{ and } h_2(x) \neq \lambda, \\ x', & \text{if } h_1(x) = \lambda \text{ and } h_2(x) \neq \lambda, \\ x, & \text{otherwise,} \end{cases}$$

and we extend h_1 and h_2 to morphisms $h_1' : \Gamma_0^* \to (\Gamma' \cup \Sigma)^*$ and $h_2' : (\Gamma' \cup \Sigma)^* \to (\Sigma \cup \Delta)^*$ through $h_1'(x) = \left\{ \begin{array}{ll} h_1(x), & x \in \Gamma \\ x, & x \in \Gamma' \end{array} \right\}$ and $h_2'(x') = \left\{ \begin{array}{ll} h_2(x), & x' \in \Gamma' \\ x', & x' \in \Sigma \end{array} \right\}$.

Clearly, the language $L' = h_2'(h_1'(h(L))) \subseteq (\Sigma \cup \Delta)^*$ is context-free. Let π^Σ and π^Δ be the projections from $(\Sigma \cup \Delta)^*$ onto Σ^* and Δ^*. Then R is strongly characterized by L' and the two projections π^Σ and π^Δ. □

Finally, we compare the relation class RAT that is introduced in Section 2.2.1 to the relation class PDR. Let the relation

$$R_{a^n b^n} = \{\, (a^n b^n, c^n) \mid n \in \mathbb{N} \,\}.$$

Obviously, $R_{a^n b^n}$ is a pushdown relation, while it is not rational, as an FST is not able to compare the syllable a^n to the syllable b^n. Further, it is rather clear that each FST can be simulated by a PDT. Hence, we have the following proper inclusion result.

Corollary 2.2.1 ([CC83])**.** $\mathsf{RAT} \subsetneq \mathsf{PDR}$.

Note that an FST can compute a relation that is not linearly bounded. Let T be an FST over $\Sigma = \{a\}$ that proceeds as follows. First, T reads all input symbols and outputs this many b-symbols. Then, it performs nondeterministically many λ-steps, and in each step it produces a b-symbol as output. It is easy to see that $Rel(T) = \{(a^n, b^m) \mid m \geq n \geq 0\}$. Obviously, there does not exist a constant c such that $m \leq c \cdot n$. On the other hand, the relation $R_{a^n b^n}$ is actually a realtime pushdown relation, while it is not rational. This leads to the following incomparability result.

Proposition 2.2.5. *The relation class* RAT *is incomparable to the relation classes* rtPDR, artPDR *and* lbPDR *with respect to inclusion.*

Pushdown Functions

Even though a DPDT T has only a single computation for each input $w \in \Sigma^*$, it may produce several different outputs. This is exemplified by the following simple example.

Example 2.2.3. *Let $T_3 = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$ be a* DPDT*, where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Delta = \{c, d, e\}$, $X = \{Z_0, Z_1\}$, $F = \{q_1, q_2\}$, and δ is defined as follows:*

$$\begin{array}{llllll}
(1) & \delta(q_0, a, Z_0) = \{(q_0, Z_0Z_1, c)\}, & (4) & \delta(q_0, b, Z_1) = \{(q_1, Z_1, d)\}, \\
(2) & \delta(q_0, a, Z_1) = \{(q_0, Z_1Z_1, c)\}, & (5) & \delta(q_1, \lambda, Z_0) = \{(q_2, \lambda, e)\}, \\
(3) & \delta(q_0, b, Z_0) = \{(q_1, Z_0, d)\}, & (6) & \delta(q_1, \lambda, Z_1) = \{(q_1, \lambda, e)\}.
\end{array}$$

It is easily seen that the input language is $L(T_3) = \{\, a^n b \mid n \geq 0 \,\}$. On input $w = a^n b$, the computation of T_3 proceeds as follows, where $m \leq n$:

$$\begin{array}{llllll}
(q_0, a^n b, Z_0, \lambda) & \vdash_{T_3}^{n} & (q_0, b, Z_0Z_1^n, c^n) & \vdash_{T_3} & (q_1, \lambda, Z_0Z_1^n, c^n d) \\
& \vdash_{T_3}^{m} & (q_1, \lambda, Z_0Z_1^{n-m}, c^n de^m) & \vdash_{T_3}^{n-m} & (q_1, \lambda, Z_0, c^n de^n) \\
& \vdash_{T_3} & (q_2, \lambda, \lambda, c^n de^{n+1}). & &
\end{array}$$

As q_1 and q_2 are final states, we see that $Rel(T)$ contains all pairs of the form $(a^n b, c^n de^k)$, where $0 \leq k \leq n+1$, that is,

$$Rel(T) = \{\, (a^n b, c^n de^k) \mid n \geq 0 \text{ and } 0 \leq k \leq n+1 \,\}.$$

Observe that by replacing transition (5) by (5′) $\delta(q_1, \lambda, Z_0) = (q_1, Z_0, e)$ we obtain a DPDT T_3' *that does not halt on input $a^n b$, satisfying*

$$Rel(T_3') = \{\, (a^n b, c^n d e^k) \mid n \geq 0 \text{ and } k \geq 0 \,\},$$

that is, $T_3'(a^n b) = \{\, c^n d e^k \mid k \geq 0 \,\}$ is even infinite.

In the above example we see that T_3' can produce infinitely many e-symbols by performing λ-steps in the final state q_1. Further, a DPDT is not able to guess whether the next symbol is the last symbol of the input. To resolve these two problems, we introduce a *well-behaved* DPDT (wbDPDT for short) that does not perform any λ-step in a final state, and that has a specific end marker on its input tape. Thus, given an input word w, a wbDPDT actually starts with the word $w\#$ on its input tape, where $\# \notin \Sigma$ is a special symbol. It is well-known that each DPDA is equivalent to a DPDA that does not perform λ-steps in final states (see, e.g., [HU79]). Actually, specific end markers are used quite commonly for DPDTs (see, e.g., [AU69]). Obviously, for a wbDPDT T, the relation $Rel(T)$ is the graph of a (partial) function. By DPDF we denote the class of all (partial) functions that are computed by wbDPDTs, by lbDPDF we denote the subclass of DPDF that consists of all (partial) functions that are linearly bounded, by artDPDF we denote the class of all (partial) functions that are computed by wbDPDTs that must pop from their pushdowns during λ-steps, and finally, rtDPDF denotes the class of all (partial) functions that are computed by wbDPDTs without λ-steps.

The relation $R_{uu^R} = \{\, (u, uu^R) \mid u \in \{a,b\}^* \,\}$ is the graph of the partial function $f_{uu^R} : \{a,b\}^* \to \{a,b\}^*$ given by $u \mapsto uu^R$ for all $u \in \{a,b\}^*$. It is easily seen that R_{uu^R} can be computed by an almost-realtime wbDPDT T that proceeds as follows. Given an input word u, T starts with the word $u\#$ on the tape, and it just reads the word u and pushes it onto its pushdown, also producing output u during these reading steps. On seeing the symbol $\#$, T empties its pushdown letter by letter, producing the output u^R. On the other hand, from Lemma 2.2.1 we know that R_{uu^R} does not belong to the relation class rtPDR. Thus, we have the following proper inclusion.

Proposition 2.2.6 ([WO16a])**.** *rtDPDF $\subsetneq$ artDPDF.*

Concerning the function class lbDPDF, we have the following result.

Proposition 2.2.7 ([WO16a])**.** *lbDPDF = DPDF.*

Proof. The inclusion from left to right is obvious, and thus it remains to show the opposite direction.

Let $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, \delta)$ be a wbDPDT. In order to show that $Rel(T)$ is linearly bounded, it suffices to prove that T is linearly time bounded. For this purpose, we have to determine an upper bound for the number of steps in an accepting computation of T for a given input. Given an input word $w \in \Sigma^n$, the corresponding accepting computation of T contains $n+1$ reading steps on the input symbols and the end marker $\#$. Additionally, T may execute a sequence of λ-steps before the first reading step, between two reading steps, and after the last reading step. Therefore, such an accepting computation consists of $n+1$ reading steps and $n+2$ sequences of λ-steps. We can partition this computation into $n+2$ phases as follows. For $1 \leq i \leq n+1$, phase i consists of a (possibly empty) sequence of λ-steps that is followed by the step that reads the i-th symbol of $w\#$, and phase $n+2$ consists of a (possibly empty) sequence of λ-steps that ends by reaching a final state. Clearly, the number of reading steps is linearly bounded. Now we have to determine an upper bound for the length of these sequences of λ-steps.

For $q \in Q \setminus F$ and $x \in X$, if δ contains a λ-transition of the form $\delta(q, \lambda, x) = (p, \gamma, v)$, then we define the set

$$N(q,x) = \{\, (q', \alpha) \in Q \times X^* \mid \exists v \in \Delta^* : (q, \lambda, x, \lambda) \vdash_T^+ (q', \lambda, \alpha, v) \,\},$$

that is, $N(q,x)$ consists of all pairs (q', α) such that T can reach state q' and pushdown content α through a non-empty sequence of λ-steps when starting in state q with pushdown content x. As T cannot proceed with an empty pushdown, we see that the pushdown is not emptied during the computation $(q, \lambda, x, \lambda) \vdash_T^+ (q', \lambda, \alpha, v)$, or it is emptied in the last step and $\alpha = \lambda$. If the set $N(q,x)$ is infinite, or if it contains a pair (q', α) such that $(q', \lambda, \alpha, \lambda) \vdash_T^+ (q', \lambda, \alpha, v)$ holds for some $v \in \Delta^*$, then T cannot use the λ-transition $\delta(q, \lambda, x) = (p, \gamma, v)$ in any of its accepting computations. Thus, we can simply delete this λ-transition from T without effecting $Rel(T)$. Accordingly, we can now assume that all the sets $N(q,x)$ are finite, and that each of them can be linearly ordered with respect to the computation relation induced by T, that is, $N(q,x) = \{(q_1, \alpha_1), (q_2, \alpha_2), \ldots, (q_m, \alpha_m)\}$, where

$$(q, \lambda, x, \lambda) \vdash_T (q_1, \lambda, \alpha_1, v_1) \vdash_T (q_2, \lambda, \alpha_2, v_1 v_2) \vdash_T \ldots \vdash_T (q_m, \lambda, \alpha_m, v_1 v_2 \ldots v_m),$$

and $v_1, v_2, \ldots, v_m \in \Delta^*$ are the output words generated in these steps. From the definition of $N(q,x)$ we see that either $\alpha_m = \lambda$, or $\alpha_m = \gamma_m y$ for some $y \in X$ and T does not have a λ-step for state q_m and pushdown symbol y. We now replace the λ-transition $\delta(q, \lambda, x) = (p, \gamma, v)$ by $\delta(q, \lambda, x) = (q_m, \alpha_m, v_1 v_2 \ldots v_m)$, which yields a wbDPDT T' from T such that $Rel(T') = Rel(T)$. Now T' only has two types of λ-transitions:

- those that pop from the pushdown, which are obtained from the sets of the form $N(q,x)$ containing a pair (p, λ), and
- those that do not decrease the height of the pushdown, and which cannot be followed by another λ-transition, which are obtained from the remaining sets of the form $N(q,x)$.

Thus, for an input $w \in \Sigma^n$, if $w \in L(T) = L(T')$, then the accepting computation of T' consists of $n+2$ phases. Initially, the pushdown contains only the symbol Z_0. As T' cannot proceed with an empty pushdown, the first phase consists of at most a single λ-step that does not pop from the pushdown, and it is followed by the step that reads the first input symbol of w. In the i-th phase, for $2 \leq i \leq n+1$, T' may perform a (possibly empty) sequence of λ-steps that pop from the pushdown, possibly a single λ-step that does not decrease the height of the pushdown, and a reading step on the i-th symbol of $w\#$. Finally, phase $n+2$ consists of a (possibly empty) sequence of λ-steps that pop from the pushdown and possibly a single λ-step that does not decrease the height of the pushdown and that reaches a final state. Therefore, we can see that during this computation, there are at most $n+1$ reading steps and $n+2$ λ-steps that push symbols onto the pushdown. Further, the pushdown initially contains the bottom marker Z_0. It follows that the height of the pushdown is bounded from above by $C \cdot (2n+3)+1$ for a constant C that only depends on T. This means that at most $C \cdot (2n+3)+1$ λ-steps can be executed that pop from the pushdown. Together with the $2n+3$ steps that push symbols onto the pushdown, the computation of T' on input $x \in \Sigma^n$ consists of at most $2n+3+C\cdot(2n+3)+1 = (2C+2)\cdot n + 3C + 4$ steps. Therefore, the length of the output produced during this computation is bounded by the number $D \cdot ((2C+2)\cdot |x| + 3C + 4)$ for some constant D that only depends on T, which completes this proof. □

We now compare artDPDF to lbDPDF and establish the following result.

Proposition 2.2.8 ([WO16a]). artDPDF = lbDPDF.

Proof. Continuing with the linearly bounded DPDT T' from the above proof, we will show that $Rel(T')$ can be computed by an almost-realtime DPDT. First, we combine each non-deleting λ-transition of the form $\delta(q,\lambda,x) = (q',\alpha y, v)$ with each reading transition of the form $\delta(q',a,y) = (q'',\gamma,v')$ into a transition of the form $\delta(q,a,x) = (q'',\alpha\gamma,vv')$, where $q \in Q \smallsetminus F$, $q',q'' \in Q$, $a \in \Sigma$, $\alpha,\gamma \in X^*$, $x,y \in X$, and $v,v' \in \Delta^*$. Further, we modify the state set and the transition function in such a way that each possible λ-transition that is applicable after the end marker # has been read just pops from the pushdown. Note that this does not result in a change for the relation $Rel(T')$, as the only λ-steps that do not pop from the pushdown and that can be used in an accepting computation after the end marker has been read end the computation by entering a final state. By these changes we obtain an almost-realtime DPDT T''. For an input $w \in L(T')$ of length n, in analogy to the above proof, the corresponding accepting computation of T' can be partitioned into $n+2$ phases. From the definition above it follows immediately that T'' can simulate phases 1 to $n+1$ of this computation. In addition, as mentioned above, it can execute the deleting λ-steps of phase $n+2$, and the final non-deleting λ-step is of this phase is replaced by a deleting λ-step. Hence, it follows that $Rel(T'') = Rel(T')$ holds. Thus, we obtain the above result. □

Together Propositions 2.2.6, 2.2.7 and 2.2.8 yield the following summary result.

Theorem 2.2.4. $\mathsf{rtDPDF} \subsetneq \mathsf{artDPDF} = \mathsf{lbDPDF} = \mathsf{DPDF}$.

Further, we study the inclusion relation between the relation classes artDPDF and rtPDR.

Theorem 2.2.5. *The relation class* artDPDF *is incomparable to the relation class* rtPDR *with respect to inclusion.*

Proof. First, as mentioned above, the relation R_{uu^R} does not belong to the relation class rtPDR, while it can be computed by an almost-realtime wbDPDT. Now we consider the relation

$$R_{ab} = \{\, (a^n b^n, c^n) \mid n \geq 0 \,\} \cup \{\, (a^n b^{2n}, d^n) \mid n \geq 0 \,\}.$$

We construct a realtime PDT T that proceeds as follows. Given an input word w, T guesses whether $|w|_a = |w|_b$ or $2|w|_a = |w|_b$. In the case that $|w|_a = |w|_b$, T pushes all a-symbols onto its pushdown, and when processing the syllable b^n, it empties its pushdown letter by letter, producing the output c^n. If T guesses that $2|w|_a = |w|_b$, then after pushing all a-symbols onto the pushdown, it first simply reads a b-symbol, and then it reads a b-symbol, also popping an a-symbol from its pushdown and producing a d-symbol as output, alternatingly. Therefore, by consuming all b-symbols T can produce the output d^n without performing λ-steps. On the other hand, it is easily seen that the relation R_{ab} cannot be computed by any DPDT, as the language $L_{ab} = \{a^n b^n, a^n b^{2n} \mid n \geq 0\}$ is not deterministic context-free. Hence, the incomparability result can be obtained. □

In this section we introduced pushdown relations and some restricted types of pushdown relations. Finally, we summarize the results on the inclusion relations between these relation classes in the diagram in Figure 2.2.

$$\begin{tikzcd}
 & & \mathsf{PDR} & \\
 & & \mathsf{lbPDR} \arrow[u] & \\
 & & \mathsf{artPDR} \arrow[u] & \mathsf{rtPDR} \arrow[l] \\
\mathsf{DPDF} \arrow[r, equal] & \mathsf{lbDPDF} \arrow[r, equal] & \mathsf{artDPDF} \arrow[u] & \mathsf{rtDPDF} \arrow[l] \arrow[u]
\end{tikzcd}$$

Figure 2.2: Hierarchy of various types of pushdown relations. An arrow denotes a proper inclusion, the equalities are denoted by =, and classes that are not connected through a sequence of arrows are incomparable with respect to inclusion.

Chapter 3

Weighted Restarting Automata

We already mentioned in Section 1 that restarting automata have been introduced as a formal model for the linguistic technique of analysis by reduction, which can be used to check the correctness of natural language sentences. In order to study quantitative aspects of restarting automata, we introduce the concept of a *weighted restarting automaton*. Such an automaton is defined by a pair (M, ω), where M is a restarting automaton on some input alphabet Σ, and ω is a *weight function* that assigns an element of a given semiring S to each transition of M. This chapter comprises three sections. In the first section, we recall the definition and basic properties of restarting automata and present some examples. The second section briefly restates the notions of monoid and semiring that we will use below. In Section three, we introduce the notion of weighted restarting automaton that is the central notion of this work, and show some examples of weighted restarting automata.

3.1 Restarting Automata

In this section, we first recall the definition and basic properties of restarting automata and present some examples of restarting automata. Further, we introduce some variants of restarting automata and give some results on their expressive power from earlier works.

3.1.1 Definitions and Examples

As described above restarting automata are language accepting devices that consist of a finite-state control and a read/write window that works on a flexible tape that is delimited by end markers. Formally, a restarting automaton is described as follows.

Definition 3.1.1. *A* restarting automaton *(*RRWW*-automaton for short) is a one-tape machine that is defined as an 8-tuple* $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$*,* *where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, the symbols $¢, \$ \notin \Gamma$ serve as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \in \mathbb{N}_+$ is the size of the* read/write window*, and*

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

is the transition function*. Here $\mathcal{PC}^{(k)}$ is the set of* possible contents *of the read/write window of M, where $\mathcal{PC}^{(0)} = \{\lambda\}$ and, for $i \geq 1$,*

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

and

$$\Gamma^{\leq i} = \bigcup_{j=0}^{i} \Gamma^j, \quad \textit{and} \quad \mathcal{PC}^{\leq(k-1)} = \bigcup_{i=0}^{k-1} \mathcal{PC}^{(i)}.$$

The function δ describes four different types of transition steps:

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

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

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

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

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

A *configuration* of M is a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here $q \in Q$ represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the read/write window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0¢w\$$, where $w \in \Gamma^*$; if $w \in \Sigma^*$, then $q_0¢w\$$ is an *initial configuration*. Thus, initial configurations are a particular type of restarting configurations. Further, we use Accept to denote the *accepting configurations*, which are those configurations that M reaches by executing an accept instruction. A configuration of the form $\alpha q \beta$ such that $\delta(q, \beta_1) = \emptyset$, where β_1 is the current content of the read/write 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 its reflexive and transitive closure $\vdash_M^*$ is the *computation relation* of M [1].

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

Any finite computation of a restarting automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the head moves along the tape performing move-right operations and a rewrite operation until a restart operation is performed and thus, a new restarting configuration is reached. If no further restart operation is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. We require that M performs *exactly one* rewrite operation during any cycle – thus each new phase starts on a shorter word than the previous one. During a tail at most one rewrite operation may be executed. 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. Accordingly, an accepting computation of M consists of a finite sequence of cycles that is followed by an accepting tail computation. It can be described as

$$q_0¢w\$ \vdash_M^c q_0¢w_1\$ \vdash_M^c q_0¢w_2\$ \vdash_M^c \ldots \vdash_M^c q_0¢w_n\$ \vdash_M^* \mathsf{Accept},$$

where $w \in \Sigma^*$ is the input word of M, and $w_1, w_2, \ldots, w_n$ are the shorter tape contents that occur in the restarting configurations during this computation.

[1]We often use the single-step computation relation to express the type of the step that is performed. For example, $\vdash_{\mathsf{MVR}}$ denotes a move-right step.

An input $w \in \Sigma^*$ is accepted by M, if there exists a computation of M which starts with the initial configuration $q_0¢w\$$, and which finally ends with executing an accept instruction. The language $L(M)$ accepted by M is the set that consists of all input strings that are accepted by M, that is,

$$L(M) = \{\, w \in \Sigma^* \mid q_0¢w\$ \vdash_M^* \mathsf{Accept} \,\}.$$

Now we restate some basic facts about computations of restarting automata. As mentioned in Section 1, a restarting automaton verifies the syntactical correctness of sentences through sequences of local simplifications. During the process of simplifying a given sentence, each step preserves the correctness or incorrectness of the sentence. Formally, error and correctness preserving properties are described as follows (see, e.g., [Ott06]).

Proposition 3.1.1 (Error Preserving Property)**.**
Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a restarting automaton, and let u, v be words over its input alphabet Σ. If $q_0¢u\$ \vdash_M^{c^*} q_0¢v\$$ holds and $u \notin L(M)$, then $v \notin L(M)$, either.

Proposition 3.1.2 (Correctness Preserving Property)**.**
Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a restarting automaton, and let u, v be words over its input alphabet Σ. If $q_0¢u\$ \vdash_M^{c^*} q_0¢v\$$ is an initial segment of an accepting computation of M, then $v \in L(M)$.

Now we continue with a simple example of a restarting automaton taken from [Ott06].

Example 3.1.1. *Let $M_1 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be the* RRWW*-automaton that is defined by taking $Q = \{q_0, q_c, q_d, q_e\}$, $\Gamma = \Sigma = \{a, b, c, d\}$, and $k = 3$, where δ contains the following transitions:*

$$\begin{array}{llcllllcl}
t_1: & (q_0, ¢c\$) & \to & \mathsf{Accept}, & t_{10}: & (q_d, bbb) & \to & (q_d, \mathsf{MVR}),\\
t_2: & (q_0, ¢d\$) & \to & \mathsf{Accept}, & t_{11}: & (q_d, bbd) & \to & (q_d, \mathsf{MVR}),\\
t_3: & (q_0, ¢ab) & \to & (q_0, \mathsf{MVR}), & t_{12}: & (q_d, bd\$) & \to & (q_d, \mathsf{MVR}),\\
t_4: & (q_0, ¢aa) & \to & (q_0, \mathsf{MVR}), & t_{13}: & (q_0, abc) & \to & (q_e, c),\\
t_5: & (q_0, aab) & \to & (q_0, \mathsf{MVR}), & t_{14}: & (q_0, abb) & \to & (q_c, b),\\
t_6: & (q_0, aaa) & \to & (q_0, \mathsf{MVR}), & t_{15}: & (q_0, abb) & \to & (q_d, \lambda),\\
t_7: & (q_c, bbb) & \to & (q_c, \mathsf{MVR}), & t_{16}: & (q_c, c\$) & \to & \mathsf{Restart},\\
t_8: & (q_c, bbc) & \to & (q_c, \mathsf{MVR}), & t_{17}: & (q_d, d\$) & \to & \mathsf{Restart},\\
t_9: & (q_c, bc\$) & \to & (q_c, \mathsf{MVR}), & t_{18}: & (q_e, \$) & \to & \mathsf{Restart}.
\end{array}$$

For example, M_1 can execute the following computations on the input $aabbc$:

$$q_0¢aabbc\$ \vdash_{M_1} ¢q_0aabbc\$ \vdash_{M_1} ¢aq_0abbc\$ \vdash_{M_1} \left\{ \begin{array}{l} ¢aq_dc\$, \\ ¢abq_cc\$. \end{array} \right.$$

The configuration $¢aq_dc\$$ does not admit any transition step anymore, that is, M_1 halts without accepting. However, from the configuration $¢abq_cc\$$, M_1 can continue as follows:

$$¢abq_cc\$ \vdash_{M_1} q_0¢abc\$ \vdash_{M_1} ¢q_0abc\$ \vdash_{M_1} ¢cq_e\$ \vdash_{M_1} q_0¢c\$ \vdash_{M_1} \mathsf{Accept}.$$

Accordingly, M_1 accepts on input aabbc. It is easily seen that the language $L(M_1)$ that is accepted by M_1 is

$$L(M_1) = \{ a^nb^nc \mid n \geq 0 \} \cup \{ a^nb^{2n}d \mid n \geq 0 \}.$$

Actually, the language $L(M_1)$ is the context-free language that is presented in Example 2.1.2, where it is accepted by the PDA P_1. In Section 3.1.2 we will give some characterizations of CFL in terms of restarting automata. By the following lemma we can obtain a simple way to describe the behaviour of restarting automata.

Lemma 3.1.1 ([Ott06]). *An* RRWW-*automaton is equivalent to an* RRWW-*automaton that makes an accept or restart step only when it sees the right border marker \$ in its read/write window.*

This lemma indicates that in each cycle and in the tail of each computation, the read/write window moves all the way to the right before a restart is executed, or before the automaton halts and accepts. Therefore, for an RRWW-automaton, each cycle (or the tail of a computation) that contains a rewrite step can be divided into three parts. The first one consists of the move-right steps that are performed before a rewrite step is made, and the second one is the rewrite step that is executed in the current cycle (or tail computation). The third part comprises the move-right steps that are performed before the automaton restarts (or halts). Accordingly, the transition function of an RRWW-automaton can also be described through a sequence of so-called *meta-instructions* [NO01] of the form $(¢ \cdot E_1, u \to v, E_2 \cdot \$)$, where E_1 and E_2 are regular languages, called the *regular constraints* of this instruction, and u and v are strings such that $|v| < |u|$. The rule $u \to v$ stands for a rewrite step of the RRWW-automaton considered. On trying to execute this meta-instruction this RRWW-automaton will get stuck (and so reject) starting from the configuration $q_0¢w\$$, if w does not admit a factorization of the form $w = w_1uw_2$ such that $w_1 \in E_1$, $w_2 \in E_2$. On the other hand, if w does have a factorization of this form, then one such factorization is chosen nondeterministically, and $q_0¢w\$$ is transformed into $q_0¢w_1vw_2\$$. In order to describe the tail of an accepting computation we use meta-instructions of the form $(¢ \cdot E \cdot \$, \mathsf{Accept})$, where the strings from the regular language E are accepted by the RRWW-automaton in tail computations. Now we illustrate this concept by describing the RRWW-automaton from Example 3.1.1 by meta-instructions.

Example 3.1.2. *Let M_1 be the* RRWW*-automaton on the input alphabet $\{a, b, c, d\}$ and without auxiliary symbols that is described by the following sequence of meta-instructions:*

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

The meta-instruction (1) *describes a cycle of M_1, in which M_1 first moves right across the prefix a^n, then it deletes the factor ab and performs move-right steps on b-symbols, and finally it restarts on seeing the suffix c\$. The meta-instruction* (2) *indicates that M_1 removes the factor abb, which means that M_1 guesses that the suffix of the input word is a d-symbol, then it moves to the right end of the tape to verify its guess. Further, it is easily seen that the meta-instructions* (3) *and* (4) *correspond to the accept transitions t_1 and t_2 of the transition function δ in Example 3.1.1, respectively. Hence, the set of meta-instructions above can be transformed into the transition function δ in Example 3.1.1, that is, $L(M_1) = \{\, a^n b^n c \mid n \geq 0 \,\} \cup \{\, a^n b^{2n} d \mid n \geq 0 \,\}$.*

This way of describing an RRWW-automaton corresponds to the characterization of the class $\mathcal{L}(\mathsf{RRWW})$ by certain infinite prefix-rewriting systems as given in [NO00a]. We close this section with the following definition.

Definition 3.1.2. *Let $M_1 = (Q_1, \Sigma_1, \Gamma_1, ¢, \$, q_0, k, \delta_1)$ be a restarting automaton. A restarting automaton $M_2 = (Q_2, \Sigma_2, \Gamma_2, ¢, \$, q_0, k, \delta_2)$ is a* subautomaton *of M_1 if and only if*

- $Q_2 \subseteq Q_1$,
- $\Sigma_2 \subseteq \Sigma_1$,
- $\Gamma_2 \subseteq \Gamma_1$,
- *and the transition function δ_2 is the restriction of δ_1 to*

$$Q_2 \times \mathcal{PC}^{(k)} \to \mathbb{P}_{\text{fin}}(Q_2 \times (\{\mathsf{MVR}\} \cup \mathcal{PC}^{\leq(k-1)}) \cup \{\mathsf{Restart}, \mathsf{Accept}\}),$$

where $\mathcal{PC}^{(k)}$ is the set of possible contents *of the read/write window of M_2, $\mathcal{PC}^{(0)} = \{\lambda\}$ and, for $i \geq 1$,*

$$\mathcal{PC}^{(i)} = (¢ \cdot \Gamma_2^{i-1}) \cup \Gamma_2^i \cup (\Gamma_2^{\leq i-1} \cdot \$) \cup (¢ \cdot \Gamma_2^{\leq i-2} \cdot \$),$$

and

$$\Gamma_2^{\leq i} = \bigcup_{j=0}^{i} \Gamma_2^j, \quad \textit{and} \quad \mathcal{PC}^{\leq(k-1)} = \bigcup_{i=0}^{k-1} \mathcal{PC}^{(i)}.$$

3.1.2 Some Variants of Restarting Automata

In this section we introduce some restricted types of restarting automata that we use in the forthcoming chapters. A restarting automaton is called an RWW-automaton if it makes a restart immediately after performing a rewrite operation. In particular, this means that it cannot perform a rewrite step during the tail of a computation. An RRWW-automaton is called an RRW-automaton if its tape alphabet Γ coincides with its input alphabet Σ, that is, if no auxiliary symbols are available. Further, for a word $u = a_1a_2 \dots a_n \in \Sigma^*$, a *scattered subword* of u is a word $v = a_{i_1}a_{i_2} \dots a_{i_l}$, where $1 \leq i_1 < i_2 < \dots < i_l \leq n$, and $|v| < |u|$. An RRWW-automaton is an RR-automaton if it is an RRW-automaton such that, for each rewrite transition $(q, u) \rightarrow (q', v)$, v is a scattered subword of u. Actually, the restarting automaton M_1 given in Example 3.1.1 is an RR-automaton. Analogously, we obtain the RW-automaton and the R-automaton from the RWW-automaton. For a type X of restarting automata, let $\mathcal{L}(\mathsf{X})$ denote the class of languages that are accepted by the restarting automata of type X. An overview on various types of restarting automata is given by [Ott06]. We summarize the inclusion relations between the classes of languages that are computed by the various types of restarting automata in Figure 3.1. Note that in Figure 3.1, $\mathcal{L}(\text{det-RWW}) = \mathcal{L}(\text{det-RRWW})$, while it is still open whether or not the inclusion $\mathcal{L}(\mathsf{RWW}) \subseteq \mathcal{L}(\mathsf{RRWW})$ is proper, and all other inclusions are proper. In addition, note that it is just a rough overview, and some details on the inclusion relations between these language classes are not shown in this figure. For example, the unconnected classes $\mathcal{L}(\mathsf{RW})$ and $\mathcal{L}(\mathsf{RR})$ are incomparable with respect to inclusion, and the inclusion relation between the unconnected classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRW})$ is unknown. Further details on these inclusion relations can be found in [Ott06].

Concerning the class of Church-Rosser languages the following characterization has been obtained.

Proposition 3.1.3 ([NO00b]). $\mathsf{CRL} = \mathcal{L}(\text{det-RWW}) = \mathcal{L}(\text{det-RRWW})$.

Further, some closure properties of the language classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ have been shown in earlier works.

Proposition 3.1.4 ([JLNO04]). *The language classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ are closed under union, concatenation, and reversal, but not under projection.*

Concerning the class of growing context-sensitive languages we give the following inclusion results.

Proposition 3.1.5 ([JLNO04]). $\mathsf{GCSL} \subsetneq \mathcal{L}(\mathsf{RWW}) \subseteq \mathcal{L}(\mathsf{RRWW})$.

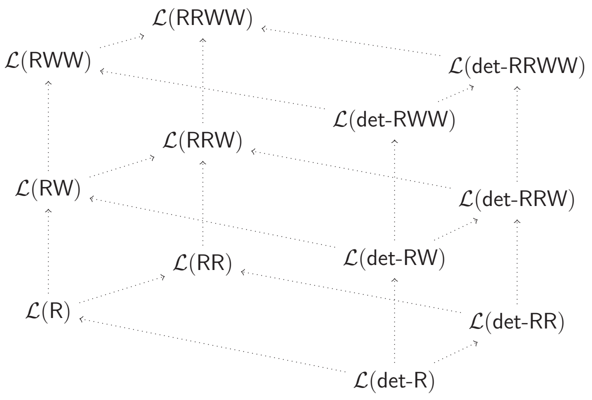

Figure 3.1: Hierarchy of classes of languages that are computed by the various types of restarting automata. An arrow denotes an inclusion.

Finally, we give an upper bound for the expressive power of RRWW-automata. On an input word of length n, a restarting automaton can execute at most n cycles. Thus, the following result can be established, where NP and P denote the well-known complexity classes, and DCSL denotes the class of *deterministic context-sensitive languages*. Note that the properness of the inclusion DCSL $\subseteq$ CSL is still open.

Proposition 3.1.6 ([Ott06])**.**

$$\begin{array}{lll} (1) & \mathcal{L}(\mathsf{RRWW}) & \subseteq \ \mathsf{NP} \cap \mathsf{CSL}. \\ (2) & \mathcal{L}(\mathsf{det\text{-}RRWW}) & \subseteq \ \mathsf{P} \cap \mathsf{DCSL}. \end{array}$$

Monotone Restarting Automata

Next we restate the notion of *monotonicity* for restarting automata that is introduced in [JMPV97]. Let $C = \alpha q \beta$ be a *rewrite configuration* of an RRWW-automaton M, that is, a configuration in which a rewrite step can be applied. Then $|\beta|$ is called the *right distance* of C, which is denoted by $D_r(C)$. A *sequence of rewrite configurations* $S = (C_1, C_2, \ldots, C_n)$ is called *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$, that is, if the distance of the place of rewriting to the right end of the tape does not increase from one rewrite step to the next. A *computation* of an RRWW-automaton M is called *monotone* if the sequence of rewrite configurations that is obtained from the cycles of that computation is monotone. Observe that here the rewrite configuration

$$\begin{tikzcd}
\mathsf{CFL} \arrow[r, equal] & \mathcal{L}(\mathsf{mon\text{-}RWW}) \arrow[r, equal] & \mathcal{L}(\mathsf{mon\text{-}RRWW}) \\
& \mathcal{L}(\mathsf{mon\text{-}RW}) \arrow[u] \arrow[r] & \mathcal{L}(\mathsf{mon\text{-}RRW}) \arrow[u] \\
& \mathcal{L}(\mathsf{mon\text{-}R}) \arrow[u] \arrow[r] & \mathcal{L}(\mathsf{mon\text{-}RR}) \arrow[u] \\
\mathsf{DCFL} \arrow[r, equal] & \mathcal{L}(\mathsf{det\text{-}mon\text{-}R(R)(W)(W)}) \arrow[u] &
\end{tikzcd}$$

Figure 3.2: Hierarchy of classes of languages that are computed by the various types of monotone restarting automata. An arrow denotes a proper inclusion, and the equalities are denoted by =.

is not taken into account that corresponds to the possible rewrite step that is executed in the tail of the computation considered. Finally, an RRWW-automaton M is called *monotone* if all its computations that start with an initial configuration are monotone. We use the prefix mon- to denote monotone types of restarting automata.

It is well-known that the classes $\mathcal{L}$(mon-RWW) and $\mathcal{L}$(mon-RRWW) coincide with the class CFL of context-free languages.

Proposition 3.1.7 ([JMPV99]). CFL $= \mathcal{L}$(mon-RRWW) $= \mathcal{L}$(mon-RWW).

For the monotone restarting automata of other types, the inclusion relations are the same as in the nonmonotone case, and the classes of languages that are accepted by these restarting automata are strictly included in the class CFL. In particular, the class DCFL of deterministic context-free languages is a proper subset of the class of languages that are accepted by mon-R-automata, i.e., DCFL $\subsetneq \mathcal{L}$(mon-R).

Now we come to deterministic monotone restarting automata. In [JMPV97] it is proved that the class $\mathcal{L}$(det-mon-R) coincide with the class DCFL . Further, it is shown in [JMPV99] that the class DCFL corresponds to the class $\mathcal{L}$(det-mon-RRWW). This yields the following result.

Proposition 3.1.8 ([JMPV97, JMPV99]). *For all types* X $\in$ {R, RR, RW RRW, RWW, RRWW}*,* $\mathcal{L}$(det-mon-X) $=$ DCFL.

Finally, from the results obtained in earlier works (e.g., [JMPV97, MPJV97, JMPV98, JMPV99]), we summarize the inclusion relations between the classes of languages that are accepted by monotone restarting automata of various types in the diagram in Figure 3.2.

Non-Forgetting Restarting Automata

The purpose of this section is to introduce another variant of restarting automata, so-called *non-forgetting restarting automata*, and to recall some facts about the expressive power of them. A restarting automaton M is called *non-forgetting*, if its restart steps are combined with a change of state just like the move-right and rewrite operations [MS04]. The prefix nf- is used to denote types of non-forgetting restarting automata.

Here we will restrict our attention to monotone non-forgetting restarting automata. From these earlier works such as [MS04, MO06, MO11], we know that the inclusion relations between the classes of languages accepted by nondeterministic monotone non-forgetting restarting automata are the same as in the forgetting case, and that also the class CFL coincides with the classes $\mathcal{L}(\mathsf{mon\text{-}nf\text{-}RWW})$ and $\mathcal{L}(\mathsf{mon\text{-}nf\text{-}RRWW})$, that is,

$$\mathcal{L}(\mathsf{mon\text{-}nf\text{-}RRWW}) = \mathcal{L}(\mathsf{mon\text{-}nf\text{-}RWW}) = \mathsf{CFL}.$$

In addition, it is easily seen that

$$\mathcal{L}(\mathsf{mon\text{-}R}) \subsetneq \mathcal{L}(\mathsf{mon\text{-}nf\text{-}R})$$

and

$$\mathcal{L}(\mathsf{mon\text{-}RW}) \subsetneq \mathcal{L}(\mathsf{mon\text{-}nf\text{-}RW}).$$

For the above proper inclusions, we consider the language

$$L(M_1) = \{\, a^n b^n c \mid n \geq 0 \,\} \cup \{\, a^n b^{2n} d \mid n \geq 0 \,\}$$

that is given in Example 3.1.1. It is well-known that $L(M_1)$ does not belong to the language class $\mathcal{L}(\mathsf{RW})$ (see, e.g., [JMPV99]), while it can be accepted by a mon-nf-R-automaton M that proceeds as follows. Given an input word w, M first guesses the suffix of w and stores this guess in its finite control. Then, M moves to the bound between a- and b-symbols, and performs a corresponding rewrite step according to its guess. For example, if M guesses that the suffix is c, then it removes the factor ab. As M is non-forgetting, after a restart step the information in the finite control is not lost. Finally, on seeing the suffix M can verify its guess.

However, the class DCFL can only be characterized by deterministic monotone non-forgetting restarting automata of the types with a single R, that is,

$$\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}R(W)(W)}) = \mathsf{DCFL},$$

and it is strictly included in the class of languages that are accepted by the automata of the types with double Rs [MO06], that is,

$$\mathsf{DCFL} \subsetneq \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RR}).$$

$$\begin{tikzcd}
 & \mathsf{CFL} \\
 & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRWW}) \arrow[u] \\
 & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRW}) \arrow[u] \\
 & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RR}) \arrow[u] \\
\mathsf{DCFL} \arrow[r, equal] & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}R(W)(W)}) \arrow[u]
\end{tikzcd}$$

Figure 3.3: Hierarchy of classes of languages that are computed by the various types of deterministic monotone non-forgetting restarting automata. An arrow denotes a proper inclusion, and the equalities are denoted by =.

Further, just like in the nondeterministic case, the proper inclusions

$$\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RR}) \subsetneq \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRW}) \subsetneq \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRWW})$$

have been shown in [MO06], and the class CFL strictly contains the class $\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRWW})$. Finally, we summarize inclusion relations between these above language classes in terms of deterministic monotone non-forgetting restarting automata in Figure 3.3.

Stateless Restarting Automata

We close this section with the notion of *stateless restarting automata*, which is almost the weakest variant of restarting automaton (see, e.g., [KMO10a, KMO10b]). A restarting automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is called stateless, if $Q = \{q_0\}$. Thus, M can simply be written as $M = (\Sigma, \Gamma, ¢, \$, k, \delta)$. We use the prefix stl- to denote stateless types of restarting automata. It is shown in [NO12] that stateless deterministic R-automata of window size one can only accept some regular languages, that is,

$$\mathcal{L}(\mathsf{stl\text{-}det\text{-}R}(1)) \subsetneq \mathsf{REG},$$

and that those of window size two can only accept some deterministic context-free languages, that is,

$$\mathcal{L}(\mathsf{stl\text{-}det\text{-}R}(2)) \subsetneq \mathsf{DCFL},$$

while deterministic stateless RWW-automata and monotone stateless RWW-automata are as powerful as the nonstateless automata of the corresponding types. We summarize the above inclusion results in Figure 3.4.

$$\begin{tikzcd}
\textsf{CRL} \arrow[r, equal] & \mathcal{L}(\textsf{stl-det-RWW}) & \textsf{CFL} \arrow[r, equal] & \mathcal{L}(\textsf{stl-mon-RWW}) \\
& \mathcal{L}(\textsf{stl-det-R}(2)) \arrow[u] \arrow[r] & \textsf{DCFL} \arrow[u] & \\
& \mathcal{L}(\textsf{stl-det-R}(1)) \arrow[u] \arrow[r] & \textsf{REG} \arrow[u] &
\end{tikzcd}$$

Figure 3.4: Hierarchy of classes of languages that are computed by the various types of stateless restarting automata. An arrow denotes a proper inclusion, and the equalities are denoted by $=$.

3.2 Monoids and Semirings

We already mentioned that the weight of a transition of a weighted restarting automaton is an element of some semiring. Thus, in this section we recall the notions of monoid and semiring and present some examples of them.

Definition 3.2.1 ([DK09]). *A* monoid *is defined as a triple $M = (M, \cdot, 1)$, where*

- *M is a non-empty set,*
- *$\cdot : M \times M \to M$ is an associative binary operation, that is, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in M$, and*
- *1 is a neutral element for $\cdot$, that is, $1 \cdot a = a \cdot 1 = a$ for all $a \in M$.*

The monoid M is called commutative *if $a \cdot b = b \cdot a$ holds for all $a, b \in M$. It is called* ordered *if there exists a partial order $\leq$ on M that is compatible with the operation $\cdot$, that is, if $a \leq b$, then $(a \cdot c) \leq (b \cdot c)$ and $(c \cdot a) \leq (c \cdot b)$ for all $a, b, c \in M$. Finally, it is called* linearly ordered *if it is ordered with respect to a linear order.*

Now we give some examples of monoids. Let $\mathbb{Z}$ be the set of all integers, let $\mathbb{Q}$ be the set of all rational numbers, and let $\mathbb{R}$ be the set of all real numbers. Obviously, $(\mathbb{N}, +, 0)$ and $(\mathbb{N}, \cdot, 1)$ are commutative monoids that are linearly ordered, while the commutative monoids $(\mathbb{Z}, \cdot, 1)$, $(\mathbb{Q}, \cdot, 1)$, and $(\mathbb{R}, \cdot, 1)$ are not ordered with respect to the natural order relation $\leq$, as in general, $a \leq b$ does not imply $a \cdot c \leq b \cdot c$. Further, let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ and $\overline{\mathbb{N}} = \mathbb{N} \cup \{-\infty\}$. Then $(\mathbb{N}_\infty, \min, \infty)$ and $(\overline{\mathbb{N}}, \max, -\infty)$ are commutative monoids that are linearly ordered. For some alphabet Σ, $(\Sigma^*, \cdot, \lambda)$ is a monoid, the *free monoid* generated by Σ. It is not commutative unless $|\Sigma| = 1$ holds. It is linearly ordered with respect to the length-lexicographical ordering (see, e.g., [BO93]). Further, $(\mathbb{P}(S), \cup, \emptyset)$, $(\mathbb{P}(S), \cap, S)$, and $(\mathbb{P}_{\text{fin}}(S), \cup, \emptyset)$ are commutative monoids for any set S. Finally, $(\mathbb{P}(\Sigma^*), \cdot, \{\lambda\})$ and $(\mathbb{P}_{\text{fin}}(\Sigma^*), \cdot, \{\lambda\})$ are monoids, where for some sets U and V, $U \cdot V = \{\, u \cdot v \mid u \in U, v \in V \,\}$ denotes the extension of the concatenation operation from words to languages. These monoids are ordered with respect to the inclusion relation, which is not a linear order.

Definition 3.2.2 ([DK09]). *A* semiring $S = (S, +, \cdot, 0, 1)$ *is a non-empty set S together with two binary operations $+ : S \times S \to S$ and $\cdot : S \times S \to S$ and two elements $0, 1 \in S$ such that the following conditions are satisfied:*

1. *$(S, +, 0)$ is a commutative monoid,*
2. *$(S, \cdot, 1)$ is a monoid,*
3. *the distributive laws*

   $$(a + b) \cdot c = (a \cdot c) + (b \cdot c) \text{ and } c \cdot (a + b) = (c \cdot a) + (c \cdot b)$$

   hold for all $a, b, c \in S$, and
4. *$0 \cdot a = a \cdot 0 = 0$ holds for all $a \in S$.*

The semiring S is called commutative *if $(S, \cdot, 1)$ is a commutative monoid. It is* (linearly) ordered *with respect to an order $\leq$, if $(S, +, 0)$ is a (linearly) ordered monoid with respect to $\leq$ and if multiplication by elements $s \geq 0$ preserves the order, that is, if $s \geq 0$ and $a \leq b$, then $(s \cdot a) \leq (s \cdot b)$ and $(a \cdot s) \leq (b \cdot s)$.*

Obviously, $(\mathbb{N}, +, \cdot, 0, 1)$ and $(\mathbb{R}, +, \cdot, 0, 1)$ as well as $(\mathbb{B}, \vee, \wedge, 0, 1)$, where $\mathbb{B} = \{0, 1\}$, are commutative semirings that are linearly ordered with respect to the natural order. For $\mathbb{Z}_\infty = \mathbb{Z} \cup \{\infty\}$ and $\overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty\}$, $(\mathbb{N}_\infty, \min, +, \infty, 0)$ and $(\mathbb{Z}_\infty, \min, +, \infty, 0)$ are the *tropical* or *min-plus semirings*, and $(\overline{\mathbb{N}}, \max, +, -\infty, 0)$ and $(\overline{\mathbb{Z}}, \max, +, -\infty, 0)$ are the *arctic* or *max-plus semirings*, which are also commutative and linearly ordered under the natural order. Further,

$$(\mathbb{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\lambda\}) \text{ and } (\mathbb{P}_{\text{fin}}(\Sigma^*), \cup, \cdot, \emptyset, \{\lambda\})$$

are semirings that are not commutative unless $|\Sigma| = 1$, and the same holds for

$$(\mathsf{REG}(\Sigma), \cup, \cdot, \emptyset, \{\lambda\}) \text{ and } (\mathsf{CFL}(\Sigma), \cup, \cdot, \emptyset, \{\lambda\}),$$

where $\mathsf{REG}(\Sigma)$ and $\mathsf{CFL}(\Sigma)$ denote the classes of regular and context-free languages over Σ. These semirings are ordered with respect to the inclusion relation. More information on and further examples of semirings can be found in [Gol99]. Applications of semirings can be found in many areas such as IT-security (see, e.g., [AAM07, Mon02]), and network analysis (see, e.g., [CB14, PB16]). We complete this subsection by restating the notions of weighted automaton and recognizable function in short.

Definition 3.2.3 ([Sb61]). *Let $S = (S, +, \cdot, 0, 1)$ be a semiring, and let Σ be a finite alphabet. A* weighted automaton *is given through a four-tuple $A = (Q, \mathrm{in}, \omega, \mathrm{out})$, where*

- *Q is a finite set of states,*
- *$\mathrm{in} : Q \to S$ assigns an* entrance *weight to each state,*
- *$\mathrm{out} : Q \to S$ assigns an* exit *weight to each state, and*
- *$\omega : Q \times \Sigma \times Q \to S$ is a weight function that assigns a weight to each possible transition.*

A path *in A is any sequence*

$$P = (q_0, a_1, q_1, a_2, q_2, \ldots, a_n, q_n),$$

where $q_0, q_1, \ldots, q_n \in Q$ and $a_1, a_2, \ldots, a_n \in \Sigma$, and the word $a_1 a_2 \ldots a_n \in \Sigma^*$ is called its label*. The* run weight *of P is the product*

$$\mathrm{rweight}(P) = \prod_{0 \leq i < n} \omega(q_i, a_{i+1}, q_{i+1}),$$

where $\mathrm{rweight}((q_0)) = 1$ is taken. Note that if $n = 0$, then $\mathrm{rweight}(P) = 1$. Further, the weight *of P is defined as*

$$\omega(P) = \mathrm{in}(q_0) \cdot \mathrm{rweight}(P) \cdot \mathrm{out}(q_n).$$

Finally, let $\mathrm{Path}(w)$ denote the set of all paths in A that have label w. Then the behavior *of A is the function $||A|| : \Sigma^* \to S$ that is defined by*

$$||A||(w) = \sum_{P \in \mathrm{Path}(w)} \omega(P)$$

for all $w \in \Sigma^*$. The set of recognizable functions *over S and Σ is the set $S_{\mathrm{REC}}\langle\langle \Sigma^* \rangle\rangle$ of all functions that are the behavior of some weighted automaton over S.*

More information on these notions can be found in [DKV09].

3.3 Weighted Restarting Automata

The aim of this section is to introduce the notion of *weighted restarting automaton* that is the central notion of this work.

Just as finite automata, given an input word $w \in \Sigma^*$, a restarting automaton M either accepts or rejects. Therefore, such an automaton can be seen as computing a Boolean function. But in case of acceptance, one may be interested in some quantitative aspects of a restarting automaton, such as the *number* of accepting computations of M on input w, or the *least number* of steps (or cycles) in such an accepting computation, or the *minimal number* of auxiliary symbols that are used during an accepting computation.

In the 1960s the notion of *weighted finite automaton* was introduced in [Sb61]. Such an automaton consists of a finite automaton A and a weight function ω that associates elements of a semiring S to the transitions of A. Now following the same fundamental idea, we define the notion of weighted restarting automaton [OW16], in order to answer quantitative questions for a restarting automaton.

Definition 3.3.1.

(a) *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a restarting automaton. A* weight function *ω from the transitions of M into a semiring $S = (S, +, \cdot, 0, 1)$ is a function $\omega : \delta \to S$, that is, ω assigns an element of S as a* weight *to each transition of M.*

(b) *A* weighted restarting automaton *of type* X *(*wX*-automaton for short) is defined as a pair $\mathcal{M} = (M, \omega)$, where M is a restarting automaton of type* X*, and ω is a weight function from the transitions of M into a semiring S.*

(c) *Let $\mathcal{M} = (M, \omega)$ be a weighted restarting automaton, where ω is a weight function from the transitions of M into the semiring $S = (S, +, \cdot, 0, 1)$. If c_1 and c_2 are configurations of M such that $c_1 \vdash_M c_2$ holds, then there exists a transition $t \in \delta$ such that c_2 is obtained from c_1 by performing the transition t. By $\omega(t)$ we denote the weight that is associated with this computational step of M.*

If the transitions $t_1, t_2, \ldots, t_n$ are used during a computation C in this order, then the weight *of this computation C is defined as $\omega(C) = \omega(t_1) \cdot \omega(t_2) \cdot \cdots \cdot \omega(t_n) \in S$. Finally, for each input word $w \in \Sigma^*$, let $C_M(w)$ be the set of all accepting computations of M on input w. Then*

$$f_\omega^M(w) = \sum_{C \in C_M(w)} \omega(C)$$

is the element of S that is associated to w by $\mathcal{M}$, that is, f_ω^M is a function from Σ^* into S.

Observe that each computation C of M is of finite length, and so $\omega(C)$ is defined as a finite product in S. Further, for each $w \in \Sigma^*$, the set $C_M(w)$ of accepting computations of M on input w is also finite, which implies that

$f^M_\omega(w)$ is obtained as a finite sum in S. These observations imply that indeed f^M_ω is a well defined function from Σ^* into S. If $w \notin L(M)$, then $C_M(w)$ is empty, which means that $f^M_\omega(w) = 0$ holds.

We close this section with some examples of weighted restarting automata. The first example is obtained from the RR-automaton M_1 of Example 3.1.1 by combining it with different weight functions. In the following examples, we will see that by using different semirings and weight functions, various quantitative aspects of M_1 can be studied.

Example 3.3.1. *Let $M_1 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be the* RR*-automaton from Example 3.1.1 that accepts the language $L(M_1) = L_1 = \{\, a^n b^n c \mid n \geq 0 \,\} \cup \{\, a^n b^{2n} d \mid n \geq 0 \,\}$.*

(a) *Let $\mathbb{B} = (\mathbb{B}, \vee, \wedge, 0, 1)$ be the Boolean semiring, and let ω_1 be the weight function that assigns weight 1 to each transition of M_1. Then $\omega_1(C) = 1$ for each accepting computation C of M_1, and*

$$f^{M_1}_{\omega_1}(w) = \left\{ \begin{array}{ll} 1, & \textit{for } w \in L_1 \\ 0, & \textit{for } w \notin L_1 \end{array} \right\},$$

that is, $f^{M_1}_{\omega_1} : \{a, b, c, d\}^* \to \mathbb{B}$ is simply the characteristic function of the language L_1.

(b) *Let $\mathbb{N}_\infty = (\mathbb{N}_\infty, \min, +, \infty, 0)$ be the tropical semiring, and let ω_2 be the weight function that assigns weight 1 to each restart transition of M_1, and that assigns weight 0 to all other transitions of M_1. Then $\omega_2(C) = |C|_{\mathrm{rs}}$ for each computation C of M_1, where $|C|_{\mathrm{rs}}$ denotes the number of restart steps in C. Although M_1 is nondeterministic (see its rewrite transitions), it has only a single accepting computation $C(w)$ for each word $w \in L_1$. Accordingly,*

$$f^{M_1}_{\omega_2}(w) = \left\{ \begin{array}{ll} |C(w)|_{\mathrm{rs}}, & \textit{for } w \in L_1 \\ \infty, & \textit{for } w \notin L_1 \end{array} \right\}.$$

In the same way, we can also define a weight function ω_2' such that the function $f^{M_1}_{\omega_2'}(w)$ determines the number of move-right steps or rewrite steps during the computation on an input w.

(c) *Let $(\mathbb{P}_{\mathrm{fin}}(\Delta^*), \cup, \cdot, \emptyset, \{\lambda\})$ be the semiring of finite languages over the finite alphabet $\Delta = \{c, d\}$, and let ω_3 be the weight function that assigns the set $\{c\}$ to the transitions $t_{16} : (q_c, c\$) \to \mathsf{Restart}$ and $t_{18} : (q_e, \$) \to \mathsf{Restart}$, that assigns the set $\{dd\}$ to $t_{17} : (q_d, d\$) \to \mathsf{Restart}$, and that* *assigns the set $\{\lambda\}$ to all other transitions. Accordingly,*

$$f_{\omega_3}^{M_1}(w) = \left\{ \begin{array}{ll} \{c^n\}, & \textit{for } w = a^n b^n c \\ \{d^{2n}\}, & \textit{for } w = a^n b^{2n} d \\ \emptyset, & \textit{otherwise} \end{array} \right\}.$$

Actually, the set of pairs $(a^n b^n c, c^n)$ and $(a^n b^{2n} d, d^{2n})$ can be viewed as the relation that is computated by the weighted restarting automaton (M_1, ω_3). Hence, using weight functions of this form, the restarting transducers that are introduced in [Hun13] can be simulated. In Chapter 5 we will describe the classes of relations that are computated by restarting transducers and weighted restarting automata in detail.

(d) *Let $(\overline{\mathbb{N}}, \max, +, -\infty, 0)$ be the arctic semiring. Let ω_4 be the weight function that assigns weight 2 to the transition $t_{13} : (q_0, abc) \to (q_e, c)$ and the transition $t_{14} : (q_0, abb) \to (q_c, b)$, weight 3 to the transition $t_{15} : (q_0, abb) \to (q_d, \lambda)$, and weight 0 to all other transitions. Then $\omega_4(C)$ is the number of symbols that are deleted during the computation C, and accordingly,*

$$f_{\omega_4}^{M_1}(w) = \begin{cases} 2n, & \textit{for } w = a^n b^n c, \\ 3n, & \textit{for } w = a^n b^{2n} d, \\ -\infty, & \textit{for } w \notin L_1. \end{cases}$$

Finally, we give an example of a nondeterministic wRWW-automaton.

Example 3.3.2. *Let $L_2 = \{\, w_1 w_1^R w_2 w_2^R \dots w_n w_n^R \mid n \geq 1, w_i \in \{a,b\}^+, |w_i| \equiv 0 \mod 2, i = 1, 2, \dots, n \,\}$, and let $M_2 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an* RWW-*automaton, where $Q = \{q_0, q_r\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \#\} \cup \{\, [c,d] \mid c, d \in \Sigma \,\}$, $k = 4$, and the transition function δ is defined as follows, where $c, d, e, f, g, h, x \in$*

Σ:

$$\begin{array}{rl}
(1) & (q_0, ¢cde) \to (q_r, ¢[c,d]e),\\
(2) & (q_0, ¢[c,d]ef) \to (q_r, ¢[c,d][e,f]),\\
(3) & (q_0, ¢[c,d]dc) \to (q_r, ¢\#),\\
(4) & (q_0, ¢\#cd) \to (q_r, ¢cd),\\
(5) & (q_0, ¢\#\$) \to \mathsf{Accept},\\
(6) & (q_0, ¢[c,d][e,f]g) \to (q_0, \mathsf{MVR}),\\
(7) & (q_0, ¢[c,d][e,f][g,h]) \to (q_0, \mathsf{MVR}),\\
(8) & (q_0, [c,d][e,f]gh) \to (q_r, [c,d][e,f][g,h]),\\
(9) & (q_0, [c,d][e,f]fe) \to (q_r, [c,d]\#),\\
(10) & (q_0, ¢[c,d]\#d) \to (q_0, \mathsf{MVR}),\\
(11) & (q_0, [e,f][c,d]\#d) \to (q_0, \mathsf{MVR}),\\
(12) & (q_0, [c,d]\#dc) \to (q_r, \#),\\
(13) & (q_0, ¢[c,d][e,f]\#) \to (q_0, \mathsf{MVR}),\\
(14) & (q_0, [c,d][e,f][g,h]x) \to (q_0, \mathsf{MVR}),\\
(15) & (q_0, [c,d][e,f][g,h]\#) \to (q_0, \mathsf{MVR}),\\
(16) & (q_r, z) \to \mathsf{Restart} \textit{ for all } z \in \Gamma^4 \cup \Gamma^{\leq 3} \cdot \{\$\}.
\end{array}$$

For example, on the input word $aabaabaaabba$ M_2 can execute the following computation:

$$\begin{array}{rll}
q_0¢aabaabaaabba\$ & \vdash^2_{(1,16)} & q_0¢[a,a]baabaaabba\$\\
& \vdash^2_{(2,16)} & q_0¢[a,a][b,a]abaaabba\$\\
& \vdash_{(6)} & ¢q_0[a,a][b,a]abaaabba\$\\
& \vdash^2_{(9,16)} & q_0¢[a,a]\#aaabba\$\\
& \vdash_{(10)} & ¢q_0[a,a]\#aaabba\$\\
& \vdash^2_{(12,16)} & q_0¢\#abba\$\\
& \vdash^2_{(4,16)} & q_0¢abba\$\\
& \vdash^2_{(1,16)} & q_0¢[a,b]ba\$\\
& \vdash^2_{(3,16)} & q_0¢\#\$\\
& \vdash_{(5)} & \mathsf{Accept}.
\end{array}$$

It is easily seen that $L(M_2) = L_2$ holds, and that on input $w \in \Sigma^+$, M_2 has an accepting computation for each factorization of w of the form $w = w_1w_1^Rw_2w_2^R \dots w_nw_n^R$ such that $w_i \in \Sigma^+$ and $|w_i| \equiv 0 \mod 2$ for all $i = 1, \dots, n$.

(a) *Let $\mathbb{N}_\infty = (\mathbb{N}_\infty, \min, +, \infty, 0)$ be the tropical semiring, and let ω_1 be the weight function that assigns weight 1 to each transition of the groups (3) and (9) of M_2, and that assigns weight 0 to all other transitions of M_2. Then, for each computation C of M_2, $\omega_1(C)$ is the number of times the symbol $\#$ is introduced during C, and hence, if C is an accepting computation on input w, then this is the number n of factors $w_iw_i^R$ in* *the factorization of w that is guessed in the course of this computation. Accordingly,*

$$f_{\omega_1}^{M_2}(w) = \left\{ \begin{array}{ll} \textit{minimal number of factors of } w, & \textit{for } w \in L_2 \\ \infty, & \textit{for } w \notin L_2 \end{array} \right\}.$$

(b) *Let $(\mathbb{P}_{\mathrm{fin}}(\Gamma^*), \cup, \cdot, \emptyset, \{\lambda\})$ be the semiring of finite languages over the finite alphabet $\Gamma = \{a, b, \#\}$, and let ω_2 be a weight function that assigns the set $\{cd\}$ to the transitions of group (1), $\{ef\}$ to the transitions of group (2), and $\{gh\}$ to the transitions of group (8), that assigns the set $\{\#\}$ to the transitions of the groups (3) and (9), and that assigns the set $\{\lambda\}$ to all other transitions. It can now be checked that $\omega_2(C) = \{aaba\#ab\#\}$ for the computation C on input $w = aabaabaaabba$ presented above. It follows that*

$$f_{\omega_2}^{M_2}(w) = \{\, w_1\# \ldots \#w_n\# \mid w = w_1 w_1^R \ldots w_n w_n^R \,\},$$

that is, $f_{\omega_2}^{M_2}(w)$ is the set of possible factorizations that witness that w belongs to L_2.

Chapter 4

Functions Computed by Weighted Restarting Automata

We already mentioned in Chapter 3 that each weighted restarting automaton represents a function from the set of words over its input alphabet into a semiring. The purpose of this chapter is to study the classes of functions that are induced by weighted restarting automata. This chapter consists of four sections, where the first one gives some basic definitions and some examples. Section two and three present the results on the syntactic and semantic properties of functions induced by weighted restarting automata such as growth rates and closure properties. Finally, this chapter is closed with a summary and some problems for future work.

4.1 Definitions and Examples

For a type X of restarting automaton, we are interested in the class of functions that are induced by wX-automata. Accordingly, we introduce the following notion.

Definition 4.1.1 ([OW16]). *For a type X of restarting automata, a finite alphabet Σ, and a semiring S, let $\mathbb{F}(\mathsf{X}, \Sigma, S)$ denote the set of all functions of the form $f_\omega^M : \Sigma^* \to S$, where M is a restarting automaton of type X with input alphabet Σ, and ω is a weight function from the transitions of M into the semiring S.*

In Section 3.3 some examples of functions induced by weighted restarting automaton have been presented. We have seen that weighted restarting automata can be used to express interesting quantitative aspects of computations and of languages of restarting automata. Here we give some further examples which involve some complex languages.

Example 4.1.1. *First, we will construct an* RRWW-*automaton that accepts all so-called* non-primitive *words [DHI+ 93] over some alphabet Σ. Recall that a word is called non-primitive if it can be written as $w = u^n$ for some word $u \in \Sigma^+$ and some integer $n \geq 2$. Let $M_1 = (Q, \Sigma, \Gamma, ¢, \$, q_0, 3, \delta)$ be the* RRWW-*automaton that executes the following computation given a word $w \in \Sigma^+$ as input:*

1. *Repeatedly replace a factor ab of length 2 by rewriting it into an auxiliay symbol of the form $[a, b, 1]$ for $a, b \in \Sigma$, proceeding from right to left. This process is ended by replacing the prefix $¢ab$ by the word $¢[a, b, 1]$. Thus, we have the initial computation*

$$q_0¢w\$ \vdash^{c^*}_{M_1} q_0¢[a, b, 1]w_1[a, b, 1]w_2 \dots [a, b, 1]w_n\$,$$

   where $n \geq 2$ and $w = abw_1abw_2 \dots abw_n$.

2. *Check whether all marked off blocks coincide, that is, $w_1 = w_2 = \dots = w_n$. This can be done by first rewriting $[a, b, 1]c$ into $[b, c, 2]$ from left to right, and then by rewriting $[b, c, 2]d$ into $[c, d, 1]$, where these two stages are repeated alternatingly. In the affirmative, halt and accept, and if a mismatch is found, halt and reject.*

It is easily seen that M_1 accepts all non-primitive words, that is,

$$L(M_1) = \{w \mid w = u^n \text{ for some } u \in \Sigma^+ \text{ and } n \geq 2\}.$$

Let $S = (\mathbb{N}, +, \cdot, 0, 1)$, and let $\omega_1 : \delta \to S$ be the weight function that assigns the weight 1 to all transitions of M_1. Then, given an input word of the form $w = u^n$, the weight of an accepting computation is 1, and the value of $f^{M_1}_{\omega_1}(w)$ coincides with the number of admissible factorizations of w. Obviously, if w is primitive, then $f^{M_1}_{\omega_1}(w) = 0$.

Note that it is still an open problem whether the complement of the language $L(M_1)$ is a context-free language. In the following example we present a wRRWW-automaton that on input a unary word a^n, yields the set of all non-trivial divisors of n.

Example 4.1.2. *Let $M_2 = (Q, \Sigma, \Gamma, ¢, \$, q_0, 3, \delta)$ be the* RRWW-*automaton that executes the following computation given a word $w = a^n$ as input:*

1. *Nondeterministically place a marker # on the tape by rewriting aa into #, proceeding from right to left. This process is repeated all the way and ends by replacing $¢aa$ by the word $¢\&$. Thus, we have the initial computation:*

$$q_0¢w\$ \vdash^{c^*}_{M_2} q_0¢\&a^{r_1}\#a^{r_2} \dots \#a^{r_k}\$,$$

   where $k \geq 2$ and $\Sigma^k_{i=1}(r_i + 2) = n$.

2. *Again check whether all marked off blocks coincide, that is, $r_1 = r_2 = \ldots = r_k$. Using the trick from the previous example, this can be done by first rewriting $\&a$ into $\&_1$ and $\#a$ into $\#_1$ from left to right, by rewriting $\&_1 a$ into $\&_2$ and $\#_1 a$ into $\#_2$, and finally by rewriting $\&_2 a$ into $\&_1$ and $\#_2 a$ into $\#_1$, where the latter two stages are repeated alternatingly. In the affirmative case, halt and accept, and in the negative case, halt and reject.*

It is easily seen that M_2 accepts on input a unary word a^n, if n has a factorization $n = p \cdot r$ for some $p, r \geq 2$, that is, $L(M_2) = \{a^n \mid n \text{ is not a prime}\}$. In fact, M_2 has an accepting computation for each such factorization.

If $A = \{a_1, a_2, \ldots, a_k\}$ and $B = \{b_1, b_2, \ldots, b_l\}$ are two sets of integers, i.e., $|A| = k$ and $|B| = l$, then the sum of A and B is defined as

$$A + B = \{\, a_i + b_j \mid 1 \leq i \leq k, 1 \leq j \leq l \,\}.$$

Further, it is easily seen that $S = (\mathbb{P}_{\text{fin}}(\mathbb{N}), \cup, +, \emptyset, \{0\})$ is a semiring, and let $\omega_2 : \delta \to S$ be a weight function that assigns the set $\{2\}$ to the transition that rewrites $¢aa$ into $¢\&$, the set $\{1\}$ to all rewrite transitions that write one of the symbol $\&_1$ and $\&_2$, and that assigns the set $\{0\}$ to all other transitions. Then the weight of an accepting computation for a^n is a set containing only one integer that is just the factor p of the chosen factorization $p \cdot r = n$. Hence, $f_{\omega_2}^{M_2}(a^n) = \{p \in \mathbb{N} \mid p \geq 2 \text{ and } \exists r \in \mathbb{N}, r \geq 2 : p \cdot r = n\}$.

We continue with the inclusion relation between the classes $S_{\text{REC}}\langle\langle\Sigma^*\rangle\rangle$ and $\mathbb{F}(\mathsf{X}, \Sigma, S)$. To do this we recall the following notion.

Definition 4.1.2 ([Kir09]). *For a commutative semiring $S = (S, +, \cdot, 0, 1)$, if $s + t \neq 0$ for all $s, t \in S$, then we say that S is* zero-sum free.

Proposition 4.1.1 ([OW16]). *For each semiring S, each alphabet Σ, and each type of restarting automaton X,*

$$S_{\text{REC}}\langle\langle\Sigma^*\rangle\rangle \subseteq \mathbb{F}(\mathsf{X}, \Sigma, S).$$

If S is a commutative semiring that is zero-sum free, then this inclusion is proper.

Proof. Let $A = (Q, \text{in}, \omega, \text{out})$ be a weighted automaton with input alphabet Σ over the semiring S. To prove the above inclusion, we first construct a wR-automaton $M = (Q', \Sigma, \Sigma, ¢, \$, q_0, 1, \delta)$ and a weight function ω' such that $||A||(w) = f_{\omega'}^{M}(w)$ holds for all $w \in \Sigma^*$. We define the wR-automaton (M, ω') by taking

- $Q' = Q \cup \{q_0\}$, where q_0 is a new state,

- and by defining δ and ω' as follows, where $p, q \in Q$ and $a \in \Sigma$:

$$\begin{array}{rlcllcl} t_{q_0,p}: & (q_0, ¢) & \to & (p, \mathsf{MVR}), & \omega'(t_{q_0,p}) & = & \mathrm{in}(p), \\ t_{q,a,p}: & (q, a) & \to & (p, \mathsf{MVR}), & \omega'(t_{q,a,p}) & = & \omega(q, a, p), \\ t_{q,\$}: & (q, \$) & \to & \mathsf{Accept}, & \omega'(t_{q,\$}) & = & \mathrm{out}(q). \end{array}$$

Then, for all $w \in \Sigma^*$, each path in A with label w corresponds to an accepting computation of M on input w. As each accepting computation of M begins with a transition of type $t_{q_0,p}$ and ends with a transition of type $t_{q,\$}$, it follows immediately that $||A||(w) = f_{\omega'}^M(w)$ holds.

If $S = (S, +, \cdot, 0, 1)$ is zero-sum free, then the support $\{ w \in \Sigma^* \mid ||A||(w) \neq 0 \}$ of each recognizable series $||A|| \in S_{\mathrm{REC}}\langle\langle\Sigma^*\rangle\rangle$ is a regular language (see [Kir09, Kir11]). As already R-automata accept non-regular languages (see, e.g., [Ott06]), it is clear that in this case the inclusion $S_{\mathrm{REC}}\langle\langle\Sigma^*\rangle\rangle \subseteq \mathbb{F}(\mathsf{X}, \Sigma, S)$ is strict. □

In a linearly ordered semiring S (see Section 3.2), the maximum and the minimum of a finite subset T of S can be defined. We close this section with the following definitions.

Definition 4.1.3 ([OW16]). *If $S = (S, +, \cdot, 0, 1)$ is a semiring that is ordered with respect to a linear order $\leq$, then*

$$\min(T) = a \in T \text{ such that } a \leq t \text{ for all } t \in T$$

and

$$\max(T) = b \in T \text{ such that } t \leq b \text{ for all } t \in T$$

for each finite non-empty subset T of S.

Definition 4.1.4 ([OW16]). *Let $S = (S, +, \cdot, 0, 1)$ be a linearly ordered semiring, let M be a restarting automaton of type X with input alphabet Σ, and let ω be a weight function that maps the transitions of M into S. As Σ^n is finite for all $n \geq 0$, we can extend the function $f_\omega^M : \Sigma^* \to S$ to a function $\hat{f}_\omega^M : \mathbb{N} \to S$ as follows:*

$$\hat{f}_\omega^M(n) = \max\{ f_\omega^M(w) \mid w \in \Sigma^*, |w| = n \}.$$

By $\hat{\mathbb{F}}(\mathsf{X}, \Sigma, S)$ we denote the set of all functions of the form $\hat{f}_\omega^M : \mathbb{N} \to S$, where M is a restarting automaton of type X with input alphabet Σ.

Obviously, for the wRWW-automaton (M_2, ω_1) that is given in Example 3.3.2, the function $\hat{f}_{\omega_1}^{M_2}(n) = \frac{n}{4}$ for $n \geq 0$. Note that by the definition of L_2, $|w|$ is divisible by 4 for each $w \in L_2$.

4.2 Growth Rates

The purpose of this section is to present some results on the growth rates of the functions that are induced by weighted restarting automata. First, we start with an upper bound for a large subclass of functions in $\hat{\mathbb{F}}(\mathsf{X}, \Sigma, S)$. For $s \in S$ and $k \in \mathbb{N}$, we use the notation s^k to denote the k-fold product $s \cdot s \cdot \dots \cdot s$. In addition, $k \cdot s$ is used to denote the k-fold sum $s + s + \ldots + s$.

Theorem 4.2.1 ([OW16])**.** *Let $S = (S, +, \cdot, 0, 1)$ be a semiring that is ordered with respect to a linear order $\leq$, let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a restarting automaton, and let ω be a weight function that maps the transitions of M into the subset $S_+ = \{ s \in S \mid s \geq 0 \}$ of S. Further, let $s_M = \max(\{ \omega(t) \mid$ t is a transition of $M \} \cup \{1\})$. Then there exist constants $c_1, c_2 \in \mathbb{N}$ such that, for all $n \geq 1$, $\hat{f}^M_\omega(n) \leq c_1^{n^2} \cdot s_M^{c_2 \cdot n^2}$ holds.*

Proof. In order to obtain the upper bound for the function $\hat{f}^M_\omega : \mathbb{N} \to S$, we have to answer the following two questions:

(1) What is the maximal length of a computation of M on an input of length n? For $n \geq 0$, let

$$\hat{l}_M(n) = \max\{ |C| \mid C \in C_M(w), w \in \Sigma^n \},$$

where $|C|$ denotes the number of steps in the computation C, that is, its length, and $C_M(w)$ is the set of accepting computations of M on input w.

(2) What is the maximal number of accepting computations of M for any input of length n? For $n \geq 0$, let

$$\hat{r}_M(n) = \max\{ |C_M(w)| \mid w \in \Sigma^n \},$$

where $|C_M(w)|$ denotes the cardinality of the set $C_M(w)$.

It is easy to see that

$$\hat{f}^M_\omega(n) \leq \hat{r}_M(n) \cdot s_M^{\hat{l}_M(n)}$$

for all $n \geq 1$. Hence, it suffices to derive upper bounds for the numbers $\hat{l}_M(n)$ and $\hat{r}_M(n)$.

First, we consider the former number. For an integer $n \geq 1$, let $\mathrm{mcl}_M(n)$ denote the maximal number of steps in any cycle of M on any input of length at most n. From the definition of the restarting automaton it follows that $\mathrm{mcl}_M(n) \leq n + 2$, as a cycle of M that begins with a tape inscription of the form $¢w\$$, where $|w| = n$, contains exactly one rewrite step, one restart step, and up to at most n move-right steps. Analogously, a tail computation that begins with the above tape inscription consists of at most $n+2$ steps, where an accept step may replace the restart step. Further, the rewrite step that M executes within a cycle shortens the length of the tape, and thus each new cycle starts on a shorter word than the previous one. Hence, for any input w of length at most n, the length of any computation of M on input w is bounded from above by the number

$$\sum_{i=0}^{n}(i+2) = \sum_{i=2}^{n+2} i = \frac{1}{2}(n+2)(n+3) - 1 \leq 5 \cdot n^2,$$

that is, we see that $\hat{l}_M(n) \leq 5 \cdot n^2$ holds for all $n \geq 1$.

Now, we turn to the number $\hat{r}_M(n)$. Let d_M denote the maximal number of instructions of M, that is,

$$d_M = \max\{|\{\, \delta(q,u) \neq \emptyset \mid \text{for each } q \in Q \text{ and possible window content } u \,\}|\}.$$

Then for any configuration of M, there are at most d_M immediate successor configurations. Hence, it follows that

$$\hat{r}_M(n) \leq d_M^{5 \cdot n^2} = \left(d_M^5\right)^{n^2}$$

for all $n \geq 1$. Thus, we obtain that

$$\hat{f}_\omega^M(n) \leq \hat{r}_M(n) \cdot s_M^{\hat{l}_M(n)} \leq \left(d_M^5\right)^{n^2} \cdot s_M^{5 \cdot n^2}$$

for all $n \geq 1$, that is, the above statement holds for the constants $c_1 = d_M^5$ and $c_2 = 5$. □

Our next result shows that the upper bound given in the theorem above is actually sharp.

Theorem 4.2.2 ([OW16]). *Let $S = (S, +, \cdot, 0, 1)$ be a linearly ordered semiring, let $s \in S$ such that $s \geq 0$, let Σ be a finite alphabet, and let $c_1, c_2 \in \mathbb{N}_+$. Then there exist a* det-R*-automaton M with input alphabet Σ and a weight function ω for M such that*

$$\hat{f}_\omega^M(n) = c_1^{n^2+5n+2} \cdot s^{c_2 \cdot (n^2+5n+2)}$$

holds for all $n \geq 0$.

Proof. We define a det-R-automaton $M = (Q, \Sigma, \Sigma, ¢, \$, q_0, 2, \delta)$, where $Q = \{q_0, q_1, q_r\}$, $\Sigma = \{a, b\}$, and δ contains the following transitions:

$$\begin{array}{rlcll}
t_{1,a}: & (q_0, ¢a) & \rightarrow & (q_1, \mathsf{MVR}) & \text{for all } a \in \Sigma, \\
t_2: & (q_0, ¢\$) & \rightarrow & \mathsf{Accept}, & \\
t_{3,a,b}: & (q_1, ab) & \rightarrow & (q_1, \mathsf{MVR}) & \text{for all } a, b \in \Sigma, \\
t_{4,a}: & (q_1, a\$) & \rightarrow & (q_r, \$) & \text{for all } a \in \Sigma, \\
t_5: & (q_r, \$) & \rightarrow & \mathsf{Restart}. &
\end{array}$$

For example, M executes the following computation on the input word $w = aabb$:

$$\begin{array}{cccccccc}
q_0¢aabb\$ & \vdash_M & ¢q_1aabb\$ & \vdash_M & ¢aq_1abb\$ & \vdash_M & ¢aaq_1bb\$ \\
 & \vdash_M & ¢aabq_1b\$ & \vdash_M & ¢aabq_r\$ & \vdash_M & q_0¢aab\$ \\
 & \vdash_M & ¢q_1aab\$ & \vdash_M & ¢aq_1ab\$ & \vdash_M & ¢aaq_1b\$ \\
 & \vdash_M & ¢aaq_r\$ & \vdash_M & q_0¢aa\$ & \vdash_M & ¢q_1aa\$ \\
 & \vdash_M & ¢aq_1a\$ & \vdash_M & ¢aq_r\$ & \vdash_M & q_0¢a\$ \\
 & \vdash_M & ¢q_1a\$ & \vdash_M & ¢q_r\$ & \vdash_M & q_0¢\$ \\
 & \vdash_M & \mathsf{Accept}. & & & &
\end{array}$$

As M is deterministic, there is only a single computation for each input. Actually, it is easily seen that $L(M) = \Sigma^*$, and that, for each word w of length n, the accepting computation of M on input w consists of n cycles. For a tape inscription x of length $k > 0$, the cycle starting from the restarting configuration $q_0¢x\$$ consists of k move-right steps, a single rewrite step that deletes the last symbol of x, and a restart step, that is, it has length $k + 2$. As the tail computation consists of a single accept step, it follows that

$$|C| = \sum_{k=1}^{n}(k + 2) + 1 = \frac{1}{2}(n^2 + 5n + 2)$$

for each computation C of M on any input of length n, that is, $\hat{l}_M(n) = \frac{1}{2}(n^2 + 5n + 2)$ in the notation of the proof of the previous theorem.

Now we define the weight function ω by taking

$$\omega(t) = c_1^2 \cdot s^{2c_2} = (s^{2c_2} + \ldots + s^{2c_2}) \ (c_1^2 \text{ times})$$

for each transition t of M. It follows that

$$f_\omega^M(w) = \left(c_1^2 \cdot s^{2c_2}\right)^{\hat{l}_M(n)}$$

for each word $w \in \Sigma^n$, which implies that

$$\begin{array}{rcl}
\hat{f}_\omega^M(n) & = & (c_1^2 \cdot s^{2c_2})^{\frac{1}{2}(n^2+5n+2)} \\
 & = & c_1^{n^2+5n+2} \cdot s^{c_2 \cdot (n^2+5n+2)}
\end{array}$$

for all $n \geq 0$. □

Now we restrict our attention to the semiring $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$ with the natural order, and we present some families of functions from $\mathbb{N}$ into that semiring that are contained in the class of growth functions $\hat{\mathbb{F}}(\mathsf{RWW}, \{a\}, \mathbb{N})$.

Theorem 4.2.3 ([OW16]). *For all constants $c_1, c_2 \in \mathbb{N}_+$, there exist a* det-R-*automaton M with input alphabet $\Sigma = \{a\}$ and a weight function ω such that $\hat{f}_\omega^M(n) = c_1 \cdot c_2^n$ holds for all $n \geq 0$.*

Proof. We define a det-R-automaton $M = (Q, \Sigma, \Sigma, ¢, \$, q_0, 3, \delta)$, where $Q = \{q_0, q_r\}$, $\Sigma = \{a\}$, and δ contains the following transitions:

$$\begin{array}{rlcl rlcl}
t_1: & (q_0, ¢aa) & \to & (q_r, ¢a), & t_4: & (q_0, ¢\$) & \to & \mathsf{Accept},\\
t_2: & (q_0, ¢a\$) & \to & (q_0, \mathsf{MVR}), & t_5: & (q_0, a\$) & \to & \mathsf{Accept},\\
t_{3,x}: & (q_r, x) & \to & \mathsf{Restart} & \multicolumn{4}{l}{\text{for all } x \in \{aaa, aa\$, a\$, \$\}.}
\end{array}$$

Then it is easily seen that $L(M) = \Sigma^*$. Each cycle of a computation of M consists of two steps, that is, a rewrite and a restart step, the tail computation for the tape inscription $¢a\$$ consists of two steps, that is, a move-right step and an accept step, and the tail computation for the tape inscription $¢\$$ consists of a single accept step. Thus, it follows that on each input of length n, the maximal length of a computation of M is

$$\hat{l}_M(n) = \begin{cases} 2n & \text{for } n \geq 1,\\ 1 & \text{for } n = 0.\end{cases}$$

Let $c_1, c_2 \in \mathbb{N}_+$, and let ω be the weight function that assigns weight c_1 to the two accept transitions t_4 and t_5, that assigns weight c_2 to the rewrite transition t_1 and the move-right transition t_2, and that assigns weight 1 to all restart transitions $t_{3,x}$ for $x \in \{aaa, aa\$, a\$, \$\}$. Given an input of length n, the computation of M contains $n-1$ rewrite steps, $n-1$ restart steps, a single move-right step, and a single accept step. Hence, it follows that $f_\omega^M(a^n) = c_1 \cdot c_2^n$ for $n \geq 1$, and $f_\omega^M(a^0) = f_\omega^M(\lambda) = c_1 = c_1 \cdot c_2^0$. □

Theorem 4.2.4 ([OW16]). *For all constants $c, k \in \mathbb{N}_+$, there exist a* det-RWW-*automaton M with input alphabet $\Sigma = \{a\}$ and a weight function ω such that*

$$\hat{f}_\omega^M(n) = \begin{cases} c \cdot n^k, & \text{if } n = 2^m \text{ for some } m \geq 0,\\ 0, & \text{otherwise.}\end{cases}$$

Proof. Let $M = (Q, \{a\}, \Gamma, ¢, \$, q_0, 3, \delta)$ be the det-RWW-automaton that is defined by taking $Q = \{q_0, q_r\}$ and $\Gamma = \{a, b, A\}$, and by defining δ as follows, where $x \in \Gamma^3 \cup \Gamma^{\leq 2} \cdot \$$:

$$\begin{array}{rlcl rlcl}
t_1: & (q_0, ¢aa) & \to & (q_0, \mathsf{MVR}), & t_9: & (q_0, bbA) & \to & (q_r, AA),\\
t_2: & (q_0, ¢bb) & \to & (q_0, \mathsf{MVR}), & t_{10}: & (q_0, bb\$) & \to & (q_r, A\$),\\
t_3: & (q_0, ¢AA) & \to & (q_0, \mathsf{MVR}), & t_{11}: & (q_0, AAb) & \to & (q_r, bb),\\
t_4: & (q_0, aaa) & \to & (q_0, \mathsf{MVR}), & t_{12}: & (q_0, AA\$) & \to & (q_r, b\$),\\
t_5: & (q_0, bbb) & \to & (q_0, \mathsf{MVR}), & t_{13,x}: & (q_r, x) & \to & \mathsf{Restart},\\
t_6: & (q_0, AAA) & \to & (q_0, \mathsf{MVR}), & t_{14}: & (q_0, ¢a\$) & \to & \mathsf{Accept},\\
t_7: & (q_0, aab) & \to & (q_r, bb), & t_{15}: & (q_0, ¢b\$) & \to & \mathsf{Accept},\\
t_8: & (q_0, aa\$) & \to & (q_r, b\$), & t_{16}: & (q_0, ¢A\$) & \to & \mathsf{Accept}.
\end{array}$$

For example, on the input $w = aaaa$ M can execute the following computation:

$$\begin{array}{rcccccc} q_0¢aaaa\$ & \vdash_M & ¢q_0aaaa\$ & \vdash_M & ¢aq_0aaa\$ & \vdash_M & ¢aaq_0aa\$ \\ & \vdash_M & ¢aabq_r\$ & \vdash_M & q_0¢aab\$ & \vdash_M & ¢q_0aab\$ \\ & \vdash_M & ¢bbq_r\$ & \vdash_M & q_0¢bb\$ & \vdash_M & ¢q_0bb\$ \\ & \vdash_M & ¢Aq_r\$ & \vdash_M & q_0¢A\$ & \vdash_M & \mathsf{Accept}. \end{array}$$

It is easily seen that M accepts each input word of length 2^n for $n \geq 0$, that is, $L(M) = \{ a^{2^n} \mid n \geq 0 \}$.

Now let $c, k \in \mathbb{N}_+$. We define the weight function ω that assigns weight 2^k to transitions t_8, t_{10}, and t_{12}, that assigns weight c to transitions t_{14}, t_{15} and t_{16}, and that assigns weight 1 to all other transitions. On the input a^{2^m}, M first executes 2^{m-1} cycles, and during this phase a^{2^m} is rewritten into $b^{2^{m-1}}$. In the first of these cycles, the rewrite transition t_8 is applied, while in the other cycles, the rewrite transition t_7 is performed. Thus, the weight of this part of the computation is 2^k. Next the tape inscription $b^{2^{m-1}}$ is rewritten into $A^{2^{m-2}}$ within 2^{m-2} cycles, where in the first cycle the rewrite transition t_{10} is used, while in the other cycles, the rewrite transition t_9 is used. Thus, also this part of the computation has weight 2^k. Finally, the tape inscription $A^{2^{m-2}}$ is rewritten into $b^{2^{m-3}}$ within 2^{m-3} cycles, where in the first cycle rewrite transition t_{12} is used, while in the other cycles, the rewrite transition t_{11} is used. Thus, also this part of the computation has weight 2^k. The latter two types of sequences of cycles alternate until tape inscription b or A is reached, which is then accepted by using the accept transition t_{15} or t_{16}. It follows that, if $n = 2^m$ for some $m \geq 0$, then

$$f_\omega^M(a^n) = f_\omega^M(a^{2^m}) = c \cdot \left(2^k\right)^m = c \cdot \left(2^k\right)^{\log n} = c \cdot n^k,$$

and $f_\omega^M(a^n) = 0$, if n is not a power of 2. □

By using nondeterministic types of restarting automata, we can combine some copies of the deterministic RWW-automaton into a nondeterministic RWW-automaton, such that a polynomial can be induced.

Theorem 4.2.5 ([OW16]). *For each polynomial $P(x)$ over $\mathbb{N}$, there exist an RWW-automaton M with input alphabet $\Sigma = \{a\}$ and a weight function ω such that*

$$\hat{f}_\omega^M(n) = \begin{cases} P(n), & \text{if } n = 2^m \text{ for some } m \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

Proof. First, let $P(x)$ be a polynomial over $\mathbb{N}$, that is,

$$P(x) = c_1 \cdot x^{k_1} + c_2 \cdot x^{k_2} + \ldots + c_r \cdot x^{k_r} + d,$$

where $r \geq 0$, $c_1, k_1, c_2, k_2, \ldots, c_r, k_r \geq 1$, and $d \geq 0$.

Then we construct an RWW-automaton M by combining $r+1$ copies of the det-RWW-automaton that is presented in the proof of the previous theorem. Accordingly, we define $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, 3, \delta)$, where $Q = \{q_0, q_r\}$, $\Gamma = \{\, a, b_i, A_i \mid i = 1, 2, \ldots, r+1 \,\}$, and δ contains the following transitions:

$$\begin{array}{llcll}
t_1: & (q_0, ¢aa) & \to & (q_0, \mathsf{MVR}), & \\
t_2: & (q_0, aaa) & \to & (q_0, \mathsf{MVR}), & \\
t_{3.i}: & (q_0, aab_i) & \to & (q_r, b_ib_i) & \text{for } i = 1, \ldots, r+1, \\
t_{4.i}: & (q_0, aa\$) & \to & (q_r, b_i\$) & \text{for } i = 1, \ldots, r+1, \\
t_5: & (q_0, ¢a\$) & \to & \mathsf{Accept}, & \\
t_6: & (q_r, x) & \to & \mathsf{Restart} & \text{for all } x \in \Gamma^3 \cup \Gamma^{\leq 2} \cdot \$, \\
t_{7.i}: & (q_0, ¢b_ib_i) & \to & (q_0, \mathsf{MVR}) & \text{for } i = 1, \ldots, r+1, \\
t_{8.i}: & (q_0, b_ib_ib_i) & \to & (q_0, \mathsf{MVR}) & \text{for } i = 1, \ldots, r+1, \\
t_{9.i}: & (q_0, b_ib_iA_i) & \to & (q_r, A_iA_i) & \text{for } i = 1, \ldots, r+1, \\
t_{10.i}: & (q_0, b_ib_i\$) & \to & (q_r, A_i\$) & \text{for } i = 1, \ldots, r+1, \\
t_{11.i}: & (q_0, ¢b_i\$) & \to & \mathsf{Accept} & \text{for } i = 1, \ldots, r+1, \\
t_{12.i}: & (q_0, ¢A_iA_i) & \to & (q_0, \mathsf{MVR}) & \text{for } i = 1, \ldots, r+1, \\
t_{13.i}: & (q_0, A_iA_iA_i) & \to & (q_0, \mathsf{MVR}) & \text{for } i = 1, \ldots, r+1, \\
t_{14.i}: & (q_0, A_iA_ib_i) & \to & (q_r, b_ib_i) & \text{for } i = 1, \ldots, r+1, \\
t_{15.i}: & (q_0, A_iA_i\$) & \to & (q_r, b_i\$) & \text{for } i = 1, \ldots, r+1, \\
t_{16.i}: & (q_0, ¢A_i\$) & \to & \mathsf{Accept} & \text{for } i = 1, \ldots, r+1.
\end{array}$$

It is easily seen that $L(M) = \{\, a^{2^n} \mid n \geq 0 \,\}$, and that for each choice of auxiliary symbols b_i and A_i $(1 \leq i \leq r+1)$, there is an accepting computation (see instructions $t_{4.i}$). Thus, on an input of the form a^{2^n} $(n \geq 1)$, M has $r+1$ accepting computations.

Now we define a weight function ω as follows

$$\omega(t_x) = \begin{cases} 2^{k_i} & \text{for } x \in \{4.i, 10.i, 15.i\}\ (1 \leq i \leq r), \\ c_i & \text{for } x \in \{11.i, 16.i\}\ (1 \leq i \leq r), \\ d & \text{for } x = 4.r+1, \\ P(1) & \text{for } x = 5, \\ 1 & \text{for all other cases.} \end{cases}$$

As in the proof of Theorem 4.2.4, it follows that the computation of M on input a^{2^m} $(m \geq 1)$ that uses the auxiliary symbols b_i and A_i for some $i \in \{1, \ldots, r\}$ has weight $c_i \cdot \left(2^{k_i}\right)^m$. Further, the corresponding computation that uses the auxiliary symbols b_{r+1} and A_{r+1} has weight d. Accordingly, it follows that, if $n = 2^m$ for some $m \geq 1$, then

$$f_\omega^M(a^n) = c_1 \cdot n^{k_1} + c_2 \cdot n^{k_2} + \ldots + c_r \cdot n^{k_r} + d = P(n).$$

Further, we have $f_\omega^M(a) = P(1)$, and $f_\omega^M(a^n) = 0$, if n is not a power of two. This completes the proof of Theorem 4.2.5. $\square$

By combining the RWW-automaton from the proof of the last theorem with the $\mathsf{det\text{-}R}$-automaton from the proof of Theorem 4.2.3, it can be shown that weighted restarting automata can also represent functions that can be expressed as a sum of a polynomial and exponential functions. Thus, we see that the class of functions $\hat{\mathbb{F}}(\mathsf{RWW}, \{a\}, (\mathbb{N}, +, \cdot, 0, 1))$ is quite rich.

4.3 Closure Properties

In [JLNO04] it is shown that the language classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ are closed under the operations of union and concatenation. In this section we extend these results to wRWW- and wRRWW-automata. We will show that the classes of functions $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ and $\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$ are closed under the operations of addition, scalar multiplication, and Cauchy product, that is, if $f, g : \Sigma^* \to S$ belong to $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ (or $\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$), then also the functions $(f + g)$, $(s \cdot f)$ (for $s \in S$), and $(f \cdot g) : \Sigma^* \to S$ belong to this class of functions, where, for all $w \in \Sigma^*$,

$$(f + g)(w) = f(w) + g(w),$$

$$(s \cdot f)(w) = s \cdot f(w),$$

and

$$(f \cdot g)(w) = \sum_{w=uv} (f(u) \cdot g(v)) .$$

Hence, if M_1 and M_2 are restarting automata of type $X \in \{\mathsf{RWW}, \mathsf{RRWW}\}$ with input alphabet Σ, and if ω_1 and ω_2 are weight functions for M_1 and M_2, then there exist restarting automata M_+, M_s, and M_c of type X and weight functions ω_+, ω_s, and ω_c such that $f_{\omega_+}^{M_+} = f + g$, $f_{\omega_s}^{M_s} = s \cdot f$, and $f_{\omega_c}^{M_c} = f \cdot g$. We begin with the operation of addition. To do this, we need the following technical lemma.

Lemma 4.3.1. *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*, if* $\mathcal{M} = (M, \omega)$ *is a* wX*-automaton of window size* $k \leq 2$*, then there exists a* wX*-automaton* $\mathcal{M}' = (M', \omega')$ *of window size* $k' = 3$ *such that* $f_{\omega}^{M}(w) = f_{\omega'}^{M'}(w)$ *for all* $w \in \Sigma^*$*.*

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a restarting automaton of type X, where $k \leq 2$, and let ω be a weight function from the transitions of M into a semiring $(S, +, \cdot, 0, 1)$. First, we consider the case that $\mathsf{X} \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RWW}\}$, that is, after performing a rewrite step M restarts immediately. If $k = 1$, then the transition function δ' of M' and the weight function ω' are described as follows.

1. First we define some transitions that allow M' to simulate move-right transitions of M. If δ contains a move-right transition of the form
$$t : (p, u) \to (q, \mathsf{MVR})$$
for $u \in \{¢\} \cup \Gamma$, then δ' contains the following transitions for all admissible words $x \in \Gamma^2 \cup (\Gamma \cdot \$) \cup \{\$\}$:
$$t_x : (p, ux) \to (q, \mathsf{MVR}),$$
and ω' assigns the weight $\omega(t)$ to all these transitions.

2. Next we define some transitions that enable M' to simulate rewrite transitions of M. If δ contains a rewrite transition of the form
$$t : (p, u) \to (q, \lambda)$$
for some $u \in \Gamma$, then δ' contains the following transitions for all admissible words $x \in \Gamma^2 \cup (\Gamma \cdot \$) \cup \{\$\}$:
$$t_x : (p, ux) \to (q, x),$$
and ω' assigns the weight $\omega(t)$ to all these transitions.

3. Now we consider the restart transitions. As $\mathsf{X} \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RWW}\}$, each rewrite operation of M is immediately followed by a restart step. Hence, if a state q of M is entered through a rewrite step, then in state q, M must restart immediately, that is, δ contains the transitions
$$t : (q, u) \to \mathsf{Restart}$$
for each possible window content $u \in \Gamma \cup \{\$\}$. Accordingly, δ contains the following transitions for all admissible words $x \in \Gamma^2 \cup (\Gamma \cdot \$) \cup \{\$, \lambda\}$:
$$t_x : (q, ux) \to \mathsf{Restart},$$
and we take $\omega'(t_x) = \omega(t)$.

4. Finally, we define those transitions that allow M' to simulate accept transitions of M. If δ contains an accept transition of the form
$$t : (q, u) \to \mathsf{Accept}$$
for some $u \in \{¢\} \cup \Gamma \cup \{\$\}$, then δ' contains the following transitions for all admissible words $x \in \Gamma^2 \cup (\Gamma \cdot \$) \cup \{\$, \lambda\}$:
$$t_x : (q, ux) \to \mathsf{Accept},$$
and we take $\omega'(t_x) = \omega(t)$.

Above we considered the case that $k = 1$, and in an analogous way the case that $k = 2$ can also be dealt with.

Now we turn to the case that $\mathsf{X} \in \{\mathsf{RR}, \mathsf{RRW}, \mathsf{RRWW}\}$. The move-right, restart and accept transitions can be simulated in the same way as for the case $\mathsf{X} \in \{\mathsf{R}, \mathsf{RW}, \mathsf{RWW}\}$. However, as k' is strictly larger than k, after performing the above rewrite transition $t_x : (p, ux) \to (q, vx)$, the window of M' skips across the word x of length $k' - k$. In order to simplify the discussion, we assume that M only performs restart operations at the right end of its tape. This is easily achieved by replacing every restart transition by a move-right step (with the same weight as the corresponding restart step), which enters a special state q_{mv}, and in state q_{mv}, M moves all the way to the right end of its tape and performs a restart step on the \$-symbol, where all these additional transitions have weight 1. Under this assumption a rewrite step of M can be simulated by combining a rewrite step of M with $k' - k$ many move-right steps. First, we consider the case that $k = 2$, and we define the following states for M':

$$Q_{rw} = \{ (q_1, xy, q_2) \mid q_1, q_2 \in Q \text{ and } x, y \in \Gamma \}.$$

If δ contains the transitions of the forms

$$\begin{array}{rlcl} t_1 : & (p, u) & \to & (q_1, v), \\ t_2 : & (q_1, xy) & \to & (q_2, \mathsf{MVR}), \\ t_3 : & (q_2, yz) & \to & (q_3, \mathsf{MVR}), \end{array}$$

where $u \in \{¢\} \cdot \Gamma \cup \Gamma^2$, $v \in \Gamma \cup \{¢, \lambda\}$, and $x, y, z \in \Gamma$, then δ' contains the following transitions for all admissible words $z_1 \in \Gamma \cup \{\$\}$

$$\begin{array}{rlcl} t'_x : & (p, ux) & \to & ((q_1, xy, q_2), vx), \\ t'_{yzz_1} : & ((q_1, xy, q_2), yzz_1) & \to & (q_3, \mathsf{MVR}), \end{array}$$

and we take $\omega'(t'_x) = \omega(t_1) \cdot \omega(t_2)$ and $\omega'(t'_{yzz_1}) = \omega(t_3)$. Note that if $v = \$$, M' does not need additional move-right steps. Analogously, the case that $|u| = 1$ can also be dealt with, which completes the proof. □

Theorem 4.3.1 ([OW16]). *For all alphabets Σ and semirings S, the classes of functions $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ and $\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$ are closed under the operation of addition.*

Proof. Let $S = (S, +, \cdot, 0, 1)$ be a semiring, let $M_1 = (Q_1, \Sigma, \Gamma_1, ¢, \$, q_0^{(1)}, k_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, \Gamma_2, ¢, \$, q_0^{(2)}, k_2, \delta_2)$ be RWW-automata with input alphabet Σ, and let ω_1 and ω_2 be weight functions that map the transitions of M_1 and of M_2 to S. In order to prove that $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ is closed under the operation of addition, we construct an RWW-automaton M_+ with input alphabet Σ and a weight function ω_+ such that

$$f_{\omega_+}^{M_+}(w) = f_{\omega_1}^{M_1}(w) + f_{\omega_2}^{M_2}(w)$$

holds for all $w \in \Sigma^*$.

On an input $w \in \Sigma^*$, the automaton M_+ can choose to simulate a computation of M_1 or a computation of M_2 on the input w. However, after a restart step M_+ is not able to remember its choice within its finite-state control. Therefore, it has to store this information on the tape, so that it can read this information after a restart step. Accordingly, M_+ will place a marking on the prefix of the input in the first cycle. Unfortunately, each rewrite step must be strictly length-reducing. In order to satisfy this condition, M_+ can replace the first two symbols a_1a_2 of the input by a special symbol $[a_1, a_2, 1]$ or $[a_1, a_2, 2]$ as an indicator on the automaton that M_+ has chosen to simulate.

However, in some later rewrite transitions, the automaton M_i ($i \in \{1, 2\}$) that is being simulated may just remove the symbol a_1 or a_2 without changing any other symbol. If M_+ replaces the symbol $[a_1, a_2, i]$ simply by some symbol encoding the remaining symbol a_2 (or a_1) together with the indicator i, i.e., $[a_1, i]$ or $[a_2, i]$, then this rewrite step of M_+ would not be length-reducing. In order to solve this problem, M_+ will combine the symbol of the form $[a_1, i]$ (or $[a_2, i]$) with the next symbol x into the new symbol $[a_1, x, i]$ (or $[a_2, x, i]$). For this purpose, M_+ needs a read/write window that is larger than those of M_1 and M_2.

Now we describe the construction of M_+ and the weight function ω_+ in detail. Let $M_+ = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be the RWW-automaton that is defined as follows:

- $Q = \{q_0, q_r\} \cup \{\, q^{(1)} \mid q \in Q_1 \,\} \cup \{\, q^{(2)} \mid q \in Q_2 \,\}$
  $\cup \{\, q^{(i)}_{\mathrm{MVR}}, q^{(i)}_{a}, q^{(i)}_{a'}, q^{(i)}_{0'}, q^{(i)}_{0''} \mid i = 1, 2 \,\}$,
- $\Gamma = \Gamma_1 \cup \Gamma_2$
  $\cup \{\, [a_1, a_2, 1], [a_1, 1], [1] \mid a_1, a_2 \in \Gamma_1 \,\}$
  $\cup \{\, [a_1, a_2, 2], [a_1, 2], [2] \mid a_1, a_2 \in \Gamma_2 \,\}$,
- $k = \max\{k_1, k_2\} + 1$, and
- the transition function δ of M_+ and the weight function ω_+ are as described below.

We now present the definition of δ step by step. By Lemma 4.3.1, here we only consider the case that $\max\{k_1, k_2\} \geq 3$, which means that $k \geq 4$.

1. First we define some transitions that enable M_+ to deal with those inputs $w \in \Sigma^*$, where $|w| \leq k-2$. For each $w \in \Sigma^{\leq k-2}$ and $w \in L(M_1) \cup L(M_2)$, we define the following transition,

$$t_w : (q_0, ¢w\$) \to \mathsf{Accept}.$$

In addition, we define $\omega_+(t_w) = f^{M_1}_{\omega_1}(w) + f^{M_2}_{\omega_2}(w)$. Hence, for all input words $w \in \Sigma^{\leq k-2}$, $f^{M_+}_{\omega_+}(w) = f^{M_1}_{\omega_1}(w) + f^{M_2}_{\omega_2}(w)$ holds.

2. Next we define some transitions that allow M_+ to process restarting configurations of the form $q_0¢z\$$, where $z \in \Gamma^+ \smallsetminus \Sigma^*$ is of length at most $k-2$, that is, the complete tape content $¢z\$$ is contained in the window of M_+. By our strategy described above, the first letter z_1 of z encodes the choice of which automaton M_+ currently simulates, that is, $z_1 \in \Gamma \smallsetminus (\Gamma_1 \cup \Gamma_2)$. Accordingly, $z_1 \in \{[a_1, a_2, i], [a_1, i], [i]\}$ for some $a_1, a_2 \in \Gamma_i$ and $i \in \{1, 2\}$. If M_i accepts starting from the restarting computation $q_0^{(i)}¢a_1a_2z'\$$ (or $q_0^{(i)}¢a_1z'\$$ or $q_0^{(i)}¢z'\$$) for $z = z_1z'$, then M_+ has an accepting transition

$$t_z : (q_0, ¢z\$) \to \mathsf{Accept},$$

and we take $\omega_+(t_z) = s$, where s is the sum of the weights of all accepting computations of M_i that start from this configuration. Note that there may be several computations of M_i that start from the initial configuration $q_0^{(i)}¢w\$$, and reach the configuration $q_0^{(i)}¢a_1a_2z'\$$, and thus there may also be several accepting computations of M_i that start from the latter configuration. During the simulation of M_1 or M_2, whenever M_+ reaches a restarting configuration $q_0¢z\$$ such that $|z| \leq k-2$, then the corresponding transition t_z is performed, which ends the current computation. Because of the distributive law of semirings, $f^{M_+}_{\omega_+}(w) = f^{M_1}_{\omega_1}(w) + f^{M_2}_{\omega_2}(w)$ holds for all $w \in \Sigma^*$. Hence, in the following we only need to consider the case that the length of the tape content is larger than the size k of the window of M_+.

3. Now we define some transitions that allow M_+ to place a marking on the prefix of a given input word of length at least $k-1 \geq 3$ in order to store its choice between simulating M_1 or M_2:

$$\begin{aligned} t_{(a_1,a_2,i)} :\ & (q_0, ¢a_1a_2x) && \to && (q_r, ¢[a_1, a_2, i]x), \\ t_{r,w} :\ & (q_r, w) && \to && \mathsf{Restart}, \end{aligned}$$

where $a_1, a_2 \in \Sigma$, $i \in \{1, 2\}$, $x \in \Sigma^{k-3}$, and w denotes all possible window contents. Further, we take

$$\omega_+(t_{(a_1,a_2,i)}) = \omega_+(t_{r,w}) = 1$$

for all $a_1, a_2 \in \Sigma$ and $i \in \{1, 2\}$.

4. Here we define those transitions that allow M_+ to simulate move-right steps of M_1 and M_2, where we must distinguish between several cases.

(4.1) If δ_i $(i \in \{1,2\})$ contains a transition of the form

$$t : (q_0^{(i)}, ¢a_1a_2u) \to (q_l, \mathsf{MVR})$$

for some $q_l \in Q_i$, $a_1, a_2 \in \Gamma_i$, and $u \in \Gamma_i^*$ satisfying $|¢a_1a_2u| = k_i$, then δ contains the following transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\begin{array}{rlcl} \hat{t}_1 : & (q_0, ¢[a_1, a_2, i]ux) & \to & (q_l^{(i)}, \mathsf{MVR}),\\ \hat{t}_2 : & (q_0, ¢[a_1, i]a_2ux) & \to & (q_l^{(i)}, \mathsf{MVR}),\\ \hat{t}_3 : & (q_0, ¢[i]a_1a_2ux) & \to & (q_{\mathrm{MVR}}^{(i)}, \mathsf{MVR}),\\ \hat{t}_4 : & (q_{\mathrm{MVR}}^{(i)}, [i]a_1a_2ux) & \to & (q_l^{(i)}, \mathsf{MVR}). \end{array}$$

For these transitions we define

$$\omega_+(\hat{t}_1) = \omega_+(\hat{t}_2) = \omega_+(\hat{t}_4) = \omega_i(t)$$

and $\omega_+(\hat{t}_3) = 1$. Then $\omega_+(\hat{t}_3) \cdot \omega_+(\hat{t}_4) = \omega_i(t)$, which means that together $\hat{t}_3$ and $\hat{t}_4$ simulate the transition t on a tape content of the form $¢[i]a_1a_2w\$$.

(4.2) If δ_i $(i \in \{1,2\})$ contains a transition of the form

$$t : (q_m, a_1a_2u) \to (q_l, \mathsf{MVR})$$

for some $q_m, q_l \in Q_i$, $a_1, a_2 \in \Gamma_i$, and $u \in \Gamma_i^*$ satisfying $|a_1a_2u| = k_i$, then δ contains the following transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\begin{array}{rlcl} \hat{t}_1 : & (q_m^{(i)}, a_1a_2ux) & \to & (q_l^{(i)}, \mathsf{MVR}),\\ \hat{t}_2 : & (q_m^{(i)}, [a_1, i]a_2ux) & \to & (q_l^{(i)}, \mathsf{MVR}), \end{array}$$

and we take

$$\omega_+(\hat{t}_1) = \omega_+(\hat{t}_2) = \omega_i(t).$$

Now we consider tape content of the form $¢[a_1, a_2, i]w\$$. In order to simulate the transition t correctly on such a tape content, we need to combine this transition with those transitions that M_i can perform in the configuration $¢a_1q_la_2w\$$. Based on the latter transitions, we have various options:

(a) If δ_i contains a transition of the form

$$t_1' : (q_l, a_2ua) \to (q_{l'}, \mathsf{MVR})$$

for some $q_{l'} \in Q_i$, $a \in \Gamma_i$, and $u \in \Gamma_i^*$ satisfying $|a_2ua| = k_i$, then δ contains the following additional transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\hat{t}'_1 : (q_m^{(i)}, [a_1, a_2, i]uax) \to (q_{l'}^{(i)}, \mathsf{MVR}),$$

where $\omega_+(\hat{t}'_1) = \omega_i(t) \cdot \omega_i(t'_1)$, as $\hat{t}'_1$ simulates the sequence of transitions t and t'_1 of M_i.

(b) If δ_i contains a transition of the form

$$t'_2 : (q_l, a_2ua) \to (q_{l'}, v)$$

for some $q_{l'} \in Q_i$, $a \in \Gamma_i$, $u, v \in \Gamma_i^*$ such that $|a_2ua| = k_i$ and $|v| < k_i$, then δ contains the following additional transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\hat{t}'_2 : (q_m^{(i)}, [a_1, a_2, i]uax) \to (q_{l'}^{(i)}, v'x),$$

where

$$v' = \begin{cases} [a_1, i]v, & \text{if } |v| < |ua|, \\ [a_1, a_3, i]\tilde{v}, & \text{if } |v| = |ua| \text{ and } v = a_3\tilde{v}. \end{cases}$$

Further, we take $\omega_+(\hat{t}'_2) = \omega_i(t) \cdot \omega_i(t'_2)$, as $\hat{t}'_2$ simulates the sequence of transitions t and t'_2.

(c) If δ_i contains a transition of the form

$$t'_3 : (q_l, a_2ua) \to \mathsf{Accept}$$

for some $a \in \Gamma_i$ and $u \in \Gamma_i^*$ satisfying $|a_2ua| = k_i$, then δ contains the following additional transitions for all admissible words $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\hat{t}'_3 : (q_m^{(i)}, [a_1, a_2, i]uax) \to \mathsf{Accept},$$

and for these transitions the weight function ω_+ is extended by taking $\omega_+(\hat{t}'_3) = \omega_i(t) \cdot \omega_i(t'_3)$.

(4.3) The case that δ_i ($i \in \{1, 2\}$) contains a transition of the form

$$t : (q_m, a_1a_2u\$) \to (q_l, \mathsf{MVR})$$

for some $q_m, q_l \in Q_i$, $a_1, a_2 \in \Gamma_i$, and $u \in \Gamma_i^*$ satisfying $|a_1a_2u\$| \leq k_i$ can be dealt with in the same way as (4.2). However, note that here we do not need to consider the case that the tape content contains a symbol of the form $[a_1, a_2, i]$, since by our construction such a symbol can only occur immediately to right of the left sentinel ¢, and $k > k_i$.

5. Here we present those transitions that allow M_+ to simulate rewrite steps of M_1 and M_2. However, after a rewrite step M_i must immediately restart. As k is strictly larger than k_i, the window of M_+ skips across the prefix of the window content for the subsequent restart transition of M_i. In order to overcome this problem, M_+ has to store the prefix in its finite control. Again we must distinguish between several cases.

(5.1) If δ_i ($i \in \{1,2\}$) contains transitions of the form

$$\begin{array}{rlcl} t: & (q_0^{(i)}, ¢a_1a_2u) & \rightarrow & (q_l, ¢v), \\ t_r: & (q_l, xy) & \rightarrow & \mathsf{Restart}, \end{array}$$

where $q_l \in Q_i$, $a_1, a_2 \in \Gamma_i$, $u, v \in \Gamma_i^*$ satisfying $|¢a_1a_2u| = k_i$ and $|v| \leq |u|+1$, $x, y \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$, and $|xy| \leq k_i$, then δ contains the following transitions for all admissible choices of $x, y, z \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\hat{t}_1 : (q_0, ¢[a_1, a_2, i]ux) \rightarrow (q_{l,x}^{(i)}, ¢\alpha x), \text{ where}$$
$$\alpha = \begin{cases} [i]v, & \text{if } |v| < |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u|,\ v = a_3\tilde{v}, \\ [a_3, a_4, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3a_4\tilde{v}, \end{cases}$$
$$\hat{t}_2 : (q_0, ¢[a_1, i]a_2ux) \rightarrow (q_{l,x}^{(i)}, ¢\alpha x), \text{ where}$$
$$\alpha = \begin{cases} [i]v, & \text{if } |v| \leq |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3\tilde{v}, \end{cases}$$
$$\hat{t}_3 : (q_0, ¢[i]a_1a_2ux) \rightarrow (q_{l,x}^{(i)}, ¢[i]vx),$$
$$\hat{t}_{r,xy} : (q_{l,x}^{(i)}, yz) \rightarrow \mathsf{Restart}.$$

Further, we define

$$\omega_+(\hat{t}_1) = \omega_+(\hat{t}_2) = \omega_+(\hat{t}_3) = \omega_i(t),$$

and

$$\omega_+(\hat{t}_{r,xy}) = \omega_i(t_r).$$

(5.2) If δ_i ($i \in \{1,2\}$) contains transitions of the form

$$\begin{array}{rlcl} t: & (q_m, a_1a_2u) & \rightarrow & (q_l, v), \\ t_r: & (q_l, xy) & \rightarrow & \mathsf{Restart}, \end{array}$$

where $q_m, q_l \in Q_i$, $a_1, a_2 \in \Gamma_i$, $u, v \in \Gamma_i^*$ satisfying $|a_1a_2u| = k_i$, $|v| \leq |u| + 1$, $x, y \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$, and $|xy| \leq k_i$, then δ contains the following transitions for all admissible choices of $x, y, z \in \Gamma_i^* \cup (\Gamma_i^* \cdot$

\$):

$$\hat{t}_1 : (q_m^{(i)}, [a_1, a_2, i]ux) \to (q_{l,x}^{(i)}, \alpha x), \text{ where}$$
$$\alpha = \begin{cases} [i]v, & \text{if } |v| < |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u|,\ v = a_3\tilde{v}, \\ [a_3, a_4, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3a_4\tilde{v}, \end{cases}$$
$$\hat{t}_2 : (q_m^{(i)}, [a_1, i]a_2ux) \to (q_{l,x}^{(i)}, \alpha x), \text{ where}$$
$$\alpha = \begin{cases} [i]v, & \text{if } |v| \leq |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3\tilde{v}, \end{cases}$$
$$\hat{t}_3 : (q_m^{(i)}, a_1a_2ux) \to (q_{l,x}^{(i)}, vx),$$
$$\hat{t}_{r,xy} : (q_{l,x}^{(i)}, yz) \to \mathsf{Restart}.$$

Further, we take

$$\omega_+(\hat{t}_1) = \omega_+(\hat{t}_2) = \omega_+(\hat{t}_3) = \omega_i(t),$$

and

$$\omega_+(\hat{t}_{r,xy}) = \omega_i(t_r).$$

(5.3) If δ_i ($i \in \{1, 2\}$) contains a transition of the form

$$t : (q_m, a_1a_2\$) \to (q_l, v\$),$$

where $q_m, q_l \in Q_i$, $a_1, a_2 \in \Gamma_i$, $v \in \Gamma_i^{\leq 1}$, then δ contains the following transitions:

$$\hat{t} : (q_m^{(i)}, a_1a_2\$) \to (q_l^{(i)}, v\$).$$

Further, let $\omega_+(\hat{t}) = \omega_i(t)$. Observe that by our assumption we do not need to consider the cases that M_+ must simulate transition t on a tape content of the form $[a_1, i]a_2\$$ or $[a_1, a_2, i]\$$.
In the same way, for a transition

$$t : (q_m, a_1\$) \to (q_l, \$)$$

of M_i, let δ contain the transition

$$\hat{t} : (q_m^{(i)}, a_1\$) \to (q_l^{(i)}, \$),$$

and let $\omega_+(\hat{t}) = \omega_i(t)$.

6. Now we consider the restart transitions. For RWW-automata, each rewrite operation is immediately followed by a restart step. Hence, if a

state q of M_i ($i \in \{1,2\}$) is entered through a rewrite step, then in state q, M_i must restart immediately, that is, δ_i contains the transitions

$$t_u : (q, u) \to \mathsf{Restart}$$

for each possible window content u. Accordingly, δ contains the following transitions for all admissible words $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\hat{t}_{u,x} : (q^{(i)}, ux) \to \mathsf{Restart},$$

and $\omega_+(\hat{t}_{u,x}) = \omega_i(t_u)$. Recall that a restart step is only performed after a rewrite step has been executed, which means that at this point the read/write window does not contain any symbol from $\Gamma \smallsetminus (\Gamma_1 \cup \Gamma_2)$.

7. Finally, we consider the accept transitions of M_1 and M_2, where we distinguish between two cases.

(7.1) If δ_i ($i \in \{1,2\}$) contains a transition of the form

$$t : (q_0, ¢a_1a_2u) \to \mathsf{Accept}$$

for some $a_1, a_2 \in \Gamma_i$ and $u \in \Gamma_i^*$ such that $|¢a_1a_2u| = k_i$, then δ contains the following transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\begin{array}{lll} \hat{t}_1 : & (q_0, ¢[a_1,a_2,i]ux) & \to \mathsf{Accept}, \\ \hat{t}_2 : & (q_0, ¢[a_1,i]a_2ux) & \to \mathsf{Accept}, \\ \hat{t}_3 : & (q_0, ¢[i]a_1a_2ux) & \to \mathsf{Accept}, \end{array}$$

and we take

$$\omega_+(\hat{t}_1) = \omega_+(\hat{t}_2) = \omega_+(\hat{t}_3) = \omega_i(t).$$

(7.2) If δ_i ($i \in \{1,2\}$) contains a transition of the form

$$t : (q_m, a_1a_2u) \to \mathsf{Accept}$$

for some $q_m \in Q_i$, $a_1, a_2 \in \Gamma_i$, and $u \in \Gamma_i^*$ such that $|a_1a_2u| = k_i$, then δ contains the following transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$:

$$\begin{array}{lll} \hat{t}_1 : & (q_m^{(i)}, [a_1,a_2,i]ux) & \to \mathsf{Accept}, \\ \hat{t}_2 : & (q_m^{(i)}, [a_1,i]a_2ux) & \to \mathsf{Accept}, \\ \hat{t}_3 : & (q_m^{(i)}, a_1a_2ux) & \to \mathsf{Accept}, \end{array}$$

and let

$$\omega_+(\hat{t}_1) = \omega_+(\hat{t}_2) = \omega_+(\hat{t}_3) = \omega_i(t).$$

This completes the proof for the case that M_1 and M_2 are RWW-automata.

Finally, we consider the case that M_1 and M_2 are RRWW-automata. First, in order to simplify the discussion, we observe that we can assume without loss of generality that M_1 and M_2 only perform restart operations at the right end of their tapes, that is, when the read/write window only contains the right sentinel \$. This is easily realized by replacing every other restart transition by a move-right step (with the same weight as the corresponding restart step), which enters a special state q_{mv}, and in state q_{mv}, the RRWW-automaton moves all the way to the right end of its tape and performs a restart step on the \$-symbol, where all these additional transitions have weight 1.

8. Under this assumption we now describe the construction of M_+ from M_1 and M_2. The transitions of M_+ for making the choice between simulating M_1 or M_2 and the transitions for simulating move-right and accept steps of M_1 and M_2 are defined as for RWW-automata (see above). Because of the above assumption on the restart operations, these are also easily simulated by M_+. Hence, it remains to deal with the rewrite transitions of M_1 and M_2. For this purpose we have to solve the following technical problem.

   Whenever M_i ($i \in \{1,2\}$) applies a rewrite operation $(q_m, u) \to (q_l, v)$ to a configuration of the form $¢w_1q_muw_2\$$, then the configuration $¢w_1vq_lw_2\$$ is obtained. As the size k of the read/write window of M_+ is strictly larger than that of the read/write window of M_i, the above operation is simulated by rewrite operations of the form $(q_m^{(i)}, ux) \to (q_{l'}^{(i)}, vx)$ for all admissible words $x \in \Gamma_i^+$. However, this means that from the configuration $¢w_1q_m^{(i)}uw_2\$ = ¢w_1q_m^{(i)}uxw_2'\$$, the configuration $¢w_1vxq_{l'}^{(i)}w_2'\$$ is obtained in a single step, that is, the window of M_+ skips across the prefix x of w_2 of length $k - k_i$, while the window of M_i must be shifted across x by applying $|x|$ many move-right steps. Unfortunately, we cannot simply define the weights of the above transitions of M_+ as the product of the weights of the rewrite transition of M_i and the corresponding move-right transitions of M_i, since these move-right transitions depend on the next $k_i - 1$ symbols following the factor x.

   To overcome this problem, we first introduce some additional states for M_+. For $i \in \{1,2\}$, let Q_i^{rw} denote the set of states of M_i that are reached through a rewrite step. Further, for $q \in Q_i^{\mathrm{rw}}$, $x \in \Gamma_i^{k-k_i}$, and $y \in \Gamma^{k_i-1} \cup \Gamma_i^{\leq k_i-2} \cdot \$$, let $C_i(q,x,y)$ be the set of computations of M_i that consist of $|x|$ move-right steps that take a configuration of the form $¢wqxyw'$ into a configuration of the form $¢wxq'yw'\$$ for some state

$q' \in Q_i$. We define the following states for M_+:

$$\begin{aligned} Q_{\text{rw}} \;=\; & \{\, q^{(1)}_{l,x,y,C} \mid q_l \in Q_1^{\text{rw}}, x \in \Gamma_1^{\le k-k_1}, \\ & y \in \Gamma_1^{k_1-1} \cup \Gamma_1^{\le k_1-2} \cdot \$, C \in C_1(q_l, x, y) \,\} \\ \cup \; & \{\, q^{(2)}_{l,x,y,C} \mid q_l \in Q_2^{\text{rw}}, x \in \Gamma_2^{\le k-k_2}, \\ & y \in \Gamma_2^{k_2-1} \cup \Gamma_2^{\le k_2-2} \cdot \$, C \in C_2(q_l, x, y) \,\}. \end{aligned}$$

Now we can proceed by replacing the rewrite transitions introduced in Case 5 above as follows, where again we distinguish between several cases.

(8.1) If δ_i $(i \in \{1,2\})$ contains a transition of the form

$$t : (q_0^{(i)}, ¢a_1a_2u) \to (q_l, ¢v)$$

for some $q_l \in Q_i^{\text{rw}}$, $a_1, a_2 \in \Gamma_i$, and $u, v \in \Gamma_i^*$ satisfying $|¢a_1a_2u| = k_i$ and $|v| \le |u| + 1$, then δ contains the following transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$, $y \in \Gamma_i^{k_i-1} \cup (\Gamma_i^{\le k_i-2} \cdot \$)$, and $C \in C_i(q_l, x, y)$:

$$\begin{aligned} &\hat{t}_{1,x,y,C} : (q_0, ¢[a_1, a_2, i]ux) \to (q^{(i)}_{l,x,y,C}, ¢\alpha x), \text{ where} \\ &\alpha = \begin{cases} [i]v, & \text{if } |v| < |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u|,\ v = a_3\tilde{v}, \\ [a_3, a_4, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3a_4\tilde{v}, \end{cases} \\ &\hat{t}_{2,x,y,C} : (q_0, ¢[a_1, i]a_2ux) \to (q^{(i)}_{l,x,y,C}, ¢\alpha x), \text{ where} \\ &\alpha = \begin{cases} [i]v, & \text{if } |v| \le |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3\tilde{v}, \end{cases} \\ &\hat{t}_{3,x,y,C} : (q_0, ¢[i]a_1a_2ux) \to (q^{(i)}_{l,x,y,C}, ¢[i]vx). \end{aligned}$$

Further,

$$\omega_+(\hat{t}_{1,x,y,C}) = \omega_+(\hat{t}_{2,x,y,C}) = \omega_+(\hat{t}_{3,x,y,C}) = \omega_i(t)$$

is chosen.

(8.2) If δ_i $(i \in \{1,2\})$ contains a transition of the form

$$t : (q_m, a_1a_2u) \to (q_l, v),$$

where $q_m \in Q_i$, $q_l \in Q_i^{\text{rw}}$, $a_1, a_2 \in \Gamma_i$, $u, v \in \Gamma_i^*$ satisfying $|a_1a_2u| = k_i$ and $|v| \le |u| + 1$, then δ contains the following transitions for all admissible choices of $x \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$, $y \in \Gamma_i^{k_i-1} \cup (\Gamma_i^{\leq k_i-2} \cdot \$)$, and $C \in C_i(q_l, x, y)$:

$$\hat{t}_{1,x,y,C} : (q_m^{(i)}, [a_1, a_2, i]ux) \to (q_{l,x,y,C}^{(i)}, \alpha x), \text{ where}$$

$$\alpha = \begin{cases} [i]v, & \text{if } |v| < |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u|,\ v = a_3\tilde{v}, \\ [a_3, a_4, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3a_4\tilde{v}, \end{cases}$$

$$\hat{t}_{2,x,y,C} : (q_m^{(i)}, [a_1, i]a_2ux) \to (q_{l,x,y,C}^{(i)}, \alpha x), \text{ where}$$

$$\alpha = \begin{cases} [i]v, & \text{if } |v| \leq |u|, \\ [a_3, i]\tilde{v}, & \text{if } |v| = |u| + 1,\ v = a_3\tilde{v}, \end{cases}$$

$$\hat{t}_{3,x,y,C} : (q_m^{(i)}, a_1a_2ux) \to (q_{l,x,y,C}^{(i)}, vx).$$

Further, we take

$$\omega_+(\hat{t}_{1,x,y,C}) = \omega_+(\hat{t}_{2,x,y,C}) = \omega_+(\hat{t}_{3,x,y,C}) = \omega_i(t).$$

(8.3) For each state $q_{l,x,y,C}^{(i)} \in Q_{\text{rw}}$, we have to add some transitions to δ. From the definition above we see that C is a computation of M_i that takes a configuration of the form $¢wq_lxyw'\$$ to the configuration $¢wxq'yw'\$$ for some $q' \in Q_i$. For $b \in \Gamma_i$, let

$$t_{q',yb} : (q', yb) \to (q_j, \mathsf{MVR})$$

be a transition of M_i that is applicable to a configuration of the form $¢wxq'ybw'\$$. Then we add the transition

$$\hat{t}_{l,x,y,C,b,z} : (q_{l,x,y,C}^{(i)}, ybz) \to (q_j^{(i)}, \mathsf{MVR})$$

to M_+ for all admissible choices of $z \in \Gamma_i^* \cup (\Gamma_i^* \cdot \$)$. Further, we take

$$\omega_+(\hat{t}_{l,x,y,C,b,z}) = \omega_i(C) \cdot \omega_i(t_{q',yb}),$$

where $\omega_i(C)$ is the weight associated to the computation C of M_i.

Finally, if M_i contains the transition

$$t_{q',\$} : (q', \$) \to \mathsf{Restart},$$

then we add the transitions

$$\hat{t}_{l,x,\$,C} : (q_{l,x,\$,C}^{(i)}, \$) \to \mathsf{Restart}$$

to M_+, where we take

$$\omega_+(\hat{t}_{l,x,\$,C}) = \omega_i(C) \cdot \omega_i(t_{q',\$}).$$

This completes the proof also for the case that M_1 and M_2 are RRWW-automata.

□

Based on the above proof, the next result is easily obtained.

Theorem 4.3.2 ([OW16]). *For all alphabets Σ, all commutative semirings S, and all types X of restarting automata, the class of functions $\mathbb{F}(\mathsf{X}, \Sigma, S)$ is closed under the operation of scalar multiplication.*

Proof. Let S be a semiring, let M be a restarting automaton of some type X with input alphabet Σ, and let ω be a weight function for M. For each input $w \in \Sigma^*$, we have

$$f_\omega^M(w) = \sum_{C \in C_M(w)} \omega(C),$$

where $C_M(w)$ is the set of all accepting computations of M on input w, and $\omega(C)$ is the product of the weight of all transitions that are used in the accepting computation C. Let $t_1, t_2, \ldots, t_n$ be the transitions that are used during the computation C, then $\omega(C) = \omega(t_1) \cdot \omega(t_2) \cdot \ldots \cdot \omega(t_n)$.

For $s \in S$, we define a new weight function ω_s as follows:

$$\omega_s(t) = \begin{cases} s \cdot \omega(t), & \text{if } t \text{ is an accept transition,} \\ \omega(t), & \text{otherwise.} \end{cases}$$

As each computation $C \in C_M(w)$ uses exactly one accept transition, which ends the current computation, it follows that $\omega_s(C) = \omega(t_1) \cdot \omega(t_2) \cdot \ldots \cdot s \cdot \omega(t_n)$. Further, as S is commutative, we see that

$$\begin{array}{rcl} \omega_s(C) & = & s \cdot \omega(t_1) \cdot \omega(t_2) \cdot \ldots \cdot \omega(t_n) \\ & = & s \cdot \omega(C), \end{array}$$

which implies that

$$\begin{array}{rcl} f_{\omega_s}^M(w) & = & s \cdot \left(\sum_{C \in C_M(w)} \omega(C) \right) \\ & = & s \cdot f_\omega^M(w) \end{array}$$

holds. □

For each w(R)RWW-automaton $\mathcal{M} = (M, \omega)$, Theorem 4.3.2 also holds for semirings that are not commutative. Let $\mathcal{M}' = (M', \omega')$ be a w(R)RWW-automaton that first places a marking on the tape by replacing the prefix $¢a_1a_2$ of the input $w = a_1a_2w'$ by a special symbol of the form $[a_1, a_2]$, and ω' assigns the weight $s \in S$ to this rewrite transition. Then, on seeing the symbol $[a_1, a_2]$ M' simulates the computations of M starting from the configuration $q_0¢[a_1, a_2]w'\$$, and these transitions are assigned the same weight as the corresponding part of the computation of M, which can be done using the technique from the proof of Theorem 4.3.1. It follows that for each computation $C = \{t_1, t_2, \ldots, t_n\} \in C_M(w)$

$$\begin{aligned}\omega'(C) &= s \cdot \omega(t_1) \cdot \omega(t_2) \cdot \ldots \cdot \omega(t_n)\\ &= s \cdot \omega(C).\end{aligned}$$

Hence, for all $w \in \Sigma^*$ $f^{M'}_{\omega'}(w) = s \cdot f^M_\omega(w)$ holds. From this observation and from Theorem 4.3.1 we see that the sets of functions $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ and $\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$ are *semi-modules* over S (see, e.g., [SS78]). Finally, we derive the following additional closure property.

Theorem 4.3.3 ([OW16]).
For all alphabets Σ and all semirings S, $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ and $\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$ are closed under the operation of Cauchy product.

Proof. Let $S = (S, +, \cdot, 0, 1)$ be a semiring, let Σ be a finite alphabet, let $M_1 = (Q_1, \Sigma, \Gamma_1, ¢, \$, q_0^{(1)}, k_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, \Gamma_2, ¢, \$, q_0^{(2)}, k_2, \delta_2)$ be RWW- or RRWW-automata with input alphabet Σ, and let ω_1 and ω_2 be weight functions that map the transitions of M_1 and of M_2 to S. In order to prove that $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ is closed under the operation of Cauchy product, we construct an RWW- or RRWW-automaton M_c with input alphabet Σ and a weight function ω_c such that

$$f^{M_c}_{\omega_c}(w) = (f^{M_1}_{\omega_1} \cdot f^{M_2}_{\omega_2})(w) = \sum_{w=uv} \left(f^{M_1}_{\omega_1}(u) \cdot f^{M_2}_{\omega_2}(v)\right)$$

holds for all $w \in \Sigma^*$. To simplify this construction we can assume that M_1 and M_2 perform accept transitions only on the right sentinel \$, which can be done by using special states and additional move-right steps.

On input $w \in \Sigma^*$, the automaton M_c first guesses a factorization $w = u \cdot v$ of w. This will be realized in the first cycle by replacing the last symbol a of the prefix u and the first symbol b of the suffix v by an auxiliary symbol of the form $[a, b]$ for $a, b \in \Sigma$. In order to choose the factorization $w = \lambda \cdot w$ or $w = w \cdot \lambda$, the first two symbols a_1 and a_2 or the last two symbols b_1 and b_2 of w are replaced by the auxiliary symbol $[¢, a_1, a_2]$ or $[b_1, b_2, \$]$, respectively. As after a restart step M_c cannot remember that it has already chosen a factorization of w, it may find that it has chosen two (or more) factorizations. In that case, the corresponding computation halts immediately without accepting.

After having guessed a factorization $w = u \cdot v$, M_c simulates a computation of M_1 on the prefix u, where the special symbol of the form $[a, b]$ serves as the right end marker. If this computation of M_1 is accepting, then M_c replaces the special symbol $[a, b]$ by a new symbol of the form $[+, b]$, it deletes all letters to the left of this symbol in subsequent cycles, and then it simulates a computation of M_2 on the suffix v. Finally, M_c accepts if this computation of M_2 is also accepting. In the same way as in the proof of Theorem 4.3.1, we can define some auxiliary symbols and additional transitions for M_c to enable it to perform the above simulations. The special symbols of the form $[a, b]$, $[+, b]$, $[¢, a_1, a_2]$, or $[b_1, b_2, \$]$ can be dealt with using the same techniques.

It follows that, for each factorization $w = u \cdot v$, for each accepting computation of M_1 on input u, and for each accepting computation of M_2 on input v, the automaton M_c has exactly one accepting computation. By assigning weight 1 to all the transitions that are used in the guessing phase and to all transitions that are used in the phase between the simulation of M_1 and the simulation of M_2, by assigning weight $\omega_1(t_1)$ to all transitions of M_c that correspond to a transition t_1 of M_1, and by assigning weight $\omega_2(t_2)$ to all transitions of M_c that correspond to a transition t_2 of M_2, it can be shown that the equality

$$f^{M_c}_{\omega_c}(w) = (f^{M_1}_{\omega_1} \cdot f^{M_2}_{\omega_2})(w) = \sum_{w=uv} \left(f^{M_1}_{\omega_1}(u) \cdot f^{M_2}_{\omega_2}(v) \right)$$

holds for all input words $w \in \Sigma^*$. □

4.4 Concluding Remarks

We have introduced the *weighted restarting automaton* in order to express and study quantitative aspects of restarting automata and their computations. First, we have seen that all polynomials and finite sums of polynomials and exponential functions can be realized by wRWW-automata. Further, we have also studied the functions of the form $\hat{f}^M_\omega : \mathbb{N} \to S$ for a linearly ordered semiring S, and an upper bound for their growth is established. Finally, we have investigated the closure properties of the classes of functions that are represented by wRWW- and wRRWW-automata, and these results are summarized in Table 4.1.

We see that it is still open whether or not the classes $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ and $\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$ are closed under the operation of pointwise multiplication. Further, it is also unknown whether the classes $\mathbb{F}(\mathsf{X}, \Sigma, S)$ are closed under the above operations also for the types of restarting automata that cannot use

	$\mathbb{F}(\mathsf{RWW}, \Sigma, S)$	$\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$
Pointwise Addition	$\surd$	$\surd$
Pointwise Multiplication	?	?
Cauchy-Product	$\surd$	$\surd$
Scalar Multiplication	$\surd$	$\surd$

Table 4.1: Summay of closure properties of functions that are represented by wRWW- and wRRWW-automata for all input alphabets Σ and semirings S.

auxiliary symbols. Finally, for all types of restarting automata, it remains to characterize the classes of functions $\mathbb{F}(\mathsf{X}, \Sigma, S)$ and $\hat{\mathbb{F}}(\mathsf{X}, \Sigma, S)$ in a syntactic manner.

Chapter 5

Relations Computed by Weighted Restarting Automata

Originally, weighted restarting automata have been introduced to study quantitative aspects of computations of restarting automata. If we consider the special case that words over a given (output) alphabet are assigned as weights to the transitions of a restarting automaton, then the automaton can be extended to define a mapping from the words over its input alphabet into the semiring of formal languages over a given (output) alphabet. It means that each weighted restarting automaton computes a relation between input words and output words.

Actually, some extensions of restarting automata to transducers have been introduced over the years, and many authors have studied the relations computed in terms of these variants of restarting automata. First, a *characteristic language* or a *proper language* can be transformed into a relation by splitting the alphabet into an input and output part, and by using a projection onto the input and output alphabets (see, e.g., [MOP09, Ott10]). Let M be a restarting automaton with tape alphabet Γ and input alphabet $\Sigma \subseteq \Gamma$. Recall that a word $w \in \Gamma^*$ is called a *sentential form* that may consist of input and auxiliary symbols, and w is accepted by M, if there is an accepting computation which starts from the restarting configuration $q_0¢w\$$. By $L_C(M)$ we denote the characteristic language of M that consists of all sentential forms accepted by M. It is easily seen that $L(M) = L_C(M) \cap \Sigma^*$. Further, let $\mathsf{Pr}^\Sigma : \Gamma^* \to \Sigma^*$ be the projection that is defined as $a \to a$ for each $a \in \Sigma$ and $A \to \lambda$ for each $A \in \Gamma \smallsetminus \Sigma$. Then, the language $L_P(M) = \mathsf{Pr}^\Sigma(L_C(M))$ is called the proper language of M. Let $\Sigma = \Sigma' \cup \Delta$, where Σ' and Δ are input and output alphabets, respectively, and we assume that Σ' and Δ are disjoint. Based on the characteristic language and proper language of M, we define the *input/output relation*

$$Rel_{io}(M) = \{\, (u, v) \in \Sigma'^* \cup \Delta^* \mid \exists w \in L(M) : u = \mathsf{Pr}^{\Sigma'}(w) \text{ and } v = \mathsf{Pr}^{\Delta}(w) \,\},$$

and the *proper relation*

$$Rel_P(M) = \{ (u,v) \in \Sigma'^* \cup \Delta^* \mid \exists w \in L_C(M) : u = \mathsf{Pr}^{\Sigma'}(w) \text{ and } v = \mathsf{Pr}^{\Delta}(w) \}.$$

We see that with M two relations can be associated.

Parallel communicating systems of restarting automata (PC-systems for short) have been introduced in [VO12]. Further, a special model of PC-systems is given in [HOV10], where such a system consists of two det-mon-RRWW-automata (M_1, M_2). Given an input of the form (w_1, w_2), M_1 processes w_1 as the input for the whole system, and M_2 processes w_2 as the output. If both automata accept, we say that the PC-system accepts this word pair (w_1, w_2). In addition, a restarting transducer has been introduced in [HO12], which is a restarting automaton that produces an output word for each restart and accept transition. Obviously, each restarting transducer can be simulated by a weighted restarting automaton, and we will describe restarting transducers in Section 5.1.1 in detail.

The purpose of this chapter is to study the classes of relations that are computed by weighted restarting automata. This chapter consists of four sections. First, in Section 5.1 we introduce some basic notions. Further, in Section 5.2 we characterize the relation class artPDR that was already introduced in Section 2.2.2 by the monotone restarting transducers that are allowed to use auxiliary symbols. Then, in Section 5.3 we investigate the classes of relations that are computed by monotone weighted restarting automata, both in the deterministic and the nondeterministic cases. Finally, a summary and some problems for future work are given in Section 5.4.

5.1 Definitions and Examples

First, we present the definition of the relation computed by a weighted restarting automaton. Let $\mathcal{M} = (M, \omega)$ be a weighted restarting automaton, where $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is a restarting automaton, and ω is a weight function from the transitions of δ into a semiring S. Here we only consider the case that S is the semiring $S = (\mathbb{P}(\Delta^*), \cup, \cdot, \emptyset, \{\lambda\})$ of languages over Δ with the operations of union and product, that is, the weight of a transition of M can be any language over Δ. Let $\mathrm{AC}_M(w) = \{A_1, A_2, \ldots, A_m\}$ be the set of all accepting computations of M on input w. We assume that the computation $A_i \in \mathrm{AC}_M(w)$ $(1 \leq i \leq m)$ uses the transitions $t_{i,1}, t_{i,2}, \ldots, t_{i,n_i}$ of M. Then the weight of a transition $t_{i,j}$ $(1 \leq j \leq n)$ is a language $\omega(t_{i,j}) = L_{i,j}$ over Δ, and the weight of the computation A_i is $\omega(A_i) = L_{i,1} \cdot L_{i,2} \cdot \ldots \cdot L_{i,n_i} = \hat{L}_i \in \mathbb{P}(\Delta^*)$. Finally, $f_\omega^M(w) = \hat{L}_1 \cup \hat{L}_2 \cup \ldots \cup \hat{L}_m \in \mathbb{P}(\Delta^*)$ is the language over Δ that is associated by $\mathcal{M}$ to w, that is, f_ω^M is a transformation from Σ^* into $\mathbb{P}(\Delta^*)$. If $w \notin L(M)$, then $\mathrm{AC}_M(w) = \emptyset$, and accordingly, $f_\omega^M(w) = \emptyset$.

In this way, the weighted restarting automaton $\mathcal{M} = (M, \omega)$ on Σ yields the relation $Rel(\mathcal{M}) = \{\, (u, v) \mid u \in L(M), v \in f^M_\omega(u) \,\} \subseteq \Sigma^* \times \Delta^*$. By $\mathcal{R}(\mathsf{wX})$ we denote the class of relations that are computed by weighted restarting automata of type wX.

Actually, the above definition is very general as it allows to choose arbitrary languages over Δ as weights for the transitions of M. Therefore, the general model of weighted restarting automata is quite powerful, and below we will introduce some more restricted types of weighted restarting automata.

Definition 5.1.1 ([WO16a]). *A weighted restarting automaton $\mathcal{M} = (M, \omega)$ of type wX is called a* finitely weighted restarting automaton *(a $\mathsf{w_{FIN}X}$-automaton for short), if the weight function ω maps the transitions of M into a semiring of the form $S = (\mathbb{P}_{\text{fin}}(\Delta^*), \cup, \cdot, \emptyset, \{\lambda\})$. It is called a* word-weighted restarting automaton *(a $\mathsf{w_{word}X}$-automaton for short), if the weight of each transition t of M is of the form $\omega(t) = \{v\}$ for some $v \in \Delta^*$.*

It is rather obvious that $\mathcal{R}(\mathsf{w_{word}X}) \subseteq \mathcal{R}(\mathsf{w_{FIN}X}) \subsetneq \mathcal{R}(\mathsf{wX})$ for each type X of restarting automaton. In fact, for nondeterministic types of restarting automata, we have the following result, which obviously does not hold for deterministic types of restarting automata.

Proposition 5.1.1 ([WO16a]).
For all $X \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$, $\mathcal{R}(\mathsf{w_{word}X}) = \mathcal{R}(\mathsf{w_{FIN}X})$.

Proof. Let $M_1 = (Q_1, \Sigma, \Gamma, ¢, \$, q^{(1)}_0, k, \delta_1)$ be an RRWW-automaton, let ω_1 be a weight function that maps each transition $t \in \delta_1$ to a finite set $\omega_1(t) \in \mathbb{P}_{\text{fin}}(\Delta^*)$, and let $\mathcal{M}_1 = (M_1, \omega_1)$ be a $\mathsf{w_{FIN}RRWW}$-automaton. We will construct a $\mathsf{w_{word}RRWW}$-automaton $\mathcal{M}_2 = (M_2, \omega_2)$ such that $Rel(\mathcal{M}_1) = Rel(\mathcal{M}_2)$.

If $\omega_1(t) = \emptyset$ for some $t \in \delta_1$, then we can simply delete t from δ_1, as for any accepting computation $A \in \text{AC}_{M_1}(w)$ of M_1 that uses transition t, $\omega_1(A) = \emptyset$, which means that this computation does not contribute to the value $f^{M_1}_{\omega_1}(w)$. Thus, in the following we can assume without loss of generality that $\omega_1(t)$ is a non-empty finite subset of Δ^* for each transition $t \in \delta_1$.

Now we describe the construction of M_2 in detail. Let $\delta_1 = \{t_1, t_2, \ldots, t_m\}$, for each $1 \le i \le m$, let $\omega_1(t_i) = \{w_{i,1}, w_{i,2}, \ldots, w_{i,r_i}\}$, where $w_{i,j} \in \Delta^*$ and $1 \le j \le r_i$, and let $\Omega_1 = \bigcup_{i=1}^m \omega_1(t_i)$. As $\mathcal{M}_2$ is to be a $\mathsf{w_{word}RRWW}$-automaton, we will introduce r_i copies of transition t_i, one for each possible weight $w_{i,j}$. In addition, as a rewrite/restart transition t_i always leads to the initial state, and as an accept transition t_i does not enter any state at all, we need to distinguish between the r_i copies of such a transition by realizing them using different originating states. Below we define $\mathcal{M}_2 = (M_2, \omega_2)$. We take $M_2 = (Q_2, \Sigma, \Gamma, ¢, \$, q^{(2)}_0, k, \delta_2)$, where

$$\begin{aligned} Q_2 \;=\; & \{q^{(2)}_0\} \cup \{\, q(w, t_i, w_{i,j}) \mid q \in Q_1, w \in \Omega_1, i \in \{1, 2, \ldots, m\}, \\ & \text{and } j \in \{1, 2, \ldots, r_i\} \,\} \cup \{\, q_{\text{acc}}(w) \mid w \in \Omega_1 \,\}, \end{aligned}$$

and δ_2 and ω_2 are defined as follows, where we consider each transition t of M_1 in turn.

1. If t is a move-right transition of the form $(q_0^{(1)}, ¢u) \to (q_l, \mathsf{MVR})$, then we add the following transitions to δ_2:

   $$\hat{t}_{w,t_i,w_{i,j}} : (q_0^{(2)}, ¢u) \to (q_l(w, t_i, w_{i,j}), \mathsf{MVR})$$

   for all $w \in \omega_1(t), 1 \leq i \leq m$, and $1 \leq j \leq r_i$, that is, M_2 guesses a possible weight $w \in \omega_1(t)$, a possible next transition t_i, and a possible weight $w_{i,j} \in \omega_1(t_i)$. In addition, we take $\omega_2(\hat{t}_{w,t_i,w_{i,j}}) = \{w\}$.

2. If t is an accept transition of the form $(q_0^{(1)}, ¢u) \to \mathsf{Accept}$, then we add the following transitions to δ_2:

   $$\begin{array}{llcll} \hat{t}_{\mathrm{mv},w} : & (q_0^{(2)}, ¢u) & \to & (q_{\mathrm{acc}}(w), \mathsf{MVR}) & \text{for all } w \in \omega_1(t), \\ \hat{t}_{\mathrm{acc},w} : & (q_{\mathrm{acc}}(w), x) & \to & \mathsf{Accept} & \text{for all } w \in \omega_1(t) \text{ and} \\ & & & & \text{all possible words } x, \end{array}$$

   that is, M_2 guesses a possible weight $w \in \omega_1(t)$, enters a corresponding state by making a move-right step, and accepts then. In addition, we take $\omega_2(\hat{t}_{\mathrm{mv},w}) = \{\lambda\}$ and $\omega_2(\hat{t}_{\mathrm{acc},w}) = \{w\}$.

3. If t is a rewrite transition of the form $(q_0^{(1)}, ¢u) \to (q_l, ¢v)$, then we add the following transitions to δ_2:

   $$\hat{t}_{w,t_i,w_{i,j}} : (q_0^{(2)}, ¢u) \to (q_l(w, t_i, w_{i,j}), ¢v)$$

   for all $w \in \omega_1(t), 1 \leq i \leq m$, and $1 \leq j \leq r_i$, that is, M_2 guesses a possible weight $w \in \omega_1(t)$, a possible next transition t_i, and a possible weight $w_{i,j} \in \omega_1(t_i)$. In addition, we take $\omega_2(\hat{t}_{w,t_i,w_{i,j}}) = \{w\}$.

4. If t is a move-right transition of the form $(q, u) \to (q_l, \mathsf{MVR})$, then we add the following transitions to δ_2:

   $$t_{q(w,t,w')} : (q(w, t, w'), u) \to (q_l(w', t_h, w_{h,s}), \mathsf{MVR})$$

   for all $w \in \Omega_1, w' \in \omega_1(t), 1 \leq h \leq m$, and $1 \leq s \leq r_h$, that is, M_2 guesses a possible next transition t_h and a possible weight $w_{h,s} \in \omega_1(t_h)$. In addition, we take $\omega_2(t_{q(w,t,w')}) = \{w'\}$.

5. If t is an accept transition of the form $(q, u) \to \mathsf{Accept}$, then we add the following transitions to δ_2:

   $$t_{q(w,t,w')} : (q(w, t, w'), u) \to \mathsf{Accept} \text{ for all } w \in \Omega_1, w' \in \omega_1(t),$$

   and we take $\omega_2(t_{q(w,t,w')}) = \{w'\}$.

6. If t is a rewrite transition of the form $(q, u) \to (q_l, v)$, then we add the following transitions to δ_2:

$$t_{q(w,t,w')} : (q(w, t, w'), u) \to (q_l(w', t_h, w_{h,s}), v)$$

for all $w \in \Omega_1, w' \in \omega_1(t)$, $1 \le h \le m$, and $1 \le s \le r_h$, that is, M_2 guesses a possible next transition t_h and a possible weight $w_{h,s} \in \omega_1(t_h)$. In addition, we take $\omega_2(t_{q(w,t,w')}) = \{w'\}$.

7. If t is a restart transition of the form $(q, u) \to \mathsf{Restart}$, then we add the following transitions to δ_2:

$$t_{q(w,t,w')} : (q(w, t, w'), u) \to \mathsf{Restart} \text{ for all } w \in \Omega_1, w' \in \omega_1(t),$$

and we take $\omega_2(t_{q(w,t,w')}) = \{w'\}$.

Thus, for each state $q \in Q_1$, M_2 contains states $q(w, t_i, w_{i,j})$ that indicate that state q was entered by a transition of weight $\{w\}$, and that t_i is the next transition to be executed, and that its weight is to be $\{w_{i,j}\}$. Of course, if transition t_i is not applicable in the next configuration, for example, as the originating state of t_i is different from q, then the corresponding computation of M_2 gets stuck, that is, it halts without accepting. Now, if $(u, v) \in Rel(\mathcal{M}_1)$, then M_1 has an accepting computation A on input u such that $v \in \omega_1(A)$. Let $t_1, t_2, \ldots, t_n$ be the transitions that are used during the computation A, and then $\omega_1(A) = \omega_1(t_1) \cdot \omega_1(t_2) \cdot \ldots \cdot \omega_1(t_n)$. Further, let $v = v_1 v_2 \ldots v_n$, where $v_1 \in \omega_1(t_1), v_2 \in \omega_1(t_2), \ldots, v_n \in \omega_1(t_n)$. From the construction of M_2 and ω_2, for each t_i and $v_i \in \omega_1(t_i)$ $(1 \le i \le n)$, δ_2 contains a transition t'_i that simulates t_i, and $\omega_2(t'_i) = \{v_i\}$. Therefore, M_2 has an accepting computation A' on input u such that $\omega_2(A') = \{v_1 v_2 \ldots v_n\} = \{v\}$. Analogously, we can see that if M_2 has an accepting computation on input u which has weight $\{v\}$, then there exists an accepting computation A of M_1 on input u such that $v \in \omega_1(A)$. Thus, it follows that $Rel(\mathcal{M}_1) = Rel(\mathcal{M}_2)$.

Finally, if M_1 is an RX-automaton for $\mathsf{X} \in \{\lambda, \mathsf{W}, \mathsf{WW}\}$, then the above construction of M_2 can easily be adjusted to also yield an RX-automaton by combining rewrite and restart transitions. Notice that no restart operation can be executed in a restarting configuration, as each cycle must contain a rewrite operation. This completes this proof. □

In fact, the result above also holds for the corresponding versions of monotone restarting automata. As all rewrite transitions of M_2 that simulate a rewrite transition t of M_1 are executed in the same position as t, it is immediate that M_2 is monotone, if M_1 is. If a relation $R \subseteq \Sigma^* \times \Delta^*$ is the graph of a partial function, that is, for each $w \in \Sigma^*$, the set of images $R(w) = \{\, v \in \Delta^* \mid (w, v) \in R \,\}$ has cardinality at most one, then the following result can be shown.

Proposition 5.1.2 ([WO16a])**.** *Let $R \subseteq \Sigma^* \times \Delta^*$ be the graph of a partial function. If $R \in \mathcal{R}(\mathsf{wX})$ for some type of restarting automaton $X \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$, then $R \in \mathcal{R}(\mathsf{w}_{\mathsf{word}}\mathsf{X})$, too. Furthermore, this is also true for the corresponding deterministic and monotone versions of restarting automaton of type* X.

Proof. Let $R \subseteq \Sigma^* \times \Delta^*$ be a partial function such that $R = \mathit{Rel}(\mathcal{M})$ for some weighted restarting automaton $\mathcal{M} = (M, \omega)$. If $\mathcal{M}$ is not a word-weighted restarting automaton, then there is a transition t of M such that $\omega(t) \subseteq \Delta^*$ is not of cardinality 1.

If $\omega(t) = \emptyset$, then $\omega(A) = \emptyset$ for each computation of M that uses transition t. Accordingly, this computation does not contribute to the value $f_\omega^M(u)$ for any input u. Thus, we can simply delete this transition. By doing this for all transitions of this form, we obtain a weighted restarting automaton $\mathcal{M}_1 = (M_1, \omega)$ that is a subautomaton of $\mathcal{M}$ of the same type as $\mathcal{M}$ such that $\omega(t_1) \neq \emptyset$ for all transitions t_1 of M_1 and $\mathit{Rel}(\mathcal{M}_1) = \mathit{Rel}(\mathcal{M})$.

Now, if $|\omega(t_1)| > 1$ for a transition t_1 of M_1, then $|\omega(A)| > 1$ for each computation A of M_1 that uses this transition. It follows that no accepting computation of M_1 must use this transition. Thus, we can simply delete this transition. By doing this for all transitions of this form, we obtain a word-weighted restarting automaton $\mathcal{M}_2 = (M_2, \omega)$ that is a subautomaton of $\mathcal{M}$, that is of the same type as $\mathcal{M}$, and for which $\mathit{Rel}(\mathcal{M}_2) = \mathit{Rel}(\mathcal{M})$ holds. $\square$

In [Hun13] it is shown that for some types X and Y of restarting automata, if X is a restricted type of Y and $\mathcal{L}(\mathsf{X}) \subsetneq \mathcal{L}(\mathsf{Y})$, then this inclusion relation also holds for the classes of relations that are computed by restarting transducers of the corresponding types. For example, an RR-automaton is a restricted version of an RRW-automaton, and $\mathcal{L}(\mathsf{RR}) \subsetneq \mathcal{L}(\mathsf{RRW})$, thus the relation class $\mathcal{R}(\mathsf{RR\text{-}Td})$ is a proper subclass of the relation class $\mathcal{R}(\mathsf{RRW\text{-}Td})$. The main idea is to extend an X-automaton M to an X-transducer T_M by assigning the symbol 1 as the output to the accept transition and the symbol λ to all other transitions, and then the language $L(M)$ is also extended to a relation $\mathit{Rel}(T_M)$, which can be seen as a semi-characteristic function for $L(M)$. If we choose a language $L \in \mathcal{L}(\mathsf{X}) \smallsetminus \mathcal{L}(\mathsf{Y})$, then there exists a restarting transducer T of type X such that $\mathit{Rel}(T) = \{(w, 1) \mid w \in L\}$, while the relation $\mathit{Rel}(T)$ is not computable by any Y-transducer. This result can be carried over to the classes of relations that are computed by weighted restarting automata, and thus we obtain the following result.

Proposition 5.1.3. *Let* $\mathsf{X}, \mathsf{Y} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*, and* X *is a restricted type of* Y*. If* $\mathcal{L}(\mathsf{X}) \subsetneq \mathcal{L}(\mathsf{Y})$*, then* $\mathcal{R}(\mathsf{pX}) \subsetneq \mathcal{R}(\mathsf{pY})$ *for each prefix* $\mathsf{p} \in \{\mathsf{w}, \mathsf{w}_{\mathsf{FIN}}, \mathsf{w}_{\mathsf{word}}\}$.

Now we present some examples of relations that are computed by weighted restarting automata.

Example 5.1.1. *Let $M_1 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be the* det-mon-R-*automaton that is defined by taking $Q = \{q_0\}$, $\Gamma = \Sigma = \{a\}$, and $k = 2$, where δ is defined as follows:*

$$\begin{array}{lll} t_1: & (q_0, ¢a) & \to & (q_0, \mathsf{MVR}), \\ t_2: & (q_0, aa) & \to & (q_0, \mathsf{MVR}), \\ t_3: & (q_0, a\$) & \to & (q_r, \$), \\ t_4: & (q_r, x) & \to & \mathsf{Restart} \textit{ for all admissible } x, \\ t_5: & (q_0, ¢\$) & \to & \mathsf{Accept}. \end{array}$$

It is easily seen that in each cycle M_1 moves to the right end of the tape, and on seeing the right sentinel \$ it removes the last a-symbol and restarts. This process is repeated all the way until the configuration ¢\$ is reached. Therefore, M_1 accepts all input words that consist of a finite number of a-symbols, that is, $L(M_1) = \{ a^n \mid n \geq 0 \}$.

Let $(\mathbb{P}_{\text{fin}}(\Delta^*), \cup, \cdot, \emptyset, \{\lambda\})$ be the semiring of finite languages over $\Delta = \{c\}$, let ω_1 be the weight function that assigns the set $\{c\}$ to the move-right transitions t_1 and t_3, and that assigns the set $\{\lambda\}$ to all other transitions, and let $\mathcal{M}_1 = (M_1, \omega_1)$. Then, a cycle from the restarting configuration $¢a^l\$$ contains l move-right steps, and thus this many c-symbols are produced as weight during this cycle. Therefore, given an input word a^n, the weight of the accepting computation is the set $\{c^r\}$, where

$$r = \sum_{i=1}^{n} i = \frac{1}{2}(n+1)n.$$

It follows that

$$f_{\omega_1}^{M_1}(w) = \begin{cases} \{c^{\frac{1}{2}(n+1)n}\}, & \textit{for } w = a^n,\ n \geq 0, \\ \emptyset, & \textit{for } w \notin L(M_1), \end{cases}$$

and hence, $Rel(\mathcal{M}_1) = \{ (a^n, c^{\frac{1}{2}(n+1)n}) \mid n \geq 0 \}$.

Example 5.1.2. *Let $M_2 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be the* det-mon-R-*automaton that is defined by taking $Q = \{q_0, q_r\}$, $\Gamma = \Sigma = \{a, b\}$, and $k = 2$, where δ is defined as follows:*

$$\begin{array}{rllll} t_{1,x_1}: & (q_0, ¢x_1) & \to & (q_0, \mathsf{MVR}) & \textit{for all } x_1 \in \Sigma, \\ t_{2,x_2x_3}: & (q_0, x_2x_3) & \to & (q_0, \mathsf{MVR}) & \textit{for all } x_2, x_3 \in \Sigma, \\ t_{3,x_4}: & (q_0, x_4\$) & \to & (q_r, \$) & \textit{for all } x_4 \in \Sigma, \\ t_4: & (q_r, x) & \to & \mathsf{Restart} & \textit{for all admissible } x, \\ t_5: & (q_0, ¢\$) & \to & \mathsf{Accept}. & \end{array}$$

It is easily seen that M_2 accepts all input words that consist of a finite number of a- and b-symbols, that is, $L(M_2) = \{ w \mid w \in \{a,b\}^* \} = \Sigma^*$.

Let $(\mathbb{P}_{\text{fin}}(\Delta^*), \cup, \cdot, \emptyset, \{\lambda\})$ be the semiring of finite languages over $\Delta = \Sigma = \{a,b\}$, and let ω_2 be the weight function such that $\omega_2(t_{3,x_4}) = \{x_4\}$, and all other transitions have the weight $\{\lambda\}$, and let $\mathcal{M}_2 = (M_2, \omega_2)$. It follows easily that

$$f^{M_2}_{\omega_2}(w) = \begin{cases} \{w^R\}, & \textit{for } w \in \{a,b\}^*, \\ \emptyset, & \textit{for } w \notin L(M_1), \end{cases}$$

and hence, $Rel(\mathcal{M}_2) = \{ (w, w^R) \mid w \in \{a,b\}^* \}$.

From Example 5.1.1 we see that a deterministic monotone word-weighted R-automaton can already compute a relation that is not linearly bounded. By $\mathcal{R}_{lb}(\mathsf{wX})$ we denote the class of linearly bounded relations in $\mathcal{R}(\mathsf{wX})$. Actually, some examples of relations computed by weighted restarting automata are already presented in Example 3.3.1 and 3.3.2.

5.1.1 Restarting Transducers

In this section we recall the notion of *restarting transducer* that was introduced in [Hun13]. In analogy to finite transducers and pushdown transducers, a restarting transducer is a restarting automaton that is equipped with an additional output function which gives an output word for each restart and each accept transition. Formally, a restarting transducer is defined by a 9-tuple $T = (Q, \Sigma, \Delta, \Gamma, \cent, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is an input alphabet, Γ is a tape alphabet, Δ is an output alphabet, $\cent, \$ \notin \Gamma$ are the left and right markers of the tape, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the read/write window, and

$$\delta : Q \times \mathcal{PC}^{(k)} \to \mathbb{P}_{\text{fin}}(Q \times (\{\mathsf{MVR}\} \cup \mathcal{PC}^{\leq(k-1)}) \cup (\{\mathsf{Restart}, \mathsf{Accept}\} \times \Delta^*))$$

is the transition function.

A *configuration* of T is described by a pair $(\alpha q \beta, z)$, where $\alpha q \beta$ is a configuration of the underlying restarting automaton and $z \in \Delta^*$ is an output word. Obviously, for an input word $w \in \Sigma^*$, the initial configuration is $(q_0 \cent w \$, \lambda)$. A restarting configuration can be described by $(q_0 \cent w' \$, v)$, where $w' \in \Gamma^*$ and $v \in \Delta^*$ is the current output. Finally, an accepting configuration is of the form (Accept, z), where $z \in \Delta^*$ is an output word. Accordingly, an accepting computation of T can be described as

$$\begin{aligned} (q_0 \cent w \$, \lambda) &\vdash^c_T (q_{i_1} \cent w_1 \$, v_1) \vdash^c_T \ldots \vdash^c_T (q_{i_m} \cent w_m \$, v_1 \cdots v_m) \\ &\vdash^*_T (\mathsf{Accept}, v_1 \cdots v_m v_{m+1}). \end{aligned}$$

The relation that is computed by T is defined as

$$Rel(T) = \{(w, z) \in \Sigma^* \times \Delta^* \mid (q_0 \cent w \$) \vdash^*_T (\mathsf{Accept}, z)\}.$$

By $\mathcal{R}(\mathsf{X\text{-}Td})$ we denote the class of relations that are computed by restarting transducers of type X. It is easily seen that restarting transducers are a special type of word-weighted restarting automata.

We close this section with an example of a relation that is computed by a restarting transducer.

Example 5.1.3. *Let $T_1 = (Q, \Sigma, \Delta, \Gamma, ¢, \$, q_0, k, \delta)$ be an* $\mathsf{RR\text{-}Td}$*, where $Q = \{q_0, q_a, q_b, q_a^r, q_b^r, q_c^r\}$, $\Sigma = \Delta = \Gamma = \{a, b, c\}$, $k = 2$, and δ is defined as follows:*

$$\begin{array}{rlcll}
t_1: & (q_0, ¢a) & \to & (q_a^r, ¢), & \\
t_2: & (q_a^r, -) & \to & (\mathsf{Restart}, a), & \\
t_3: & (q_0, ¢b) & \to & (q_b^r, ¢), & \\
t_{4,x_1,x_2}: & (q_b^r, x_1x_2) & \to & (q_b^r, \mathsf{MVR}) & \textit{for } x_1, x_2 \in \{b, c\}, \\
t_{5,x}: & (q_b^r, x\$) & \to & (\mathsf{Restart}, b) & \textit{for } x \in \{b, c\}, \\
t_{6,x}: & (q_0, ¢x) & \to & (q_a, \mathsf{MVR}) & \textit{for } x \in \{b, c\}, \\
t_{7,x_1,x_2}: & (q_a, x_1x_2) & \to & (q_a, \mathsf{MVR}) & \textit{for } x_1, x_2 \in \{b, c\}, \\
t_{8,x}: & (q_a, xa) & \to & (q_a^r, x) & \textit{for } x \in \{b, c\}, \\
t_9: & (q_0, ¢c) & \to & (q_b, \mathsf{MVR}), & \\
t_{10}: & (q_b, cc) & \to & (q_b, \mathsf{MVR}), & \\
t_{11}: & (q_b, cb) & \to & (q_b^r, c), & \\
t_{12}: & (q_0, ¢c) & \to & (q_c^r, ¢), & \\
t_{13}: & (q_c^r, cc) & \to & (q_c^r, \mathsf{MVR}), & \\
t_{14}: & (q_c^r, c\$) & \to & (\mathsf{Restart}, c), & \\
t_{15}: & (q_0, ¢\$) & \to & (\mathsf{Accept}, \lambda). &
\end{array}$$

It is easily seen that T_1 accepts all the words over the alphabet Σ and it proceeds as follows. First, T_1 removes a-symbols from the input and produces them in restart steps as output. Then, it proceeds to do the same for b- and c-symbols of the input. Thus, the relation that is computed by T_1 is $Rel(T_1) = \{\, (w, a^{|w|_a} b^{|w|_b} c^{|w|_c}) \mid w \in \{a, b, c\}^* \,\}$. For example, on the input word $bccabbca$, T_1 has the following accepting computation:

$$\begin{array}{lllll}
(q_0¢bccabbca\$, \lambda) & \vdash^*_{\mathsf{MVR}} & (¢bcq_acabbca\$, \lambda) & \vdash_{\mathsf{Rewrite}} & (¢bccq_a^rbbca\$, \lambda) \\
 & \vdash_{\mathsf{Restart}} & (q_0¢bccbbca\$, a) & \vdash^*_{\mathsf{MVR}} & (¢bccbbq_aca\$, a) \\
 & \vdash_{\mathsf{Rewrite}} & (¢bccbbcq_a^r\$, a) & \vdash_{\mathsf{Restart}} & (q_0¢bccbbc\$, aa) \\
 & \vdash_{\mathsf{Rewrite}} & (¢q_b^rccbbc\$, aa) & \vdash^*_{\mathsf{MVR}} & (¢ccbbq_b^rc\$, aa) \\
 & \vdash_{\mathsf{Restart}} & (q_0¢ccbbc\$, aab) & \vdash^*_{\mathsf{MVR}} & (¢cq_bcbbc\$, aab) \\
 & \vdash_{\mathsf{Rewrite}} & (¢ccq_b^rbc\$, aab) & \vdash_{\mathsf{MVR}} & (¢ccbq_b^rc\$, aab) \\
 & \vdash_{\mathsf{Restart}} & (q_0¢ccbc\$, aabb) & \vdash^*_{\mathsf{MVR}} & (¢cq_bcbc\$, aabb) \\
 & \vdash_{\mathsf{Rewrite}} & (¢ccq_b^rc\$, aabb) & \vdash_{\mathsf{Restart}} & (q_0¢ccc\$, aabbb) \\
 & \vdash^c_{T_1} & (q_0¢cc\$, aabbbc) & \vdash^c_{T_1} & (q_0¢c\$, aabbbcc) \\
 & \vdash^c_{T_1} & (q_0¢\$, aabbbccc) & \vdash_{\mathsf{Accept}} & (\mathsf{Accept}, aabbbccc).
\end{array}$$

It is worth to note that T_1 is nondeterministic, so that it can guess whether there is still an a-symbol (or a b-symbol) on the tape when removing a b-symbol (or a c-symbol)

Obviously, the relation $Rel(\mathcal{M}_2)$ given in Example 5.1.2 can also be computed by a restarting transducer. Every relation that is computed by a restarting transducer is *linearly bounded*, as a restarting transducer outputs symbols only during restart and accept steps, and any computation on an input of length n contains at most $n+1$ such steps. This means that for a restarting transducer T and for each pair $(u,v) \in Rel(T)$ $(|u| \neq \lambda)$, there is a constant c, such that $|v| \leq c \cdot |u|$. Further, by Example 5.1.1 already a $\mathsf{det\text{-}mon\text{-}w_{word}R}$-automaton can compute a relation that is not linearly bounded. Hence, we can easily establish the following proper inclusion.

Corollary 5.1.1.
For all $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$, $\mathcal{R}(\mathsf{X\text{-}Td}) \subsetneq \mathcal{R}(\mathsf{w_{word}X})$.

In addition, as each rewrite step of a restarting transducer must be strictly length-reducing, a restarting transducer is not able to produce an output without consuming any input symbol. Therefore, the next result is obtained.

Corollary 5.1.2 ([Hun13])**.** *There is no restarting transducer that computes the relation* $R = \{(\lambda, a^n) \mid n \geq 0\}$.

5.2 Restarting Transducers and Pushdown Relations

In this section we compare the classes of relations that are computed by restarting transducers to each other, and we relate them to various types of pushdown relations. First, we prove that $\mathsf{mon\text{-}RWW\text{-}Td}$s and $\mathsf{mon\text{-}RRWW\text{-}Td}$s characterize the class artPDR by establishing the following two lemmas.

Lemma 5.2.1 ([WO16a])**.** $\mathsf{artPDR} \subseteq \mathcal{R}(\mathsf{mon\text{-}RWW\text{-}Td})$.

Proof. Let $R \subseteq \Sigma^* \times \Delta^*$ be the relation that is computed by the almost-realtime PDT $T = (P, \Sigma, \Gamma, \Delta, p_0, Z_0, F, \delta)$. We simulate T by a $\mathsf{mon\text{-}RWW\text{-}Td}$ using a construction from [KMO10b].

Let $l = \max\{\, |\gamma| \mid \exists (p', \gamma, v) \in \delta(p, a, A) \,\}$, and let $\Gamma' = \Gamma'_1 \cup \Gamma'_2$, where $\Gamma'_1 = \{\, (x, p) \mid x \in \Gamma^+,\ |x| \leq 2l, \text{ and } p \in P \,\}$ and $\Gamma'_2 = \{\, (y) \mid y \in \Gamma^{2l} \,\}$. Thus, a symbol $(x,p) \in \Gamma'_1$ encodes a word $x \in \Gamma^*$ of length at most $2l$ together with a state p of T, while a symbol $(y) \in \Gamma'_2$ encodes a word $y \in \Gamma^*$ of length $2l$. Finally, let $M = (Q_M, \Sigma, \Gamma', \Delta, ¢, \$, q_0, 4, \delta')$ be the $\mathsf{RWW\text{-}Td}$ that simulates T as follows.

In each cycle M simulates two steps of T, using a symbol from Γ_1' to encode the current state of T. Assume that an accepting computation of T on input $w = a_0a_1a_2 \dots a_n$ begins by first applying the transition

$$(p_1, B_1 \dots B_{m_1}C_1, v_1) \in \delta(p_0, a_0, Z_0)$$

and then the transition

$$(p_2, B_{m_1+1} \dots B_{m_1+m_2}C_2, v_2) \in \delta(p_1, a_1, C_1).$$

As $m_1 < l$ and $m_2 < l$, $|B_1 \dots B_{m_1}B_{m_1+1} \dots B_{m_1+m_2}C_2| < 2l$. Accordingly, starting with the input configuration corresponding to input w, M can execute the rewrite step $(q_0, ¢a_0a_1a_2) \to (q_0, ¢(xC_2, p_2)a_2)$ (and then it must restart), where $x := B_1 \dots B_{m_1}B_{m_1+1} \dots B_{m_1+m_2}$, producing the output v_1v_2.

Assume that by executing the next two steps, the PDT T reaches the configuration

$$(p_4, a_4 \dots a_n, B_1 \dots B_{m_1}B_{m_1+1} \dots B_{m_1+m_2-1}x_1, v_1v_2v_3v_4),$$

that is, the input symbols a_2a_3 are read, the state changes to p_4, the two topmost symbols $B_{m_1+m_2}C_2$ on the pushdown are replaced by the string $x_1 \in \Gamma^{\leq 2l}$, and the output v_3v_4 is produced. If $m_1 + m_2 - 1 + |x_1| \leq 2l$, then M rewrites $(xC_2, p_2)a_2a_3a_4$ into $(x', p_4)a_4$, where $x' = B_1 \dots B_{m_1}B_{m_1+1} \dots B_{m_1+m_2-1}x_1$. If $m_1 + m_2 - 1 + |x_1| > 2l$, then M rewrites $(xC_2, p_2)a_2a_3a_4$ into $(x')(x'', p_4)a_4$, where $x'x'' = B_1 \dots B_{m_1}B_{m_1+1} \dots B_{m_1+m_2-1}x_1$ and $|x'| = 2l$.

Continuing in this way it follows that the tape content of M is always of the form

$$(x_1)(x_2) \dots (x_i)(x_{i+1}, p)a_ja_{j+1} \dots a_n,$$

where $(x_{i+1}, p) \in \Gamma_1'$, and $(x_1), (x_2), \dots, (x_i) \in {\Gamma_2'}^*$. Here $(x_1)(x_2) \dots (x_i)x_{i+1}$ encodes the current content of the pushdown of T, p is the current state of T, and $a_ja_{j+1} \dots a_n$ is the suffix of the input that T still has to read. As long as $j < n - 1$, M can simulate the next two non-λ-steps of T by rewriting the factor $(x_i)(x_{i+1}, p)a_ja_{j+1}$. Let $x_{i+1} = \alpha_1C$. If the topmost symbol C of the pushdown of T is replaced by the factor α_2 after processing the symbols a_ja_{j+1}, and $|\alpha_1\alpha_2| > 2l$, then M rewrites $(x_i)(x_{i+1}, p)a_ja_{j+1}$ into $(x_i)(x_{i+1}')(x_{i+2}, p')$, where $x_{i+1}'x_{i+2} = \alpha_1\alpha_2$; if $|\alpha_1\alpha_2| \leq 2l$, then M rewrites $(x_i)(x_{i+1}, p)a_ja_{j+1}$ into $(x_i)(x_{i+2}, p')$, where $x_{i+2} = \alpha_1\alpha_2$; if the factor x_{i+1} is popped from the pushdown after processing the symbols a_ja_{j+1}, then M rewrites $(x_i)(x_{i+1}, p)a_ja_{j+1}$ into (x_{i+2}, p').

If T executes a λ-step, then it changes its state, pops a symbol from the pushdown, and produces an output syllable. However, each rewrite transition of M must be length-reducing. In order to solve this problem, we must combine up to $2l$ λ-steps of T (or several λ-steps together with the next or the next two non-λ-steps) into a single simulation step of M. Assume that on the tape content

$$(x_1)(x_2)\dots(x_i)(x_{i+1},p)a_ja_{j+1}\dots a_n,$$

the computation of T continues by applying the following λ-steps

$$\begin{array}{rcl}(q_1,\lambda,v'_1) & \in & \delta(p,\lambda,C_1),\\ (q_2,\lambda,v'_2) & \in & \delta(q_1,\lambda,C_2),\\ & \vdots & \\ (q_s,\lambda,v'_s) & \in & \delta(q_{s-1},\lambda,C_s).\end{array}$$

For simulating these λ-steps, we must distinguish between several cases.

1. If $x_{i+1} = \alpha C_s \dots C_2C_1$ for $\alpha = \alpha' C \in \Gamma^+$, and $C \in \Gamma$, that is, $s < |x_{i+1}|$, we combine these λ-steps with the next two non-λ-steps. This means that after executing the λ-steps above, the symbols $C_1, C_2, \dots, C_s$ are popped from the pushdown, and the topmost symbol on the pushdown is C. Assume that the computation continues by first performing the non-λ-transition

$$(q_j, B_1B_2\dots B_{r_1}C'_1, v_j) \in \delta(q_s, a_j, C),$$

and then the non-λ-transition

$$(q_{j+1}, B_{r_1+1}B_{r_1+2}\dots B_{r_1+r_2-1}C'_2, v_{j+1}) \in \delta(q_j, a_{j+1}, C'_1).$$

As $r_1 < l$ and $r_2 < l$, $|B_1B_2\dots B_{r_1+r_2-1}C'_2| < 2l$. It is also clear that $|\alpha| < 2l$. If $|\alpha B_1B_2\dots B_{r_1+r_2-1}C'_2| < 2l$, then M simulates the transitions above by rewriting the four symbols $(x_i)(x_{i+1},p)a_ja_{j+1}$ into the factor $(x_i)(\alpha B_1B_2\dots B_{r_1+r_2-1}C'_2, q_{j+1})$; otherwise, M rewrites the four symbols $(x_i)(x_{i+1},p)a_ja_{j+1}$ into the factor $(x_i)(\alpha_1)(\alpha_2,q_{j+1})$, where $\alpha_1\alpha_2 = \alpha B_1B_2\dots B_{r_1+r_2-1}C'_2$, and $|\alpha_1| = 2l$.

2. If there exists an index m such that $x_{i+1} = C_m \dots C_2C_1$, and $x_i = \alpha C_s \dots C_{m+1}$, where $1 \leq m < s$ and $\alpha \in \Gamma^+$, that is, $s > |x_{i+1}|$, we combine the λ-steps above by rewriting the factor $(x_i)(x_{i+1},p)$ into the symbol (α, q_s). Analogously, if $s > 2l$, M can simulate these λ-steps by two or more rewrite steps.

This simulation continues until either T rejects (and then M rejects as well), or until $j = n-1$ is reached. At that point M can detect whether T will accept or reject, and it will then act likewise. It follows that M is monotone, and that $Rel(M) = R$ holds. □

Lemma 5.2.2 ([WO16a])**.** $\mathcal{R}(\textsf{mon-RRWW-Td}) \subseteq \textsf{artPDR}$.

Proof. Let $M = (Q, \Sigma, \Gamma, \Delta, ¢, \$, q_0, k, \delta)$ be a mon-RRWW-Td. Using the simulation technique from [JMPV99] it can be shown that M can be simulated by a PDT T. Let $¢uqvw\$$ be a rewrite configuration in an accepting computation of M, and assume that M now executes the rewrite step $(q', v') \in \delta(q, v)$. Then the next cycle starts from the restarting configuration $q_0¢uv'w\$$, and as M is monotone, the next rewrite operation is performed within a suffix of $uv'w$ of length at most $|vw|$. Thus, the prefix uv' can be stored on the pushdown of T, while the input contains the suffix w still unread. After performing this rewrite step, the RRWW-Td M moves to the right. In order to simplify the discussion, we assume without loss of generality that M only restarts and produces its output at the right end of the tape. As T cannot scan its input completely each time it simulates a rewrite step, it guesses the output z produced by M at the end of the current cycle, and it keeps the state q' reached by the above rewrite step and the output z guessed in its finite-state control. When it processes the remaining part of w, it updates this state information. Finally, when w has been processed completely, then T checks whether all the states of M stored in its finite-state control correspond to restart steps and to the corresponding output strings.

In fact, as M is monotone, it can be checked quite easily that T is almost-realtime. Actually, whenever T executes a λ-transition, then this means that it simulates a rewrite step of M that has exactly the same right distance as the previous rewrite step. As each rewrite step of M is strictly length-reducing, this implies that the prefix of the tape inscription of M that is stored on the pushdown of T is reduced in length, which means that T pops a symbol (or several symbols) from its pushdown. In addition, whenever T simulates a rewrite step of M, then it must remember the state q' that M enters through this rewrite step and the output z that M will produce in the current cycle. Luckily, there are only finitely many pairs of the form (q', z) of M, and hence, T can actually store all the pairs occurring in the computation being simulated in its finite-state control. □

As $\mathcal{R}$(mon-RWW-Td) $\subseteq \mathcal{R}$(mon-RRWW-Td) obviously holds, Lemma 5.2.1 and 5.2.2 imply the next result.

Theorem 5.2.1. $\mathcal{R}$(mon-RWW-Td) $= \mathcal{R}$(mon-RRWW-Td) $=$ artPDR.

In the following we consider the deterministic restarting transducers. We will show that the class of (almost-realtime or linearly bounded) DPDF can be characterized by det-mon-RWW-Tds. Again we prove this result through two lemmas.

Lemma 5.2.3 ([WO16a])**.** artDPDF $\subseteq \mathcal{R}$(det-mon-RWW-Td).

Proof. As shown in the proof of Lemma 5.2.1, an almost-realtime PDT can be simulated by a mon-RWW-Td. If the almost-realtime PDT is deterministic, then the simulation can be performed by a mon-RWW-Td that is deterministic, too. This yields the inclusion above. □

Lemma 5.2.4 ([WO16a])**.** $\mathcal{R}(\textsf{det-mon-RWW-Td}) \subseteq \textsf{artDPDF}$.

Proof. As shown in the proof of Lemma 5.2.2, a mon-RRWW-Td M can be simulated by a PDT T that is almost-realtime. Even if M is deterministic, the simulation by T still requires some nondeterminism, as each time that T simulates a rewrite step of M, it must guess (and remember) the output that M will produce at the end of the corresponding cycle. However, if M is just a det-mon-RWW-Td, then M restarts immediately after a rewrite step, which means that in each cycle it also produces its output at that point. It follows that M can be simulated by a DPDT that is almost-realtime. □

The equality $\textsf{artDPDF} = \mathcal{R}(\textsf{det-mon-RWW-Td})$ can be obtained because of Lemmas 5.2.3 and 5.2.4. Further, recall that $\textsf{artDPDF} = \textsf{lbDPDF} = \textsf{DPDF}$ is given in Theorem 2.2.4. Hence, the next result can be established.

Theorem 5.2.2. $\textsf{artDPDF} = \textsf{lbDPDF} = \textsf{DPDF} = \mathcal{R}(\textsf{det-mon-RWW-Td})$.

From Theorem 5.2.1 we have seen that the nondeterministic mon-RWW- and mon-RRWW-Tds are equally expressive. However, for the corresponding deterministic restarting transducers, this is not the case. To prove the fact that the det-mon-RRWW-Td is strictly more expressive than the det-mon-RWW-Td, we consider the following example function $\gamma_1 : \{a,b,c,d\}^* \to \{a,b\}^*$ that is defined as follows:

$$\gamma_1(w) = \begin{cases} a^k, & \text{if } w = a^k b^k c,\ k \geq 1, \\ b^k, & \text{if } w = a^k b^k d,\ k \geq 1, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Lemma 5.2.5 ([WO16a])**.** $\gamma_1 \notin \mathcal{R}(\textsf{det-mon-RWW-Td})$.

Proof. Because of Theorem 5.2.2 it suffices to show that γ_1 does not belong to the class DPDF. We assume that there exists a well-behaved DPDT T that computes γ_1. Given an input of the form $a^k b^k x$, where $x \in \{c,d\}$, T cannot begin to produce non-empty output before it has scanned the symbol x. Thus, T works in three phases:

1. First it processes the prefix $a^k b^k$ producing empty output.

2. Then it reads the symbol x, possibly producing a prefix y^r of its output y^k, where $y = a$, if $x = c$, and $y = b$, if $x = d$, and r is a constant.

3. Finally, T performs a sequence of λ-steps during which it produces the remaining part y^{k-r} of the output.

Now from T we can construct a DPDA A that works as follows. While reading the prefix a^kb^k, A simulates T step by step. When it encounters the symbol x, then A enters a special state $q_{2,x}$ that indicates that the last two phases of T's computation have been reached. From this state it is required to read the word y^r from its input tape, and then it continues to simulate the third phase of T's computation, where instead of producing output y^{k-r}, A expects to read input y^{k-r}. It follows that $L(A) = \{\, a^kb^kca^k, a^kb^kdb^k \mid k \geq 1 \,\}$. As this language is not context-free, we have a contradiction. This shows actually that $\gamma_1 \notin \mathsf{DPDF}$. □

Based on Lemma 5.2.5 the following result can be obtained.

Theorem 5.2.3. $\mathcal{R}(\mathsf{det\text{-}mon\text{-}RWW\text{-}Td}) \subsetneq \mathcal{R}(\mathsf{det\text{-}mon\text{-}RRWW\text{-}Td})$.

Proof. It is clear that γ_1 can be computed by a det-mon-RRWW-Td, as such a transducer may scan its tape completely in each cycle. Thus, it can remove a factor ab, move to the end of the tape, and restart, producing an a-symbol as output, if the last letter is a c-symbol, or producing a b-symbol as output, if the last letter is a d-symbol. □

5.3 Relations Computed by Monotone Weighted Restarting Automata

In the previous section we have shown that the class of relations that are computed by mon-RWW-Tds and mon-RRWW-Tds coincides with the relation class artPDR, and that the function class DPDF can be characterized by det-mon-RWW-Tds, while det-mon-RRWW-Tds even compute some functions that are not in DPDF. Here we study the question of whether (word-)weighted RWW- and RRWW-automata that are monotone are more expressive than the corresponding transducers.

We have seen that the det-mon-$\mathsf{w_{word}}$R-automaton given in Example 5.1.1 already computes a relation that is not linearly bounded. Thus, we begin this investigation by studying the relation between the classes $\mathcal{R}_{lb}(\mathsf{mon\text{-}w_{word}RWW})$ and $\mathcal{R}(\mathsf{mon\text{-}RWW\text{-}Td})$. For this purpose, let $\tau_0 : \{a,b\}^* \to \{a,b\}^*$ be the function $\tau_0(w) = ww$. As the domain of τ_0 is the regular set $\{a,b\}^*$, and as its range is the copy language $L_{\text{copy}} = \{\, ww \mid w \in \{a,b\}^* \,\}$, which is not even growing context-sensitive (see, e.g., [BO98, Lau88]), we see that τ_0 is not a pushdown relation. However, we have the following result.

Lemma 5.3.1 ([WO16a]). $\tau_0 \in \mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}w_{word}RWW})$

Proof. Let $\mathcal{M}_0 = (M_0, \varphi_0)$ be the $\mathsf{w_{word}RWW}$-automaton that is defined by $M_0 = (Q, \Sigma, \Gamma, ¢, \$, q_0, 3, \delta)$ and $\varphi_0 : \delta \to \mathbb{P}_{\text{fin}}(\Sigma^*)$, where $Q = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{\, [c, d] \mid c, d \in \Sigma \,\}$, and δ and φ_0 are given through the following table, where $c, d, e \in \Sigma$:

$$\begin{array}{lllllll}
t_0: & (q_0, ¢\$) & \to & \mathsf{Accept}, & \varphi_0(t_0) & = & \{\lambda\}, \\
t_1: & (q_0, ¢c\$) & \to & \mathsf{Accept}, & \varphi_0(t_1) & = & \{cc\}, \\
t_2: & (q_0, ¢cd) & \to & (q_1, \mathsf{MVR}), & \varphi_0(t_2) & = & \{\lambda\}, \\
t_3: & (q_1, cde) & \to & (q_1, \mathsf{MVR}), & \varphi_0(t_3) & = & \{c\}, \\
t_4: & (q_1, cd\$) & \to & [cd]\$, & \varphi_0(t_4) & = & \{cd\}, \\
t_5: & (q_0, ¢[cd]\$) & \to & \mathsf{Accept}, & \varphi_0(t_5) & = & \{cd\}, \\
t_6: & (q_0, ¢c[de]) & \to & (q_1, \mathsf{MVR}), & \varphi_0(t_6) & = & \{\lambda\}, \\
t_7: & (q_1, c[de]\$) & \to & \mathsf{Accept}, & \varphi_0(t_7) & = & \{cde\}.
\end{array}$$

On input $w \in \Sigma^*$, $|w| \geq 2$, M_0 executes exactly one cycle and an accepting tail. During the cycle it scans the input from left to right, and it encodes the last two symbols a_{n-1} and a_n into an auxiliary symbol of the form $[a_{n-1}, a_n]$. The weight of this cycle with respect to φ_0 is just the word w. After the rewrite/restart step, M_0 scans its tape again from left to right and accepts on seeing the suffix $[a_{n-1}, a_n]\$$. This part of the computation has again weight w. Hence, we see that M_0 is deterministic and monotone, and that $f_{\varphi_0}^{M_0}(w) = ww = \tau_0(w)$, that is, $\mathit{Rel}(\mathcal{M}_0)$ is the graph of the function τ_0. □

Let $\tau_{ab} \subseteq \{a, b\}^* \times \{c, d\}^*$ be the relation

$$\tau_{ab} = \{(ab^n, cd^n) \mid n \geq 0\}.$$

For the relation τ_{ab} we have the following results.

Lemma 5.3.2. $\tau_{ab} \notin \mathcal{R}(\mathsf{RRW\text{-}Td})$.

Proof. We assume that $\tau_{ab} \in \mathcal{R}(\mathsf{RRW\text{-}Td})$, that is, there exists an $\mathsf{RRW\text{-}Td}$ T that computes the relation τ_{ab}. Given an input of the form ab^n, T first outputs a c-symbol, and then it outputs the symbol d n-times, which is the number of b-symbols in the input. The main problem in doing this is the fact that T has to remember that it has already produced a c-symbol. However, as T is not able to use auxiliary symbols, in each cycle T can only replace the window content u by a shorter string v that is from Σ^*, which means that the initial tape content $\alpha u \beta$ changes to $\alpha v \beta$. For the tape content $\alpha v \beta$, we must distinguish between two cases. If $\alpha v\beta$ is not of the form ab^n, that is, it does not belong to the input language anymore, then T would also accept this word as input, as $\alpha v\beta \in \Sigma^*$; if $\alpha v\beta$ belongs to the input language, then T cannot remember that it already produced a c-symbol in the previous cycles, so that it would output more than one c-symbol, contradicting our assumption on T. $\square$

Lemma 5.3.3. *$\tau_{ab} \in \mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}w_{word}R}(1))$.*

Proof. In order to prove the above result, we construct a $\mathsf{det\text{-}mon\text{-}w_{word}R}(1)$-automaton $\mathcal{M} = (M, \omega)$ for the relation τ_{ab}. Actually, for each input of the form ab^n, M just needs to execute an accepting tail and simply read the input from left to right. When reading the prefix a, the associated weight is a c-symbol, and when it scans the suffix b^n of the input, the associated weight is the syllable d^n, which completes our proof. $\square$

The results above together yields the following proper inclusions.

Theorem 5.3.1 ([WO16a])**.** *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$,

$$\begin{array}{llllll}
\text{(a)} & \mathcal{R}(\mathsf{mon\text{-}X\text{-}Td}) & \subsetneq & \mathcal{R}_{lb}(\mathsf{mon\text{-}w_{word}X}) & \subsetneq & \mathcal{R}(\mathsf{mon\text{-}w_{word}X}), \\
\text{(b)} & \mathcal{R}(\mathsf{det\text{-}mon\text{-}X\text{-}Td}) & \subsetneq & \mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}w_{word}X}) & \subsetneq & \mathcal{R}(\mathsf{det\text{-}mon\text{-}w_{word}X}).
\end{array}$$

Proof. As already $\mathsf{det\text{-}mon\text{-}w_{word}R}$-automata can compute relations that are not linearly bounded as shown in Example 5.1.1, it is rather clear that the second inclusions in (a) and (b) hold. Further, from Theorem 5.2.1 we know that each relation that can be computed by a $\mathsf{mon\text{-}RRWW\text{-}Td}$ is also a pushdown relation. Since the relation $Rel(\mathcal{M}_0)$ given in Lemma 5.3.1 is not a pushdown relation, it follows that the first inclusions in (a) and (b) hold for the types $\mathsf{X} \in \{\mathsf{RWW}, \mathsf{RRWW}\}$. Finally, by Lemma 5.3.2 and 5.3.3, it is immediate that the first inclusions in (a) and (b) also hold for the types $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}\}$. $\square$

We now study the relation between the class $\mathcal{R}(\mathsf{mon\text{-}wRWW})$ and the class $\mathcal{R}(\mathsf{mon\text{-}wRRWW})$, their linearly bounded versions, and their deterministic counterparts. It is well-known that the class of languages that are accepted by $\mathsf{mon\text{-}RWW}$-automata coincides with the class of languages that are accepted by $\mathsf{mon\text{-}RRWW}$-automata, and this is also true in the deterministic case (see, e.g., [JMPV99]). Does it also hold for the classes of relations that are computed by weighted restarting automata of the corresponding types? For answering this question, we consider the relation $\tau_1 \subseteq \{a, b, c\}^* \times \{d, e\}^*$ that is defined as follows

$$\tau_1 = \{\, (a^l b^l c^m, d^m e^l) \mid l, m \geq 1 \,\}.$$

Lemma 5.3.4 ([WO16a]). $\tau_1 \in \mathcal{R}_{lb}(\text{det-mon-w}_{\text{word}}\text{RRWW})$.

Proof. In order to prove the above result, we construct a det-mon-w$_{\text{word}}$RRWW-automaton $\mathcal{M} = (M, \omega)$ such that $Rel(\mathcal{M}) = \tau_1$. Let $\mathcal{M}$ proceed as follows. Given an input of the form $w = a^l b^l c^m$, M replaces the prefix aa by the auxiliary symbol A, and then it moves to the suffix c^m. All the transitions used during this part of the computation have weight $\{\lambda\}$. Then it moves across the suffix c^m, where each of these move-right steps has weight $\{d\}$, and finally it restarts on reaching the end marker \$. This step has weight $\{\lambda\}$ again. Now it detects the initial symbol A, which tells M that the first cycle has already been completed successfully. Thus, it now moves right to the boundary between the prefix Aa^{l-2} and the infix b^l, deletes a factor ab, and restarts. Here all steps used have weight $\{\lambda\}$ but the rewrite step, which has weight $\{e\}$. This continues until the prefix Abb is reached, which is then replaces by B by a rewrite step of weight $\{ee\}$, and then M checks that the remaining suffix of the tape content is of the form c^m, and in the affirmative, it halts and accepts. Here again all move-right and restart steps have weight $\{\lambda\}$, and so does the accept step. It follows that M is a det-RRWW-automaton that is monotone, and that the associated weight function ω yields $Rel(\mathcal{M}) = \tau_1$. Clearly τ_1 is linearly bounded. □

Lemma 5.3.5 ([WO16a]). $\tau_1 \notin \mathcal{R}(\text{mon-wRWW})$.

Proof. Assume that $\tau_1 \in \mathcal{R}(\text{mon-wRWW})$, that is, there exists a mon-wRWW-automaton $\mathcal{M}'$ such that $Rel(\mathcal{M}') = \tau_1$. As τ_1 is actually a (partial) function, Proposition 5.1.2 implies that there exists a mon-w$_{\text{word}}$RWW-automaton $\mathcal{M} = (M, \omega)$ such that $Rel(\mathcal{M}) = \tau_1$, where $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is a mon-RWW-automaton, and ω is a weight function that maps each transition of M into a singleton. Interpreting the weight $\omega(t)$ of a transition as output, we see that, for an input of the form $a^l b^l c^m$, $\mathcal{M}$ first outputs the symbol d m-times, which is the number of c-symbols in the input, and then it outputs l e-symbols, which is the number of a- and b-symbols in the input.

As the language $L = \{ a^l b^l c^m \mid l, m \geq 1 \}$ is not regular, M needs to execute rewrite steps in all its accepting computations on input $a^l b^l c^m$, if l is sufficiently large. For the position of the first rewrite step applied in an accepting computation we must distinguish between two cases.

(1) Assume that the first rewrite step is applied within the suffix c^m. While processing this suffix, $\mathcal{M}$ can easily produce the output d^m. Then M must compare the syllables a^l and b^l, and while doing so it should produce the output e^l. For this purpose, M needs to perform rewrite steps within the prefix $a^l b^l$, as the language $L = \{ a^l b^l \mid l \geq 1 \}$ is not regular. However, M is monotone, which means that the position of a rewrite step in a cycle

cannot have a larger right distance than the rewrite step in the previous cycle. If l is sufficiently large, M must reduce both the syllables a^l and b^l in order to compare them. In a cycle of a restarting automaton, there is a unique *rewrite configuration* $\text{¢}xquy\$$, in which a rewrite step $(q,u) \to (q',v)$ is to be executed, and by such a rewrite step the configuration $\text{¢}xvq'y\$$ is reached. In the following cycle, the left distance of the rewrite step can be reduced at most by $|u|-|v|$, that is, the rewrite configuration is $\text{¢}x_1px_2vy\$$, where $x = x_1x_2$ and $|x_2| \leq |u|-|v|$. Let $\hat{k} = \max\{\, |u|-|v| \mid t : (q,u) \to (q',v)$ is a rewrite transition of $M\,\}$. If l is much larger than $\hat{k}$, M needs to execute a series of rewrite steps to reduce the left distance, so that it can perform rewrite steps within the syllable a^l and reduce it.

Assume that the left distance is $2l+r$ $(0 \leq r \leq m)$, when M begins to shorten the left distance. This means that during this process the syllable ab^lc^r has to be reduced by rewrite steps to a word aw that fits into the window of M, where $w \in \Gamma^{\leq k-1}$. In order to compare the number of b-symbols to the number of a-symbols, M must save the information on the number of b-symbols by encoding it into the word w. Let w_i be the code word representing the number i. The word w_i is replaced by the word w_{i+j} in a rewrite step which reduces the infix b^{l-i} to b^{l-i-j}, where $0 \leq i \leq l-1$, $j \geq 1$, and $1 \leq i+j \leq l$. Assume that in each rewrite step at most k' b-symbols can be removed for some constante $0 < k' < k$, and M needs n rewrite steps to reduce the syllable b^l to the word w, which means that there are n rewrite configurations during this process. Let i_j be the number of b-symbols that are reduced in the j-th rewrite step. Now we can describe the process of reducing the syllable b^l by the following sequence

$$b^lw_0 \to b^{l-i_1}w_{i_1} \to b^{l-i_1-i_2}w_{i_1+i_2} \to \ldots \to b^0w_l$$

for some $0 \leq i_1, i_2, \ldots, i_n \leq k'$ and $i_1 + i_2 + \ldots + i_n = l$, where $w = w_l$. It is easily seen that each word in the above sequence is of the form $b^{l-s}w_s$ for some integer $0 \leq s \leq l$. As in each rewrite step during this process at least one b-symbol needs to be removed, the word for storing the number of deleted b-symbols is at most of length $k-1$. However, there are at most $|\Gamma|^{k-1}$ many different words of length $k-1$ representing the number of b-symbols. Choose $l > 1$ to be a constant such that $l > |\Gamma|^{k-1}$, and then there are two integers i, j, $1 \leq i < j \leq l$, such that $w_i = w_j$, but $|b^{l-i}| > |b^{l-j}|$. This means that the stored information on the number of deleted b-symbols is incorrect. Assume that $j = i+s$ for some positive integer $1 \leq s \leq l-1$, and then M cannot distinguish between b^i and b^{i+s}. Together with $a^lb^lc^m$, M would also accept the word $a^lb^{l'}c^m$ for some $l \neq l'$, contradicting our assumption on M.

(2) Assume that the first rewrite step is applied within the prefix a^lb^l. From the arguments above, it follows that the first rewrite step must be executed within the syllable a^l or at the border between the syllables a^l and b^l, that

is, M first compares the syllable a^l to the syllable b^l. There are two cases for producing the output $d^m e^l$.

(2.1) Assume that M already produces the output syllable e^l during the process of comparing a^l to b^l. As the output syllable e^l is preceded by the prefix d^m, $\mathcal{M}$ must already output the syllable d^m before it starts to output e-symbols. As shown in the proof of Theorem 4.2.1, the length of any computation of M on an input of length n is at most $\frac{1}{2}(n+2)(n+3)-1$. It follows that M can perform at most $\frac{1}{2}(2l+2)(2l+3)-1$ steps, while it processes the prefix $a^l b^l$. Choose $r \geq 1$ to be a constant such that $|v| \leq r$ for all transitions t of M and $v \in \omega(t)$, and choose m such that $m > (\frac{1}{2}(2l+2)(2l+3)-1) \cdot r$. Then $\mathcal{M}$ is not able to produce enough d-symbols, while it is processing the prefix $a^l b^l$.

(2.2) Assume that $\mathcal{M}$ first records the number of a- and b-symbols during the process of comparing a^l to b^l, then it outputs the syllable d^m while processing the suffix c^m, finally it outputs the syllable e^l according to the recorded number of a- and b-symbols. If l is sufficiently large, in order to compare the syllable a^l to b^l, M has to shorten them, and at least one a-symbol and one b-symbol are removed in each rewrite step. Hence, there are at most $k-2$ symbols for storing the information on the number of a- and b-symbols, that is, there are at most $|\Gamma|^{k-2}$ different words for storing this number. Let w_i be the code word representing the number i, and then $w = w_l$, as the number of a- and b-symbols is l. In analogy to Case (1), choose $l > 1$ to be a constant such that $l > |\Gamma|^{k-2}$. Then there is an integer $1 \leq i \leq |\Gamma|^{k-2}$ such that $w_l = w_i$. Assume that $i = l - s$ for some positive integer $l - |\Gamma|^{k-2} \leq s \leq l-1$. Then M cannot distinguish between $a^l b^l c^m$ and $a^{l-s} b^{l-s} c^m$. This means that it will produce $d^m e^l$ for the input $a^{l-s} b^{l-s} c^m$, which is an incorrect number of e-symbols.

We have seen that, for an input of the form $a^l b^l c^m$, M cannot correctly produce the output $d^m e^l$, contradicting our assumption on M. □

It is rather clear that $\mathcal{R}(\mathsf{mon\text{-}wRWW})$ is contained in $\mathcal{R}(\mathsf{mon\text{-}wRRWW})$, and $\mathcal{R}(\mathsf{det\text{-}mon\text{-}wRWW})$ is contained in $\mathcal{R}(\mathsf{det\text{-}mon\text{-}wRRWW})$. Thus, Lemmas 5.3.4 and 5.3.5 yield the following proper inclusions, as τ_1 is actually a (partial) function.

Theorem 5.3.2 ([WO16a]). *For each prefix* $\mathsf{x} \in \{\mathsf{w}, \mathsf{w_{FIN}}, \mathsf{w_{word}}\}$,

(a) $\mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}xRWW}) \subsetneq \mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}xRRWW})$,
(b) $\mathcal{R}(\mathsf{det\text{-}mon\text{-}xRWW}) \subsetneq \mathcal{R}(\mathsf{det\text{-}mon\text{-}xRRWW})$,
(c) $\mathcal{R}_{lb}(\mathsf{mon\text{-}xRWW}) \subsetneq \mathcal{R}_{lb}(\mathsf{mon\text{-}xRRWW})$,
(d) $\mathcal{R}(\mathsf{mon\text{-}xRWW}) \subsetneq \mathcal{R}(\mathsf{mon\text{-}xRRWW})$.

We remark that Theorem 5.3.2 is the first result that establishes a difference in the computational power between a model of the monotone RWW-automaton and the corresponding model of the monotone RRWW-automaton.

The relation $\tau_1 = \{ (a^l b^l c^m, d^m e^l) \mid l, m \geq 1 \}$ considered above is a pushdown function that is not computed by any mon-$\mathsf{w}_{\mathsf{word}}$RWW-automaton. On the other hand, the function τ_0 is computed by a det-mon-$\mathsf{w}_{\mathsf{word}}$RWW-automaton that is linearly bounded, but it is not a pushdown relation. Thus, we have the following incomparability result.

Theorem 5.3.3 ([WO16a]). *For each prefix* $\mathsf{x} \in \{\mathsf{w}, \mathsf{w}_{\mathsf{FIN}}, \mathsf{w}_{\mathsf{word}}\}$*, the classes of relations* $\mathcal{R}_{lb}$(det-mon-xRWW) *and* $\mathcal{R}_{lb}$(mon-xRWW) *are incomparable to the classes* lbPDR *and* PDR *with respect to inclusion.*

Recall that the class of languages that are accepted by mon-RWW- and mon-RRWW-automata coincides with the language class CFL, and that the class of languages that are accepted by det-mon-RWW- and det-mon-RRWW-automata coincides with the language class DCFL (see, e.g., [JMPV99]) . We have seen that this result does not carry over to the classes of relations that are computed by weighted restarting automata.

We now turn to the class of relations that are computed by mon-wRRWW-automata. Let $\tau_2 \subseteq \{a, b, c\}^* \times \{d, e\}^*$ be the relation

$$\tau_2 = \{ (a^l b^l c^{m+r} a^r, d^m e^l d^m e^r) \mid l, m, r \geq 1 \}.$$

Lemma 5.3.6 ([WO16a]). $\tau_2 \notin \mathcal{R}$(mon-wRRWW).

Proof. Assume that $\tau_2 \in \mathcal{R}$(mon-wRRWW), that is, there is a mon-wRRWW-automaton $\mathcal{M}'$ such that $Rel(\mathcal{M}') = \tau_2$. As the relation τ_2 is a partial function, by Proposition 5.1.2 it is also computed by a mon-wRRWW-automaton $\mathcal{M} = (M, \omega)$, where $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is a mon-RRWW-automaton, and ω is a weight function that maps each transition of M into a singleton over $\Delta = \{d, e\}$. Interpreting the weight $\omega(t) \in \{d, e\}^*$ of a transition t as output, $\mathcal{M}$ first outputs the syllable d^m, then e^l, then d^m again, and finally e^r given the word $a^l b^l c^{m+r} a^r$ as input.

As the exact value of m is unknown, $\mathcal{M}$ must first guess it, and produce correspondingly many d-symbols. Further, after comparing the numbers of a- and b-symbols and producing correspondingly many e-symbols, $\mathcal{M}$ must again produce m d-symbols, that is, it must somehow remember this number. We consider two cases for storing the guessed value of m.

(1) While moving right across the input syllable c^{m+r}, M can guess the value of m, and produce correspondingly many d-symbols. In order to remember the number m, it can place a marking on the syllable c^{m+r} before performing the restart step. This means that M has already executed a rewrite step

within the syllable c^{m+r} before the prefix a^l has been compared to the infix b^l. However, M is monotone, and it must apply rewrite steps for comparing the numbers of a- and b-symbols. In the proof of Lemma 5.3.5 it is shown that M cannot do this successfully, if it has already performed a rewrite step within the syllable c^{m+r}.

(2) From the arguments above, M can only perform rewrites on the prefix a^l or at the border between the syllables a^l and b^l of the input before comparing the number of a-symbols to the number of b-symbols. It follows that M can only remember the guessed value of m by encoding it into the prefix $a^l b^l$ of the input. Since the information about the exponent l is lost during the process of comparing the syllable a^l to the syllable b^l (see the proof of Lemma 5.3.5), $\mathcal{M}$ must produce the output syllable e^l during this process. Further, as the output syllable e^l is preceded by the prefix d^m, the prefix d^m of the output must be produced before this process starts, which means that $\mathcal{M}$ can only perform rewrites on the prefix $a^l b^l$ of the input, while it produces the output d^m. However, as shown in the proof of Lemma 5.3.5, there are only finitely many numbers that can be represented by a code word of a fixed length. Accordingly, let $h_M(n)$ denote the maximum number that can be represented by a code word of the length n for the restarting automaton M. If we choose $m > 1$ to be a constant such that $m > h_M(2l)$, the information about the guessed value of m cannot be correctly saved within the space covered by the syllable $a^l b^l$, which is analog to Case (2) in the proof of Lemma 5.3.5. Hence, if the number m is sufficiently large, then $\mathcal{M}$ cannot reproduce the syllable d^m again, which contradicts our assumption. □

Clearly τ_2 is a linearly bounded pushdown relation, too. Hence, from Lemma 5.3.1 and Lemma 5.3.6 the following incomparability result follows.

Theorem 5.3.4 ([WO16a]). *For each prefix* $\mathsf{x} \in \{\mathsf{w}, \mathsf{w_{FIN}}, \mathsf{w_{word}}\}$*, the classes of relations* $\mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}xRRWW})$ *and* $\mathcal{R}_{lb}(\mathsf{mon\text{-}xRRWW})$ *are incomparable to the classes* lbPDR *and* PDR *with respect to inclusion.*

5.3.1 Monotone Weighted Restarting Automata with Window Size One

In this section we study the classes of relations that are computed by monotone weighted restarting automata with window size one and compare them to each other. A restarting automaton with window size one can only delete the symbol in the window in a rewrite step. In this case RWW-and RRWW-automata are equally expressive as R- and RR-automata, respectively, and hence here we only consider mon-wR- and mon-wRR-automata.

First, we begin this investigation by studying the inclusion relation between the classes $\mathcal{R}(\mathsf{mon\text{-}wR}(1))$ and $\mathcal{R}(\mathsf{mon\text{-}wRR}(1))$. Since an R-automaton is a restricted version of an RR-automaton, by Proposition 5.1.3 we just need to find a language $L \in \mathcal{L}(\mathsf{mon\text{-}RR}(1)) \smallsetminus \mathcal{L}(\mathsf{mon\text{-}R}(1))$ in order to prove the properness of this inclusion. It is well-known that each language that is accepted by a $\mathsf{mon\text{-}R}(1)$-automaton is regular (see, e.g., [Mrá01]). Further, a $\mathsf{mon\text{-}RR}(1)$-automaton can accept a language that is not regular, and the following language example is given in [KO12]:

$$L_4 = \{a^m b^n \mid m = n \text{ or } m + 1 = n\}.$$

As $L_4 \in \mathcal{L}(\mathsf{mon\text{-}RR}(1)) \smallsetminus \mathcal{L}(\mathsf{mon\text{-}R}(1))$ [HO12], it follows that $\mathcal{R}(\mathsf{mon\text{-}wR}(1)) \subsetneq \mathcal{R}(\mathsf{mon\text{-}wRR}(1))$, and obviously this result carries over to the class of relations that are computed by linearly bounded weighted restarting automata and restarting transducers. Hence, the following results can be established.

Theorem 5.3.5. *For each prefix* $\mathsf{x} \in \{\mathsf{w}, \mathsf{w_{FIN}}, \mathsf{w_{word}}\}$*,*

$$\begin{array}{lll} (a) & \mathcal{R}(\mathsf{mon\text{-}xR}(1)) & \subsetneq \mathcal{R}(\mathsf{mon\text{-}xRR}(1)), \\ (b) & \mathcal{R}_{lb}(\mathsf{mon\text{-}xR}(1)) & \subsetneq \mathcal{R}_{lb}(\mathsf{mon\text{-}xRR}(1)), \\ (c) & \mathcal{R}(\mathsf{mon\text{-}R(1)\text{-}Td}) & \subsetneq \mathcal{R}(\mathsf{mon\text{-}RR(1)\text{-}Td}). \end{array}$$

Actually, some results on the expressive power of restarting transducers with window size one are presented in [HO15]. Now we continue by studying the inclusion relations between the classes $\mathcal{R}(\mathsf{mon\text{-}w_{word}R(R)}(1))$, $\mathcal{R}_{lb}(\mathsf{mon\text{-}w_{word}R(R)}(1))$ and $\mathcal{R}(\mathsf{mon\text{-}R(R)(1)\text{-}Td})$. In fact, a $\mathsf{det\text{-}mon\text{-}w_{word}R}(1)$-automaton can already compute a relation that does not belong to the class PDR. Let $\tau_3 \subseteq \{a, \#\}^* \times \{c\}^*$ be the relation

$$\tau_3 = \{((\#a)^n, c^{\frac{(n+1)(n+4)}{2}}) \mid n \geq 0\}.$$

Lemma 5.3.7. $\mathcal{R}(\mathsf{det\text{-}mon\text{-}w_{word}R}(1)) \smallsetminus \mathsf{PDR} \neq \emptyset$.

Proof. In order to prove the result above, we need to find a relation that is computed by a $\mathsf{det\text{-}mon\text{-}w_{word}R}(1)$-automaton, while it is not a pushdown relation. Let $\mathcal{M} = (M, \omega)$ be a $\mathsf{det\text{-}mon\text{-}w_{word}R}$-automaton, where $M = (\{q_0, q_r\}, \{a, \#\}, \{a, \#\}, ¢, \$, 1, q_0, \delta)$, and the transition function δ is defined as follows:

$$\begin{array}{llllll} t_1: & (q_0, ¢) & \to (q_0, \mathsf{MVR}), & t_4: & (q_r, -) & \to \mathsf{Restart}, \\ t_2: & (q_0, \#) & \to (q_0, \mathsf{MVR}), & t_5: & (q_0, \$) & \to \mathsf{Accept}. \\ t_3: & (q_0, a) & \to (q_r, \lambda), & & & \end{array}$$

Let ω be a weight function that assigns the set $\{c\}$ to the transitions t_1,t_2 and t_5, and that assigns the set $\{\lambda\}$ to the transitions t_3 and t_4.

It is easily seen that M accepts all input words of the form $w \in (\#^* a \#^*)^*$. In each cycle $\mathcal{M}$ processes the prefix of the tape $¢\#^n a$, producing the syllable c^{n+1} as output, and then it deletes the a-symbol and restarts. If $\mathcal{M}$ cannot see any a-symbol until it reaches the right sentinel \$, then it accepts, again producing a c-symbol. Obviously, for each input $(\#a)^n$, we obtain that $f_\omega^M((\#a)^n) = \{c^{\frac{(n+1)(n+4)}{2}}\}$. It is easily seen that the input language $\{ (\#a)^n) \mid n \geq 0 \}$ is regular, while the output language $\{ c^{\frac{(n+1)(n+4)}{2}} \mid n \geq 0 \}$ ist not context-free. As the image of a regular language under a pushdown transducer is context-free (see, e.g., [GR66, GR68]), it follows that $Rel(\mathcal{M}) \notin \mathsf{PDR}$, which completes this proof. $\square$

By Lemma 5.3.3, we see that the relation τ_{ab} can be computed by a det-mon-w$_{\mathsf{word}}$R(1)-automaton, while it cannot be computed by any RRW-Td. This together with Lemma 5.3.7 yields the following proper inclusions.

Theorem 5.3.6.

$$\begin{array}{llllll}
(a) & \mathcal{R}(\mathsf{mon\text{-}R(1)\text{-}Td}) & \subsetneq & \mathcal{R}_{lb}(\mathsf{mon\text{-}w_{word}R(1)}) & \subsetneq & \mathcal{R}(\mathsf{mon\text{-}w_{word}R(1)}),\\
(b) & \mathcal{R}(\mathsf{det\text{-}mon\text{-}R(1)\text{-}Td}) & \subsetneq & \mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}w_{word}R(1)}) & \subsetneq & \mathcal{R}(\mathsf{det\text{-}mon\text{-}w_{word}R(1)}),\\
(c) & \mathcal{R}(\mathsf{mon\text{-}RR(1)\text{-}Td}) & \subsetneq & \mathcal{R}_{lb}(\mathsf{mon\text{-}w_{word}RR(1)}) & \subsetneq & \mathcal{R}(\mathsf{mon\text{-}w_{word}RR(1)}),\\
(d) & \mathcal{R}(\mathsf{det\text{-}mon\text{-}RR(1)\text{-}Td}) & \subsetneq & \mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}w_{word}RR(1)}) & \subsetneq & \mathcal{R}(\mathsf{det\text{-}mon\text{-}w_{word}RR(1)}).
\end{array}$$

In addition, from Lemma 5.3.7 the following incomparability result can also be obtained.

Theorem 5.3.7. *For each prefix* $\mathsf{x} \in \{\mathsf{w}, \mathsf{w_{FIN}}, \mathsf{w_{word}}\}$*, the classes of relations* $\mathcal{R}((\mathsf{det\text{-}})\mathsf{mon\text{-}xR(1)})$ *and* $\mathcal{R}((\mathsf{det\text{-}})\mathsf{mon\text{-}xRR(1)})$ *are incomparable to the classes* lbPDR *and* PDR *with respect to inclusion.*

5.4 Concluding Remarks

We have studied the classes of (binary) relations that are computed by weighted restarting automata that are monotone, both in the deterministic and the nondeterminitic case. We have compared these classes to the classes of relations that are computed by corresponding versions of restarting transducers and to some classes of pushdown relations. The inclusion results obtained are summarized in the diagram in Figure 5.1. In particular, we have shown that the mon-RWW-Tds and mon-RRWW-Tds characterize the almost-realtime pushdown relations artPDR, and that the det-mon-RWW-Tds characterize the deterministic pushdown functions DPDF. Also we have seen that monotone (word-) weighted RWW-automata are strictly weaker in computational power than monotone (word-)weighted RRWW-automata, both in the deterministic as well as in the nondeterministic case. These are the first results where it

$$\begin{tikzcd}
\mathcal{R}(\mathsf{det\text{-}mon\text{-}wX}) \arrow[r] & \mathcal{R}(\mathsf{det\text{-}mon\text{-}wRX}) \arrow[r] & \mathcal{R}(\mathsf{mon\text{-}wRX}) & \mathcal{R}(\mathsf{mon\text{-}wX}) \arrow[l] \\
\mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}wX}) \arrow[r] \arrow[u] & \mathcal{R}_{lb}(\mathsf{det\text{-}mon\text{-}wRX}) \arrow[r] \arrow[u] & \mathcal{R}_{lb}(\mathsf{mon\text{-}wRX}) \arrow[u] & \mathcal{R}_{lb}(\mathsf{mon\text{-}wX}) \arrow[l] \arrow[u] \\
 & \mathcal{R}(\mathsf{det\text{-}mon\text{-}RX\text{-}Td}) \arrow[r] \arrow[u] & \mathcal{R}(\mathsf{mon\text{-}RX\text{-}Td}) \arrow[ur] & \mathsf{PDR} \\
 & \mathcal{R}(\mathsf{det\text{-}mon\text{-}X\text{-}Td}) \arrow[r] \arrow[u] \arrow[uul] & \mathcal{R}(\mathsf{mon\text{-}X\text{-}Td}) \arrow[u, equal] & \\
\mathsf{DPDF} = \mathsf{lbDPDF} \arrow[r, equal] & \mathsf{artDPDF} \arrow[r] \arrow[u, equal] & \mathsf{artPDR} \arrow[r] \arrow[u, equal] & \mathsf{lbPDR} \arrow[uu] \\
 & \mathsf{rtDPDF} \arrow[r] \arrow[u] & \mathsf{rtPDR} \arrow[u] &
\end{tikzcd}$$

Figure 5.1: Hierarchy of classes of (binary) relations that are computed by the various types of monotone restarting transducers and (word-)weighted restarting automata. X denotes the type of restarting automaton RWW, an arrow denotes a proper inclusion, and classes that are not connected through a sequence of arrows (or by trivial inclusion) are incomparable with respect to inclusion.

has been shown that a version of the (nondeterministic) monotone RWW-automaton differs in expressive power from the corresponding version of the (nondeterministic) monotone RRWW-automaton. Of course, it remains to derive a characterization of the classes of relations computed by these automata in terms of other types of devices.

Finally, we have studied the classes of relations that are computed by weighted monotone restarting automata with a window of size one, and the inclusion results obtained are summarized in the diagram in Figure 5.2. We have seen that the class $\mathcal{R}(\mathsf{mon\text{-}wR}(1))$ is strictly contained in the class $\mathcal{R}(\mathsf{mon\text{-}wRR}(1))$, and this is also true for the corresponding version of linearly bounded weighted restarting automata and restarting transducers. Further, we have shown that the classes $\mathcal{R}((\mathsf{det\text{-}})\mathsf{mon\text{-}wR}(1))$ and $\mathcal{R}((\mathsf{det\text{-}})\mathsf{mon\text{-}wRR}(1))$ are incomparable to the classes lbPDR and PDR with respect to inclusion. However, the inclusion relation between the classes $\mathcal{R}_{lb}((\mathsf{det\text{-}})\mathsf{mon\text{-}wRR}(1))$ and lbPDR is still open.

$$\begin{tikzcd}
\mathcal{R}(\mathsf{mon\text{-}wR}(1)) \arrow[r] & \mathcal{R}(\mathsf{mon\text{-}wRR}(1)) \arrow[r, dash, dashed] \arrow[dr, dash, dashed] & \mathsf{PDR} \\
\mathcal{R}_{lb}(\mathsf{mon\text{-}wR}(1)) \arrow[r] \arrow[u] & \mathcal{R}_{lb}(\mathsf{mon\text{-}wRR}(1)) \arrow[u] & \mathsf{lbPDR} \arrow[u] \\
\mathcal{R}(\mathsf{mon\text{-}R}(1)\text{-}\mathsf{Td}) \arrow[r] \arrow[u] & \mathcal{R}(\mathsf{mon\text{-}RR}(1)\text{-}\mathsf{Td}) \arrow[r] \arrow[u] & \mathsf{artPDR} \arrow[u]
\end{tikzcd}$$

Figure 5.2: Hierarchy of classes of (binary) relations that are computed by monotone (word-)weighted R- and RR-automata with window size one and the corresponding versions of restarting transducers. An arrow denotes a proper inclusion, a dashed line denotes an incomparability relation with respect to inclusion, and the inclusion relations between the classes that are not connected through a sequence of arrows (or by trivial inclusion) are unknown.

Chapter 6

Languages Accepted by Weighted Restarting Automata

In the previous chapters, we studied the classes of functions and relations that are computed by weighted restarting automata. The purpose of this chapter is to introduce classes of languages that are accepted by weighted restarting automata. We use weighted restarting automata to define classes of formal languages by combining the acceptance condition of a restarting automaton with a condition on the weight of its accepting computations.

Actually, there are already a number of studies on non-standard ways to define formal languages instead of directly using automata and grammars. In [Kam09] M. Kambites gives a finite automaton with a register that can store an element of a given monoid or group, and the content of the register is used as an acceptance condition. An input is accepted by such an automaton if and only if the automaton accepts and the additional acceptance condition is satisfied. Cavaliere and Leupold have introduced so-called *observer systems* (see, e.g., [CL04, CL06]). An observer system translates the behaviour of an automaton or a grammar into a readable output, and there is a relative acceptance condition on this output. In addition, observer systems have also been applied to restarting transducers (see, e.g., [LH15]).

Following the same fundamental idea, we extend weighted restarting automata to language acceptors [WO16b]. Recall that in a weighted restarting automaton $\mathcal{M} = (M, \omega)$, the weight function ω assigns an element of a given semiring S as a weight to each transition of the restarting automaton M. The product (in S) of the weights of all transitions that are used in a computation then yields a weight for that computation, and the sum over all weights of all accepting computations of M for a given input word $w \in \Sigma^*$ yields a value from S. Thus, a partial function $f_\omega^M : \Sigma^* \to S$ is obtained. By placing an acceptance condition T on the value $f_\omega^M(w)$, we can define a subset $L_T(\mathcal{M})$ of the language $L(M)$ that is accepted by M. In this way, a weighted restarting automaton can be used as a language acceptor, which accepts a sublanguage of the language that is accepted by the underlying (unweighted) restarting automaton.

Obviously, the acceptance conditions relative to different semirings are not equally powerful. For example, the *Boolean semiring* $(\{0,1\},+,\cdot,0,1)$ can easily be simulated by other semirings. Here we consider some semirings over integers such as the *tropical semiring* $\mathbb{Z}_\infty = (\mathbb{Z}_\infty, \min, +, \infty, 0)$, and the semiring of formal languages such as the context-free languages $\mathsf{CFL}(\Delta)$ and the regular languages $\mathsf{REG}(\Delta)$ over a given finite alphabet Δ. In the latter case we restrict our attention to the word-weighted restarting automata introduced in Chapter 5. In this chapter we study the classes of languages that are accepted by weighted restarting automata relative to the acceptance conditions from various semirings and compare them to each other.

6.1 Definitions and Examples

The goal of this section is to present some basic notions and examples. We begin with the following two definitions, which are the central notions of this chapter.

Definition 6.1.1 ([WO16b]). *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a restarting automaton, let ω be a weight function from M into a semiring S, and let $\mathcal{M} = (M, \omega)$. For a subset T of S, $L_T(\mathcal{M}) = \{\, w \in L(M) \mid f_\omega^M(w) \in T \,\}$ is the language* accepted by M relative to T*, that is, a word $w \in \Sigma^*$ belongs to the language $L_T(\mathcal{M})$ iff $w \in L(M)$ and $f_\omega^M(w) \in T$* [1].

Definition 6.1.2 ([WO16b]). *Let X be a type of restarting automaton, let S be a semiring, and let $\mathbb{H}$ be a family of subsets of S. Then*

$$\mathcal{L}(\mathsf{X}, S, \mathbb{H}) = \{\, L_T(\mathcal{M}) \mid \begin{array}{l}\mathcal{M} \text{ is a weighted restarting automaton of type } \mathsf{X}\\ \text{and } T \in \mathbb{H} \,\}\end{array}$$

is the class of languages that are accepted by weighted restarting automata of type X relative to $\mathbb{H}$.

Now we continue with some examples that illustrate our definitions.

Example 6.1.1. *Let $M_1 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be the* mon-R*-automaton that is defined by taking $Q = \{q_0, q_r\}$, $\Gamma = \Sigma = \{a, b\}$, and $k = 4$, where δ is defined*

[1] Note that if the subset T is not recursive, then it is undecidable whether $f_\omega^M(w) \in T$ for an input $w \in \Sigma^*$.

as follows:

$$\begin{array}{rlcl@{\qquad}rlcl}
t_1: & (q_0, ¢aaa) & \to & (q_0, \mathsf{MVR}), & t_7: & (q_0, abb\$) & \to & (q_r, b\$),\\
t_2: & (q_0, aaaa) & \to & (q_0, \mathsf{MVR}), & t_8: & (q_0, abb\$) & \to & (q_r, \$),\\
t_3: & (q_0, aaab) & \to & (q_0, \mathsf{MVR}), & t_9: & (q_0, ¢ab\$) & \to & \mathsf{Accept},\\
t_4: & (q_0, aabb) & \to & (q_0, \mathsf{MVR}), & t_{10}: & (q_0, ¢\$) & \to & \mathsf{Accept},\\
t_5: & (q_0, abbb) & \to & (q_r, bb), & t_{11}: & (q_0, ¢aab) & \to & (q_0, \mathsf{MVR}),\\
t_6: & (q_0, abbb) & \to & (q_r, b), & t_{12}: & (q_0, ¢abb) & \to & (q_0, \mathsf{MVR}),\\
t_{13,x}: & (q_r, x) & \to & \mathsf{Restart} \text{ for all admissible } x. & & & &
\end{array}$$

It is easily seen that $L(M_1) = \{ a^m b^n \mid 0 \le m \le n \le 2m \}$. *Further, for* $a^n b^n$ *and for* $a^n b^{2n}$, M_1 *has just a single accepting computation.*

Let $(\mathsf{REG}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$ *be the semiring of regular languages over* $\Delta = \{c, d\}$, *and let* ω_1 *be the weight function that is defined as follows:*

$$\omega_1(t_i) = \begin{cases} \{c\}, & \text{if } i = \{5, 7, 9\},\\ \{d\}, & \text{if } i \in \{6, 8\},\\ \{\lambda\}, & \text{otherwise.} \end{cases}$$

Finally, let $\mathcal{M}_1 = (M_1, \omega_1)$, *and let*

$$T_1 = \{ \{c^m\} \mid m \ge 0 \} \cup \{ \{d^n\} \mid n \ge 0 \}.$$

Then $f_{\omega_1}^{M_1}(w) \in T_1$ *iff* $w \in L(M_1)$, *and* $|w|_a = |w|_b$ *or* $2 \cdot |w|_a = |w|_b$, *which yields*

$$L_{T_1}(\mathcal{M}_1) = \{ a^n b^n \mid n \ge 0 \} \cup \{ a^n b^{2n} \mid n \ge 0 \}.$$

It is known that the language $L_{T_1}(M_1)$ *is not even accepted by any* RW-*automaton [JMPV99]. Hence, we see that the notion of relative acceptance increases the expressive power of* R-*automata.*

Example 6.1.2. *Let* $M_2 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ *be the* det-mon-R-*automaton that is defined by taking* $Q = \{q_0, q_r\}$, $\Gamma = \Sigma = \{a, b, c\}$, *and* $k = 3$, *where* δ *is defined as follows:*

$$\begin{array}{rlcl@{\qquad}rlcl}
t_1: & (q_0, ¢aa) & \to & (q_0, \mathsf{MVR}), & t_7: & (q_0, ¢ab) & \to & (q_0, \mathsf{MVR}),\\
t_2: & (q_0, aaa) & \to & (q_0, \mathsf{MVR}), & t_8: & (q_0, ¢cc) & \to & (q_0, \mathsf{MVR}),\\
t_3: & (q_0, aab) & \to & (q_0, \mathsf{MVR}), & t_9: & (q_0, ccc) & \to & (q_0, \mathsf{MVR}),\\
t_4: & (q_0, abb) & \to & (q_r, b), & t_{10}: & (q_0, cc\$) & \to & \mathsf{Accept},\\
t_5: & (q_0, abc) & \to & (q_r, c), & t_{11}: & (q_0, ¢c\$) & \to & \mathsf{Accept},\\
t_6: & (q_0, ab\$) & \to & (q_r, \$), & t_{12}: & (q_0, ¢\$) & \to & \mathsf{Accept},\\
t_{13,x}: & (q_r, x) & \to & \mathsf{Restart} \text{ for all admissible } x. & & & &
\end{array}$$

It is easily seen that $L(M_2) = \{\, a^m b^m c^n \mid m, n \geq 0 \,\}$. *Let* $\mathbb{Z}_\infty = (\mathbb{Z}_\infty, \min, +, \infty, 0)$ *be the* tropical semiring, *and let* ω_2 *be the weight function from the transitions of* M_2 *into the semiring* $\mathbb{Z}_\infty$ *that is defined as follows*

$$\omega_2(t_i) = \begin{cases} 1, & \text{if } i \in \{4,5,6\}, \\ -1, & \text{if } i \in \{8,9,10,11\}, \\ 0, & \text{otherwise.} \end{cases}$$

Let $\mathcal{M}_2 = (M_2, \omega_2)$, *and let* $T_2 = \{0\}$. *Then* $f_{\omega_2}^{M_2}(w) \in T_2$ *if and only if* $w \in L(M_2)$, *and* w *must be of the form* $a^n b^n c^n$ *for* $n \geq 0$, *that is,*

$$L_{T_2}(\mathcal{M}_2) = \{\, a^n b^n c^n \mid n \geq 0 \,\}.$$

It is clear that $L_{T_2}(\mathcal{M}_2) \notin \mathsf{CFL}$. *We see that by using the notion of relative acceptance a* det-mon-R-*automaton can accept a language that is not context-free.*

Example 6.1.3. *Let* $\mathcal{M}_3 = (M_3, \omega_3)$ *be the* $\mathsf{w}_{\mathsf{word}}\mathsf{RWW}$-*automaton that works exactly as the* wRWW-*automaton* (M_2, ω_2) *that is given in Example 3.3.2, and then*

$$f_{\omega_3}^{M_3}(w) = \{\, w_1\#w_2\#\ldots\#w_n\# \mid w = w_1 w_1^R \ldots w_n w_n^R \,\}.$$

Further, let

$$T_3 = \{\, X \mid X \in \mathbb{P}_{\mathrm{fin}}(\Gamma^*) \text{ and } |x|_\# = 2 \text{ for all } x \in X \,\}.$$

Then $f_{\omega_3}^{M_3}(w) \in T_3$ *if and only if* $w \in L(M_3)$, *and all admissible factorizations of* w *as* $w = w_1 w_1^R w_2 w_2^R \ldots w_n w_n^R$ *satisfy the restriction that* $n = 2$, *that is,*

$$L_{T_3}(\mathcal{M}_3) = \{\, w \in \Sigma^+ \mid \ w \in L(M_3), \text{all admissible factorizations of } w \text{ have length } 2 \,\}.$$

While $L(M_3)$ *is context-free, it can be shown that* $L_{T_3}(\mathcal{M}_3)$ *is not context-free. For example, for* $m, n \in \mathbb{N}$, *we have that* $w(m,n) = a^2 b^{2m} b^{2m} a^2 a^2 b^{2n} b^{2n} a^2 \in L(M_3)$, *and if* $m \neq n$, *then* $w(m,n)$ *has only a single admissible factorization, which is of length two, but for* $m = n$, *it also has an admissible factorization of length one. This observation together with the Ogden's Lemma [Ogd68] can now be used to prove that* $L_{T_3}(\mathcal{M}_3)$ *is not context-free.*

As shown in the examples above, for a weighted restarting automata $\mathcal{M} = (M, \omega)$, by placing a condition T on the value $f_\omega^M(w)$, some words from the language $L(M)$ are filtered out. This result shows that the sublanguages of the form $L_T(\mathcal{M})$ can be more complex than the language of the form $L(M)$ itself.

6.2 On the Classes of Languages Accepted Relative to Some Semirings over Integers

First, we consider the languages that are accepted by weighted restarting automata by using acceptance conditions relative to subsets of the tropical semiring $\mathbb{Z}_\infty = (\mathbb{Z}_\infty, \min, +, \infty, 0)$. Our first result states that by using the notion of acceptance relative to the family $\mathbb{H}_0 = \{\{0\}\}$, we can avoid auxiliary symbols. For proving this result, we need the following technical lemma.

Lemma 6.2.1 ([WO16b]). *For each* (R)RWW-*automaton M, there exists an* (R)RWW-*automaton M' accepting only on empty tape such that $L(M) = L(M')$.*

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an (R)RWW-automaton. We construct an (R)RWW-automaton M' that simulates M as follows. In each cycle M' first guesses whether to simulate a cycle of M or whether M has already accepted.

(1) In the former case, another cycle is simulated, in which M' performs the same move-right, rewrite, and restart steps as M. However, each accept transition of M is simulated by a rewrite transition of M' that replaces the content of the window by a special symbol #, which indicates that the corresponding computation of M has accepted. If during the simulation of a cycle of M, the symbol # is encountered by M', then M' halts without accepting.

(2) In the latter case, on seeing the symbol #, M' simply erases all symbols that are to the left of the symbol # in the window and restarts. If the symbol # is not encountered, then M' halts without accepting. This process is repeated until a tape content of the form $¢\#\alpha\$$ is reached, which M' then deletes symbol by symbol from right to left. After deleting the last symbol, M' halts and accepts. It should be clear that $L(M) = L(M')$. □

Theorem 6.2.1 ([WO16b]).
For all $\mathsf{X} \in \{\mathsf{RRW}, \mathsf{RW}\}$, $\mathcal{L}(\mathsf{XW}) \subseteq \mathcal{L}(\mathsf{X}, \mathbb{Z}_\infty, \mathbb{H}_0)$.

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an RRWW-automaton with input alphabet Σ. In order to prove the above inclusion, we construct a wRRW-automaton $\mathcal{M} = (M', \omega)$, where M' is an RRW-automaton, and ω is a weight function from the transitions of M' to $\mathbb{Z}_\infty$ such that $L_{\{0\}}(\mathcal{M}) = L(M)$. By Lemma 6.2.1, we can assume without loss of generality that M always accepts on empty tape. Let $M' = (Q, \Gamma, \Gamma, ¢, \$, q_0', k', \delta')$ be the RRW-automaton that is obtained from M by simply taking all symbols as input symbols. For a word $x \in \Gamma^*$, let $|x|_{\Gamma\smallsetminus\Sigma}$ denote the number of occurrences of symbols from the set $\Gamma \smallsetminus \Sigma$ in the word x, that is, the number of occurrences of auxiliary letters in x. Now we define the weight function ω as follows

$$\omega(t) = \begin{cases} |v|_{\Gamma\smallsetminus\Sigma} - |u|_{\Gamma\smallsetminus\Sigma}, & \text{for each rewrite transition } t \\ & \text{of the form } (q',v) \in \delta(q,u), \\ 0, & \text{otherwise.} \end{cases}$$

Then, for each accepting computation AC of M' on input $w \in \Gamma^+$, $\omega(AC)$ is the number of auxiliary symbols that are written onto the tape during this computation minus the number of auxiliary symbols that are removed from the tape during this computation. Since M' (just as M) always accepts on empty tape, we see that $\omega(AC) = -|w|_{\Gamma\smallsetminus\Sigma}$. Hence, $f_\omega^{M'}(w) = 0$ iff w does not contain any auxiliary symbols, that is, $L_{\{0\}}(\mathcal{M}) = L(M)$. In exactly the same way an RWW-automaton can also be simulated by a wRW-automaton relative to the subset $\{0\}$. □

It remains open whether the inclusion above is a proper one. Now we continue by considering the languages that are accepted by weighted restarting automata relative to subsets of the semiring $\overline{\mathbb{N}} \times \mathbb{N}$, that is, the direct product of the semirings $\overline{\mathbb{N}} = (\overline{\mathbb{N}}, \max, +, -\infty, 0)$ and $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$. For each $(m,n), (m',n') \in \overline{\mathbb{N}} \times \mathbb{N}$, the operations $\oplus$ and $\odot$ are described as follows

$$\begin{aligned} (m,n) \oplus (m',n') &= (\max(m,m'), n+n'), \\ (m,n) \odot (m',n') &= (m+m', n\cdot n'). \end{aligned}$$

Further, let $\mathbb{H}_{exp}$ be the family of subsets of the semiring $\overline{\mathbb{N}} \times \mathbb{N}$ that is defined as follows

$$\mathbb{H}_{exp} = \{\{(n,m) \in \overline{\mathbb{N}} \times \mathbb{N} \mid n \geq 1, m \geq 2^{n-1}\}\}.$$

It is well-known that the *3-satisfiability problem* (3-SAT for short) is NP-complete, and in [JLNO04] a specific encoding $\varphi(\alpha)$ of 3-SAT is given, where α is a Boolean formula in conjunctive normal form of degree 3. Let $N(\alpha)$ denote the number of satisfying assignments for the Boolean formula α, and let $L_{3SAT,half}$ be the following language

$$L_{3SAT,half} = \{\varphi(\alpha) \mid \begin{array}{l} \alpha \in \text{3-SAT}, N(\alpha) \geq 2^{n-1}, \\ \text{where } \alpha \text{ contains the variables } x_1, x_2, \ldots, x_n\}. \end{array}$$

Concerning the language $L_{3SAT,half}$ we have the following result.

Proposition 6.2.1 ([Wan17]). $L_{3SAT,half} \in \mathcal{L}(\mathsf{RWW}, \overline{\mathbb{N}} \times \mathbb{N}, \mathbb{H}_{exp})$.

Proof. Recall that there exists an RWW-automaton M that accepts the language L_{3SAT} that contains the encoded words of the form $\varphi(\alpha)$ of 3-SAT [JLNO04]. Given an input of the form $\varphi(\alpha)$ of a Boolean formula α in conjunctive normal form of degree 3, M guesses a truth assignments ψ for each variable occurring in α, and it simply verifies whether $\psi(\alpha)$ evaluates to the truth value 1. Let ω be a weight function that is defined as follows. When guessing the value of a variable, ω assigns the weight $(1,1)$ to the corresponding transition of M. If n is the number of variables in the given formula α, then the computation part for guessing a truth assignment has the associated weight $(n,1)$. Further, ω assigns $(0,1)$ to all other transitions of M, and thus the weight of an accepting computation is the weight $(n,1)$. It follows that $f_\omega^M(\varphi(\alpha)) = (n,m)$, where m is the number of accepting computations on the input $\varphi(\alpha)$, i.e., the number of satisfying truth assignments for α. Finally, if we take $T = \{(n,m) \in \overline{\mathbb{N}} \times \mathbb{N} \mid n \geq 1, m \geq 2^{n-1}\}$, then it holds that $L_T((M,\omega)) = L_{3SAT,half}$. □

Actually, the problem of determining the number of satisfying truth assignments for a Boolean formula is a #P-complete problem (see, e.g., [Val79]). Further, in [Ott06] it is shown that the language class $\mathcal{L}(\mathsf{RWW})$ is reducible in linear time to the language class $\mathcal{L}(\mathsf{R})$ by using a further encoding technique. This means that for each language $L \in \mathcal{L}(\mathsf{RWW})$, there exists an encoded version $L' \in \mathcal{L}(\mathsf{R})$ that can be obtained from L in linear time. It follows that the result above can be extended to the class $\mathcal{L}(\mathsf{R}, \overline{\mathbb{N}} \times \mathbb{N}, \mathbb{H}_{exp})$. Hence, we see that by using the acceptance condition relative to subsets of the family $\mathbb{H}_{exp}$, even wR-automata can accept quite hard languages.

Now we continue with a closure property for the language classes of the forms $\mathcal{L}(\mathsf{RRWW}, \mathbb{Z}_\infty, \mathbb{H})$ and $\mathcal{L}(\mathsf{RRWW}, \overline{\mathbb{N}} \times \mathbb{N}, \mathbb{H})$. For this closure property we need the following lemma.

Lemma 6.2.2. *An* RRWW-*automaton is equivalent to an* RRWW-*automaton that makes no rewrite step during a tail computation.*

Proof. Let M be an RRWW-automaton. By Lemma 3.1.1 we can assume that M makes a restart or an accept step only when it sees the right border marker \$ in its read/write window. We now construct an RRWW-automaton M' such that M' makes no rewrite step during a tail computation, and $L(M') = L(M)$.

The automaton M' guesses after a restart step whether to simulate a cycle or a tail of M. In the former case M' just executes move-right, rewrite, and restart steps exactly as M; in the latter case it executes move-right and accept steps as M, and if the tail of M contains a rewrite step, then this can be replaced by some move-right steps of M', and M' enters the corresponding state that is reached through this rewrite step. □

Theorem 6.2.2 ([Wan17]).
The language classes $\mathcal{L}(\mathsf{RRWW}, \mathbb{Z}_\infty, \mathbb{H})$ and $\mathcal{L}(\mathsf{RRWW}, \overline{\mathbb{N}} \times \mathbb{N}, \mathbb{H})$ are closed under the operation of reversal for each family $\mathbb{H}$ of subsets of $\mathbb{Z}_\infty$ or $\overline{\mathbb{N}} \times \mathbb{N}$.

Proof. It is shown in [JLNO04] that the language class $\mathcal{L}(\mathsf{RRWW})$ is closed under the operation of reversal, and here we just apply the same technique.

Let $\mathcal{M} = (M, \omega)$ be a wRRWW-automaton, where $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is an RRWW-automaton with tape alphabet Γ, and ω is a weight function from the transitions of M to the semiring $\overline{\mathbb{N}} \times \mathbb{N}$, and let $T \in \mathbb{H}$. Without loss of generality we assume that M performs a restart or an accept step only when it sees the right border marker \$ in its read/write window. This can be realized by replacing any other restart transition by a move-right step with the same weight as the corresponding restart step, and by assigning the weight $(0, 1)$ to all additional move-right steps. In order to prove this closure property, we construct a wRRWW-automaton $\mathcal{M}' = (M', \omega')$ and a set $T' \in \mathbb{H}$, where $M' = (Q', \Sigma, \Gamma, ¢, \$, q_0', k', \delta')$ is an RRWW-automaton, and ω' is a weight function from the transitions of M' to the semiring $\overline{\mathbb{N}} \times \mathbb{N}$, such that $L_{T'}(\mathcal{M}') = L_T(\mathcal{M})^R$.

Recall that the transition function of M can be described by a set of meta-instructions of the form $(¢ \cdot E_1, u \to v, E_2 \cdot \$)$ or $(¢ \cdot E_0 \cdot \$, \mathsf{Accept})$, where E_0, E_1 and E_2 are regular languages, $u, v \in \Gamma^*$, and $|v| < |u|$ [NO01]. For each meta-instruction of this form of M, let M' have a meta-instruction $(¢ \cdot E_2^R, u^R \to v^R, E_1^R \cdot \$)$ or $(¢ \cdot E_0^R \cdot \$, \mathsf{Accept})$, where $E_i^R = \{w^R \mid w \in E_i\}$ for $i \in \{0, 1, 2\}$. This means that for a cycle C of M starting from a restarting configuration $q_0¢xuyz\$$, where C looks as follows:

$$\begin{aligned} q_0¢xuyz\$ &\vdash_M^* ¢xq_luyz\$ \vdash_M ¢xvq_myz\$ \\ &\vdash_M^* ¢xvyq_rz\$ \vdash_M q_0¢xvyz\$, \end{aligned}$$

M' has to transform the configuration $q_0'¢z^Ry^Ru^Rx^R\$$ into the configuration $q_0'¢z^Ry^Rv^Rx^R\$$. It remains to ensure that the weight of the cycle of M' simulating the cycle C is equal to $\omega(C)$. We assume that $|x|, |y| \geq k$. Further, let $x = x_1x_2 \ldots x_n$, let $y = y_1y_2 \ldots y_s$, let $u = u_1u_2 \ldots u_k$, where $n, s \geq k$, and let $t_{MVR,1}, t_{MVR,2}, \ldots, t_{MVR,k-1}$ are the last $k-1$ move-right steps before the rewrite step in the cycle C. It follows that the transitions $t_{MVR,1}$, $t_{MVR,2}$, …, $t_{MVR,k-1}$ are based on the window contents

$$\begin{gathered} x_{n-k+2}x_{n-k+3} \ldots x_nu_1, \\ x_{n-k+3}x_{n-k+4} \ldots x_nu_1u_2, \\ \vdots \\ x_nu_1u_2 \ldots u_{k-1}, \end{gathered}$$

respectively.

However, if M' simply performs the rewrite step $u^R \to v^R$, then it cannot execute the move-right steps that correspond to the transitions of $t_{MVR,1}$, $t_{MVR,2}$, …, $t_{MVR,k-1}$ of M, as after this rewrite step the read/write window is placed immediately to the right of v. In addition, the last $k-1$ move-right steps of M', say $t'_{MVR,1}, t'_{MVR,2}, \ldots, t'_{MVR,k-1}$, before this rewrite step are based on the window contents

$$\begin{gathered} y_{k-1}y_{k-2}\cdots y_1u_k, \\ y_{k-2}y_{k-3}\cdots y_1u_ku_{k-1}, \\ \vdots \\ y_1u_ku_{k-1}\dots u_2, \end{gathered}$$

respectively, and the move-right transitions corresponding to them are not executed during the cycle C.

In order to solve this problem, let M' be an RRWW-automaton with a read/write window of size $2k$, such that M' can simulate the rewrite step of M in advance without performing the move-right steps $t'_{MVR,1}$, $t'_{MVR,2}$, …, $t'_{MVR,k-1}$. Further, M' stores the string u^R within its finite-state control, and we define ω' such that it combines the sum of the weights of $t_{MVR,1}$, $t_{MVR,2}$, …, $t_{MVR,k-1}$ with the weight of the move-right transition after the rewrite step (see the Case 8 in the proof of Theorem 4.3.1). For other transitions, let ω' assign the same weights to these transitions as the corresponding transitions of M.

By Lemma 6.2.2 we can assume that no tail computation of M contains a rewrite step, that is, a tail computation consists of some move-right steps and an accept step, and thus it can easily be simulated in an analogous way. As both the semirings $\overline{\mathbb{N}} = (\overline{\mathbb{N}}, \max, +, -\infty, 0)$ and $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$ are commutative, the semiring $\overline{\mathbb{N}} \times \mathbb{N}$ is also commutative. Hence, for each input w, it holds that $f^{M'}_{\omega'}(w) = f^M_\omega(w^R)$. Finally, by taking $T' = T$, we obtain that $L_{T'}(\mathcal{M}') = L_T(\mathcal{M})^R$.

As the tropical semiring $\mathbb{Z}_\infty$ is commutative, using essentially the same technique, this closure property can also be proved for the tropical semiring $\mathbb{Z}_\infty$. □

Finally, we close this section with some results on the membership problem of the language classes $\mathcal{L}(\mathsf{RRWW}, \mathbb{Z}_\infty, \mathbb{H})$ and $\mathcal{L}(\mathsf{RRWW}, \overline{\mathbb{N}} \times \mathbb{N}, \mathbb{H})$.

Theorem 6.2.3. *For a weighted restarting automaton $\mathcal{M}$, and a subset T of the tropical semiring $\mathbb{Z}_\infty$, the membership problem for the language $L_T(\mathcal{M})$ is solvable deterministically in time $O(2^{n^2} \cdot n^2 \cdot \log n)$.*

Proof. Let $\mathcal{M} = (M, \omega)$ be a weighted restarting automaton over the tropical semiring $\mathbb{Z}_\infty$, and let T be a subset of $\mathbb{Z}_\infty$. Actually, the time consumption consists of the following two phases. The first phase is used for determining whether $w \in L(M)$, and in the second one it needs to be verified whether $f^M_\omega(w) \in T$.

First, we consider the time consumption of the first phase. For each input of length n, M can execute at most n cycle, and each cycle consists of at most n steps (see, e.g., [Ott06]). Therefore, for each input w, it can be determined in square time whether $w \in L(M)$ or not, which yields the nondeterministic polynomial time bound $O(n^2)$, that is, it is solvable deterministically in time $O(2^{n^2})$.

It remains to check whether $f_\omega^M(w) \in T$ for a given input w. Let $AC_M(w)$ be the set of accepting computations of M on the input w, and then $f_\omega^M(w) = \min\{\omega(AC) \mid AC \in AC_M(w)\}$. In order to determine the minimal $\omega(AC)$ for $AC \in AC_M(w)$, we need to compute the weight of each accepting computation AC. It is rather clear that there exists a constant c such that $|AC| \leq c \cdot n^2$, and thus $\omega(AC)$ is the sum of at most $c \cdot n^2$ numbers. It follows that there exists a constant d such that $\omega(AC) \leq d \cdot n^2$. As each addition operation can be done in time $O(\log n)$, $\omega(AC)$ is computable in time $O(n^2 \cdot \log n)$. In the proof of Theorem 4.2.1 it is shown that for each input of length n, the maximal number of accepting computations is bounded by $O(2^{n^2})$. Hence, it can be determined whether $f_\omega^M(w) \in T$ for a given input w of length n in time $O(2^{n^2} \cdot n^2 \cdot \log n)$. It follows that the time bound for this membership problem is

$$O(2^{n^2}) + O(2^{n^2} \cdot n^2 \cdot \log n) = O(2^{n^2} \cdot n^2 \cdot \log n).$$

□

Theorem 6.2.4. *For a weighted restarting automaton $\mathcal{M}$, and a subset T of the semiring $\overline{\mathbb{N}} \times \mathbb{N}$, the membership problem for the language $L_T(\mathcal{M})$ is solvable deterministically in time $O(2^{n^2} \cdot n^2 \cdot (\log n)^2)$.*

Proof. Let $\mathcal{M} = (M, \omega)$ be a wX-automaton over the semiring $\overline{\mathbb{N}} \times \mathbb{N}$, and let T be a subset of $\overline{\mathbb{N}} \times \mathbb{N}$. In analogy to the previous proof, the time consumption for determining whether $w \in L(M)$ is bounded by $O(2^{n^2})$, and thus we restrict our attention to the second phase. Given an input w, assume that $f_\omega^M(w) = (m_1, n_1)$, and if $f_\omega^M(w) \in T$, there exists a pair $(m_2, n_2) \in T$ such that $m_1 = m_2$ and $n_1 = n_2$. Since the operations of the semiring $\overline{\mathbb{N}} = (\overline{\mathbb{N}}, \max, +, -\infty, 0)$ are similar to those of the tropical semiring $\mathbb{Z}_\infty$, m_1 can be computed in time $O(2^{n^2} \cdot n^2 \cdot \log n)$ as shown in the proof of Theorem 6.2.3. In the following we consider the time consumption for computing n_1.

For the semiring $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$, if AC is an accepting computation of M, and $\omega(AC) = (m_{AC}, n_{AC})$, then n_{AC} is the product of the weights of the transitions used during the computation AC. As shown above, there exists a constant c such that $|AC| \leq c \cdot n^2$, and thus $n_{AC} \leq d^{n^2}$ for some constant d. As each multiplication operation can be completed in time $O((\log n)^2)$, n_{AC} can be computed in time $O(n^2 \cdot (\log n)^2)$. It follows that the time consumption for computing all accepting computations is bounded by $O(2^{n^2} \cdot n^2 \cdot (\log n)^2)$. Finally, as the sum of the weights of two accepting computations can be computed in time $O(n^2)$, adding up the weights of all accepting computations can be done in time $O(2^{n^2} \cdot n^2)$. It follows that the time bound for computing n_1 is

$$O(2^{n^2} \cdot n^2 \cdot (\log n)^2) + O(2^{n^2} \cdot n^2) = O(2^{n^2} \cdot n^2 \cdot (\log n)^2).$$

Hence, the time consumption for the second phases is

$$O(2^{n^2} \cdot n^2 \cdot \log n) + O(2^{n^2} \cdot n^2 \cdot (\log n)^2) = O(2^{n^2} \cdot n^2 \cdot (\log n)^2),$$

which is also the time bound for both phases. □

6.3 On the Classes of Languages Accepted by Word-Weighted Restarting Automata

In this section we study the classes of languages that are accepted by weighted restarting automata relative to subsets of a semiring of regular languages $\mathsf{REG}(\Delta) = (\mathsf{REG}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$. In general, the weight of a transition of a restarting automaton M can be any regular language over Δ. However, here we only consider word-weighted restarting automata that have been introduced in Chapter 5. For these word-weighted restarting automata, we define the following notion of relative acceptance.

Definition 6.3.1 ([WO16b]). *Let $\mathcal{M} = (M, \omega)$ be a $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automaton with input alphabet Σ, where ω maps the transitions of M to singleton sets over Δ.*

(a) *For a set $T \in \mathsf{REG}(\Delta)$, $\hat{L}_T(\mathcal{M}) = \{\, w \in L(M) \mid f_\omega^M(w) \cap T \neq \emptyset \,\}$ is the language* accepted by $\mathcal{M}$ relative to *the set T, that is, a word $w \in \Sigma^*$ belongs to the language $\hat{L}_T(\mathcal{M})$ iff $w \in L(M)$ and $f_\omega^M(w)$ contains at least one element of T.*

(b) *Let $\mathbb{H}$ be a family of subsets of $\mathsf{REG}(\Delta)$. Then*

$$\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathbb{H}) = \{\, \hat{L}_T(\mathcal{M}) \mid \mathcal{M} \text{ is a } \mathsf{w}_{\mathsf{word}}\mathsf{X}\text{-automaton and } T \in \mathbb{H}\}$$

is the class of languages that are accepted by $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automata relative to $\mathbb{H}$.

For word-weighted restarting automata we have the following inclusion result.

Lemma 6.3.1 ([WO16b]). *For all $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$,*

$$\mathcal{L}(\mathsf{nf\text{-}X}) \subseteq \hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)).$$

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a non-forgetting restarting automaton of type X. In order to prove the above inclusion, we construct a $\mathsf{w_{word}X}$-automaton $\mathcal{M}' = (M', \omega)$ and we define a set $T \in \mathsf{REG}(\Delta)$ such that $L(M) = \hat{L}_T(\mathcal{M}')$. The main problem in simulating M is the fact that, when executing a restart step, M can enter any state, while M' must return to its initial state q_0. To overcome this problem, for each restart transition $t : (q, \mathsf{Restart}) \in \delta(p, u)$ of M, the automaton M' will have a restart transition $t' : \mathsf{Restart} \in \delta'(p, u)$ with associated weight $\omega(t') = \{q\}$. Further, when starting from a restarting configuration, M' guesses the state q of M with which M begins the current cycle, and it then proceeds to simulate the next cycle of M starting in this state. In addition, the corresponding transition is given the weight $\{q\}$, which means that M' outputs its guessed state again in the new cycle. If in each cycle, M' guesses the correct state of M, then the weight of the resulting accepting computation of M' is of the form $\{q_0q_1q_1q_2q_2 \dots q_nq_n\}$ for some $q_1, q_2, \dots, q_n \in Q$ and $n \geq 0$. Accordingly, we take $\Delta = Q$ and $T = \{\, q_0q_1q_1q_2q_2 \dots q_nq_n \mid q_1, q_2, \dots, q_n \in Q, n \geq 0 \,\}$. To realize the above simulation, we take $M' = (Q', \Sigma, \Gamma, ¢, \$, q_0, k, \delta')$, where $Q' = Q \cup \{\, (q, q_1, q_2) \mid q, q_1, q_2 \in Q \,\}$, and the transition relation δ' and the weight function ω are defined as shown below:

1. First, in order to allow M' to guess the state with which M begins the current cycle, δ' contains the transition $t' : ((p, q, q'), op) \in \delta'(q_0, ¢u)$ with associated weight $\{q\}$ for each transition $t : (p, op) \in \delta(q, ¢u)$ of M and each $q, q' \in Q$. In a state of the form (p, q, q'), the first state component p is the current state of M, the second component q is the guessed state with which the current cycle of M begins, and the third component q' is the guessed state that M will enter through the next restart step (if any).

2. In a state of the form (p, q, q'), M' proceeds just as M proceeds in state p, leaving state components 2 and 3 untouched, until it reaches a restart transition. All these move-right, rewrite and accept steps of M' have weight $\{\lambda\}$.

3. Finally, for each restart transition of the form $(q', \mathsf{Restart}) \in \delta(q, u)$, δ' contains the transitions $t_{q_1} : \mathsf{Restart} \in \delta'((q, q_1, q'), u)$ for all $q_1 \in Q$, which all have weight $\{q'\}$.

For an input $w \in \Sigma^*$, M' may have many more accepting computations than M. In fact, in general, $L(M')$ will be a proper superset of $L(M)$. However, $f_\omega^{M'}(w) \cap T \neq \emptyset$, iff M' has an accepting computation AC on input w such that $\omega(AC) \in T$, that is, $\omega(AC)$ is of the form $q_0q_1q_1q_2q_2 \dots q_nq_n$. This means that within this computation, M' always guesses the correct state after each restart step, which shows that AC is the correct simulation of an accepting computation of M. It follows that $\hat{L}_T(\mathcal{M}) = L(M)$. □

In fact, also the converse inclusions hold.

Lemma 6.3.2 ([WO16b]). *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$,

$$\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) \subseteq \mathcal{L}(\mathsf{nf}\text{-}\mathsf{X}).$$

Proof. Let Δ be a finite alphabet, let $\mathcal{M} = (M, \omega)$ be a $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automaton, where $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is a restarting automaton of type X, and ω is a weight function that assigns to each transition of M a subset of Δ^* of cardinality one, and let $T \in \mathsf{REG}(\Delta)$. In order to prove the above inclusion we provide a non-forgetting restarting automaton M' of type X such that $L(M') = \hat{L}_T(\mathcal{M})$. For each $w \in \Sigma^*$, we have $w \in \hat{L}_T(\mathcal{M})$ iff $w \in L(M)$ and $f_\omega^M(w) \cap T \neq \emptyset$. As the set T is regular, there exists a DFA $A = (Q_A, \Delta, \delta_A, q_0^A, F_A)$ such that $L(A) = T$, where Q_A is a finite set of states, Δ is the input alphabet for A, $\delta_A : Q_A \times \Delta \to Q_A$ is the transition function, q_0^A is the initial state, and F_A is a set of accepting states. For an input $w \in \Sigma^*$, M' has to check whether $w \in L(M)$ and whether $f_\omega^M(w) \cap L(A) \neq \emptyset$. Therefore, M' needs to simulate both M and A simultaneously. Accordingly, each state of M' is a pair $[p, q]$, where $p \in Q$ and $q \in Q_A$, and when simulating a step of M, M' needs to ensure that A has a transition that is applicable to the weight of this step. As it is non-forgetting, M' can always remember the actual state of A, even after executing a restart step. If M accepts, then M' also accepts, provided that A reaches a final state by reading the weight of the current accept step.

Now we describe the non-forgetting restarting automaton M' in detail. Let $M' = (Q', \Sigma, \Gamma, ¢, \$, q_0', k, \delta')$ be a non-forgetting restarting automaton of type X that is defined by taking $Q' = \{[p, q] \mid p \in Q, q \in Q_A\}$, $q_0' = [q_0, q_0^A]$, and the transition function δ' is as described below.

1. First we define some transitions that allow M' to simulate move-right steps of M. If δ contains a move-right transition of the form

   $$t : (p, u) \to (q, \mathsf{MVR})$$

   for some $p, q \in Q$ and $\omega(t) = \{u'\}$, and if $\delta_A^*(z, u') \neq \emptyset$ for some $z \in Q_A$, then δ' contains the transition

   $$([p, z], u) \to ([q, \delta_A^*(z, u')], \mathsf{MVR}).$$

2. Next we define some transitions that allow M' to simulate rewrite steps of M. If δ contains a rewrite transition of the form

   $$t : (p, u) \to (q, v)$$

for some $p, q \in Q$ and $\omega(t) = \{u'\}$, and if $\delta_A^*(z, u') \neq \emptyset$ for some $z \in Q_A$, then δ' contains the transition

$$([p, z], u) \to ([q, \delta_A^*(z, u')], v).$$

3. Now we define those transitions that allow M' to simulate restart steps of M. If δ contains a rewrite transition of the form

$$t : (p, u) \to \mathsf{Restart}$$

for some $p \in Q$ and $\omega(t) = \{u'\}$, and if $\delta_A^*(z, u') \neq \emptyset$ for some $z \in Q_A$, then δ' contains the transition

$$([p, z], u) \to ([q_0, \delta_A^*(z, u')], \mathsf{Restart}).$$

4. Finally, we define those transitions that allow M' to simulate accept steps of M. If δ contains an accept transition of the form

$$t : (p, u) \to \mathsf{Accept}$$

for some $p \in Q$ and $\omega(t) = \{u'\}$, and if there exists $q \in F_A$ such that $q \in \delta_A^*(z, u')$ for some $z \in Q_A$, then δ' contains the transition

$$([p, z], u) \to \mathsf{Accept}.$$

It is easily seen that $L(M') = \hat{L}_T(\mathcal{M})$, which completes this proof. □

Together, Lemmas 6.3.1 and 6.3.2 yield the following characterization.

Theorem 6.3.1 ([WO16b]). *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$,

$$\mathcal{L}(\mathsf{nf\text{-}X}) = \hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)).$$

It is well-known that the classes of languages that are computed by (nf-) mon-RWW- and (nf-)mon-RRWW-automata coincide with the class of context-free languages (see, e.g., [MO06, JMPV99]). Based on this result a characterization of the class CFL of context-free languages in terms of word-weighted restarting automata can be derived. Hence, the following equalities can be established.

Theorem 6.3.2.

$$\mathsf{CFL} = \begin{cases} \mathcal{L}(\mathsf{mon\text{-}RWW}) \\ \mathcal{L}(\mathsf{mon\text{-}RRWW}) \\ \mathcal{L}(\mathsf{nf\text{-}mon\text{-}RWW}) \\ \mathcal{L}(\mathsf{nf\text{-}mon\text{-}RRWW}) \\ \hat{\mathcal{L}}(\mathsf{mon\text{-}RWW}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) \\ \hat{\mathcal{L}}(\mathsf{mon\text{-}RRWW}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) \end{cases}$$

We have seen that by using an acceptance condition relative to subsets of the semiring of regular languages, the expressive power of mon-RWW- and mon-RRWW-automata cannot be increased. Further, this type of acceptance condition can also be used to replace restarting automata with auxiliary symbols by automata without auxiliary symbols, as a nf-RRW-automaton (or a nf-RW-automaton) can easily simulate the computations of an RRWW-automaton (or an RWW-automaton). Hence, the following result can be established.

Corollary 6.3.1. *For* $\mathsf{X} \in \{\mathsf{RRW}, \mathsf{RW}\}$*,* $\mathcal{L}(\mathsf{XW}) \subseteq \hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$*.*

In the following we continue with some closure properties of the language class $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$.

Theorem 6.3.3 ([WO16b])**.** *The class* $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$ *is closed under the operation of union for each* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*.*

Proof. Let $\mathcal{M}_1 = (M_1, \omega_1)$ and $\mathcal{M}_2 = (M_2, \omega_2)$ be $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automata, and let $T_1, T_2 \in \mathsf{REG}(\Delta)$. By Theorem 6.3.1, there exist non-forgetting restarting automata M_1' and M_2' of type X such that $L(M_1') = L_{T_1}(\mathcal{M}_1)$ and $L(M_2') = L_{T_2}(\mathcal{M}_2)$. Thus, in order to prove the above closure property, it suffices to construct a non-forgetting restarting automaton M of type X such that $L(M) = L(M_1') \cup L(M_2')$. At the start, M nondeterministically chooses an index $i \in \{1, 2\}$, and then it simply works exactly like M_i'. As M is non-forgetting, it can store its guess within its finite-state control. If M_i' accepts, then M also accepts. Thus, M accepts on input w iff at least one of M_1' or M_2' accepts on input w. It follows that $L(M) = L(M_1') \cup L(M_2')$. □

In [JLNO04] it is shown that the language classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ are closed under the operation of concatenation. This result also holds for the class of languages that are defined by word-weighted restarting automata.

Theorem 6.3.4 ([WO16b])**.** *The class* $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$ *is closed under the operation of concatenation for each* $\mathsf{X} \in \{\mathsf{RWW}, \mathsf{RRWW}\}$*.*

Proof. The proof for RWW- and RRWW-automata given in [JLNO04] proceeds as follows. On input a word w, a factorization $w = uv$ is guessed such that u is accepted by the first automaton and v is accepted by the second. To fix this guess, the last symbol a of u and the first symbol b of v are rewritten into a special symbol $[a, b]$, and then the first automaton is simulated on u. If and when it accepts, then the second automaton is simulated on v. In the same way, we can proceed for nf-RWW- and nf-RRWW-automata. By Theorem 6.3.1 this yields the intended closure property for $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$. □

Now we return to the operation of reversal. In [JLNO04] it is shown that the class of languages that are accepted by RRWW-automata is closed under reversal. As the proof carries over to nf-RRWW-automata, we immediately obtain the following result from Theorem 6.3.1.

Theorem 6.3.5 ([WO16b]). *The class $\hat{\mathcal{L}}(\mathsf{RRWW}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$ is closed under the operation of reversal.*

Finally, we close this section with a result on the membership problem of the language class $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$.

Theorem 6.3.6. *For a weighted restarting automaton $\mathcal{M}$ of type X, and a subset $T \in \mathsf{REG}(\Delta)$, the membership problem for the language $\hat{L}_T(\mathcal{M})$ is solvable nondeterministically in time $O(n^2)$.*

Proof. Let $\mathcal{M}$ be a $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automaton over the semiring $\mathsf{REG}(\Delta)$, and let $T \in \mathsf{REG}(\Delta)$, then $\hat{L}_T(\mathcal{M}) \in \hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$. By Theorem 6.3.1, there exists a non-forgetting restarting automaton M' of type X such that $L(M') = L_T(\mathcal{M})$. By Theorem 6.2.3, we know that for each input word w, it can be determined whether $w \in L(M')$ with the nondeterministic time bound $O(n^2)$. □

6.4 A Stronger Restriction for Word-Weighted Restarting Automata

A word $w \in \Sigma^*$ is an element of the language $\hat{L}_T(\mathcal{M})$ for a word-weighted restarting automaton $\mathcal{M} = (M, \omega)$ and a set $T \in \mathsf{REG}(\Delta)$, if $w \in L(M)$ and the weight $\omega(AC)$ is an element of T for at least one accepting computation AC of M on input w. Thus, there may be other accepting computations of M on this very input that have an associated weight that does not belong to the set T. The following definition requires that $\omega(AC)$ must belong to T for *each* accepting computation AC of M on input w.

Definition 6.4.1 ([WO16b]). *Let $\mathcal{M} = (M, \omega)$ be a $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automaton with input alphabet Σ, where ω maps the transitions of M to singleton sets over Δ. For a set $T \subseteq \mathbb{P}_{\text{fin}}(\Delta^*)$, $L_T(\mathcal{M}) = \{\, w \in L(M) \mid f^M_\omega(w) \in T \,\}$ is the language* strongly accepted by $\mathcal{M}$ relative to *the set T, that is, a word $w \in \Sigma^*$ belongs to the language $L_T(\mathcal{M})$ iff $w \in L(M)$ and $f^M_\omega(w)$ is an element of T.*

Actually, this definition is exactly in the spirit of Definition 6.1.1. Indeed, let S be the semiring of formal languages over Δ. For a $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automaton $\mathcal{M} = (M, \omega)$ and an input $w \in \Sigma^*$, the value $f^M_\omega(w)$ is a finite subset of Δ^*. Thus, it suffices to consider subsets of S that consist of finite languages.

Accordingly, if T is a collection of finite subsets of Δ^*, then an input word $w \in \Sigma^*$ belongs to the language $L_T(\mathcal{M})$ iff $w \in L(M)$ and $f_\omega^M(w)$ is an element of T.

We now consider the language class of the form $\mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H})$. In general, this class is quite rich, and the following result can be established.

Proposition 6.4.1. *For each language L over the alphabet Σ, there exists a* $\mathsf{stl\text{-}det\text{-}mon\text{-}w_{word}R}(1)$*-automaton $\mathcal{M}$ and a subset $T \subseteq \mathbb{P}_{\text{fin}}(\Sigma^*)$ such that $L_T(\mathcal{M}) = L$, that is, $L \in \mathcal{L}(\mathsf{stl\text{-}det\text{-}mon\text{-}R}(1), \mathbb{P}_{\text{fin}}(\Sigma^*), \mathbb{H})$ for some family $\mathbb{H}$ of subsets of $\mathbb{P}_{\text{fin}}(\Sigma^*)$.*

Proof. Let L be a language over the alphabet Σ, and let $\mathcal{M} = (M, \omega)$ be a $\mathsf{stl\text{-}det\text{-}mon\text{-}w_{word}R}(1)$-automaton that proceeds as follows. For each input $w \in \Sigma^*$, M just moves to the right end of the tape, and it accepts on seeing the right border marker \$. The weight of each reading step on the word w is the set of the current symbol that is in the read/write window of M, and the reading step on the left border marker ¢ and the accept step on the right border marker \$ have the associated weight $\{\lambda\}$. It is easily seen that M accepts all input words over the alphabet Σ, that is, $L(M) = \Sigma^*$, and $f_\omega^M(w) = \{w\}$ for all $w \in \Sigma^*$. Further, we take $T = \{\, \{w\} \mid w \in L \,\}$, which means that if $w \in L$, then $f_\omega^M(w) = \{w\} \in T$. Hence, it follows that $L_T(\mathcal{M}) = L$. □

In the following we compare the language class of the form $\mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H})$ to the languages classes that are introduced in the previous sections. However, we see that in the way described in the proof of Proposition 6.4.1 any language can be accepted by a $\mathsf{stl\text{-}det\text{-}mon\text{-}R}(1)$-automaton, which is the simplest type of restarting automaton. Therefore, we consider some restricted families of subsets from various semirings:

1. $\mathbb{H}^z_{fin}$ denotes the family of finite sets of integers.
2. $\mathbb{H}^N_{fin}$ denotes the family of finite sets of natural numbers.
3. $\mathbb{H}^{cfl(\Delta)}_{fin}$ denotes the family of sets of finite languages over some finite alphabet Δ such that, for each $T \in \mathbb{H}^{cfl(\Delta)}_{fin}$, $\bigcup\limits_{V \in T} V \in \mathsf{CFL}(\Delta)$.
4. $\mathbb{H}^{reg(\Delta)}_{fin}$ denotes the family of sets of finite languages over some finite alphabet Δ such that, for each $T \in \mathbb{H}^{reg(\Delta)}_{fin}$, $\bigcup\limits_{V \in T} V \in \mathsf{REG}(\Delta)$.

First, we start by investigating the inclusion relations between the language classes $\mathcal{L}(\mathsf{X}, \mathbb{Z}_\infty, \mathbb{H}^z_{fin})$ and $\mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}^{cfl(\Delta)}_{fin})$, and between their subclasses $\mathcal{L}(\mathsf{X}, \mathbb{N}_\infty, \mathbb{H}^N_{fin})$ and $\mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}^{reg(\Delta)}_{fin})$, where X is a type of restarting automaton.

Theorem 6.4.1. *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$,

$$\begin{array}{rlcl} (a) & \mathcal{L}(\mathsf{X}, \mathbb{Z}_\infty, \mathbb{H}^{z}_{fin}) & \subseteq & \mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}^{cfl(\Delta)}_{fin}), \\ (b) & \mathcal{L}(\mathsf{X}, \mathbb{N}_\infty, \mathbb{H}^{N}_{fin}) & \subseteq & \mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}^{reg(\Delta)}_{fin}). \end{array}$$

Proof. First, we prove the inclusion (a). Let $\mathcal{M} = (M, \omega)$ be a wX-automaton, and let $T \in \mathbb{H}^{z}_{fin}$. For this proof we will construct a word-weighted restarting automaton $\mathcal{M}'$ of type X and a subset $T' \in \mathbb{H}^{cfl(\Delta)}_{fin}$ such that $L_{T'}(\mathcal{M}') = L_T(\mathcal{M})$.

Let $\mathcal{M}' = (M', \omega')$ be a word weighted restarting automaton, where M' simply works exactly like M, and the weight function ω' over the output alphabet $\Delta = \{a, b\}$ is defined below. For each transition t of M,

$$\omega'(t) = \begin{cases} \{a^{|\omega(t)|}\}, & \text{if } \omega(t) \geq 0, \\ \{b^{|\omega(t)|}\}, & \text{otherwise.} \end{cases}$$

In this way, given an input word w, for each accepting computation $AC(w)$ of M with associated weight $\omega(AC(w)) = i \in \mathbb{Z}_\infty$, there exists a word $u \in \Delta^*$ such that $\omega'(AC(w)) = \{u\}$ and $|u|_a - |u|_b = i$. However, there may be other accepting computations on the word w, and $f^M_\omega(w) = \min\{\, \omega(AC) \mid AC \in AC_M(w)\,\}$. This implies that for all $u \in f^{M'}_{\omega'}(w)$, $|u|_a - |u|_b \geq i$, and there exists a word $u' \in f^{M'}_{\omega'}(w)$ satisfying $|u'|_a - |u'|_b = i$. Now we define the set

$$\begin{aligned} T' \;=\; \bigcup_{i \in T} \{\, V \mid \;& V \in \mathbb{P}_{\text{fin}}(\{\, w \mid |w|_a - |w|_b \geq i \text{ for } w \in \Delta^* \,\}) \\ & \text{and } \exists w \in V : |w|_a - |w|_b = i \,\}. \end{aligned}$$

It is easily seen that each set V of T' is finite, and $\min\{\, |u|_a - |u|_b \mid u \in V\,\} = i$ for some $i \in T$. For each $w \in \Sigma^*$, if $w \in L(M)$ and $f^M_\omega(w) = i \in T$, then $\mathcal{M}'$ has an accepting computation with weight $\{u\}$ for some $u \in \Delta^*$ satisfying $|u|_a - |u|_b = i$, and there is no accepting computation with a weight $\{u'\}$ such that $|u'|_a - |u'|_b < i$. This means that there is a finite set in T' that coincides with the set $f^{M'}_{\omega'}(w)$, and it follows that $w \in L_{T'}(\mathcal{M}')$. Finally, it should be obvious that

$$\bigcup_{V \in T'} V = \{\, w \mid |w|_a - |w|_b \geq i \text{ for } w \in \Delta^* \text{ and } i = \min T \,\} \in \mathsf{CFL}(\Delta),$$

and thus $T' \in \mathbb{H}^{cfl(\Delta)}_{fin}$.

In an analogous way, the inclusion (b) can also be obtained. □

Actually, following the same fundamental idea, the above inclusions can also be shown if the tropical semirings $\mathbb{Z}_\infty$ and $\mathbb{N}_\infty$ are replaced by the so-called *arctic semirings* $(\overline{\mathbb{Z}}, \max, +, -\infty, 0)$ and $(\overline{\mathbb{N}}, \max, +, -\infty, 0)$, respectively. In Section 6.3 the language class $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$ has been introduced, and in the same way, we can also define the language class $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{CFL}(\Delta), \mathsf{CFL}(\Delta))$. We continue by comparing the language class of the form $\mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H})$ to the language classes $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$ and $\hat{\mathcal{L}}(\mathsf{X}, \mathsf{CFL}(\Delta), \mathsf{CFL}(\Delta))$. As a deterministic restarting automaton has at most one accepting computation on each input, the following equivalences can be obtained immediately.

Corollary 6.4.1. *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*,*

$$\begin{array}{lrcl} (a) & \hat{\mathcal{L}}(\mathsf{det}\text{-}\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) & = & \mathcal{L}(\mathsf{det}\text{-}\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{reg(\Delta)}), \\ (b) & \hat{\mathcal{L}}(\mathsf{det}\text{-}\mathsf{X}, \mathsf{CFL}(\Delta), \mathsf{CFL}(\Delta)) & = & \mathcal{L}(\mathsf{det}\text{-}\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{cfl(\Delta)}). \end{array}$$

For the nondeterministic case we have the following inclusion result.

Theorem 6.4.2 ([Wan17])**.** *For all* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$*,*

$$\begin{array}{lrcl} (a) & \hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) & \subseteq & \mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{reg(\Delta)}), \\ (b) & \hat{\mathcal{L}}(\mathsf{X}, \mathsf{CFL}(\Delta), \mathsf{CFL}(\Delta)) & \subseteq & \mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{cfl(\Delta)}). \end{array}$$

Proof. First, we prove the inclusion (a). Let $L \in \hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta))$, then there exist a $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automaton $\mathcal{M} = (M, \omega)$ und a set $T \in \mathsf{REG}(\Delta)$ such that $\hat{L}_T(\mathcal{M}) = L$. Further, let $\mathcal{M}' = (M', \omega')$ be a $\mathsf{w}_{\mathsf{word}}\mathsf{X}$-automaton that simply works exactly as $\mathcal{M}$, and let the set

$$T' = \{V_1 \cup V_2 \mid V_1 \in \mathbb{P}_{\text{fin}}(T) \smallsetminus \{\emptyset\}, V_2 \in \mathbb{P}_{\text{fin}}(\overline{T})\},$$

where $\overline{T} = \Delta^* \smallsetminus T$. It is clear that $T' \in \mathbb{H}_{fin}^{reg(\Delta)}$, as $\bigcup\limits_{V \in T'} V = \Delta^* \in \mathsf{REG}(\Delta)$.

If $w \in \hat{L}_T(\mathcal{M})$, then M accepts on the input w, and $f_\omega^M(w)$ contains at least one word from the set T, that is, $1 \leq |f_\omega^M(w) \cap T|$. Let $V_1 = f_\omega^M(w) \cap T$, and let $V_2 = f_\omega^M(w) \smallsetminus T$. By the above definition of T', $V_1 \cup V_2 \in T'$ can be obtained. It follows that $f_\omega^M(w) \in T'$, and hence $w \in L_{T'}(\mathcal{M}')$. On the other hand, if $w \in L_{T'}(\mathcal{M}')$, then $w \in L(M')$ and $f_{\omega'}^{M'}(w) \in T'$. Further, by the definition of T', we obtain $f_{\omega'}^{M'}(w) \cap T \neq \emptyset$. Hence, the word w also belongs to $\hat{L}_T(\mathcal{M})$.

Along the same line, the inclusion (b) can also be shown. □

By Theorem 6.4.2, we obtain that a word-weighted restarting automaton with the stronger acceptance condition can simulate the computations of a non-forgetting restarting automaton of the corresponding type. It is still open whether the above inclusions are proper. We now consider the following language

$$L_{copy} = \{\, uu \mid u \in \Sigma^* \,\}.$$

Note that up to now it has not yet been shown that the language L_{copy} can be accepted by a nf-RR-automaton. However, using the stronger acceptance notion of word-weighted restarting automata, we have the following positive result.

Proposition 6.4.2 ([Wan17]). *$L_{copy} \in \mathcal{L}(\mathsf{R}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{reg(\Delta)})$.*

Proof. In order to prove this result, we will construct a $\mathsf{w_{word}R}$-automaton $\mathcal{M} = (M, \omega)$ and a subset $T \in \mathbb{H}_{fin}^{reg(\Delta)}$ such that $L_T(\mathcal{M}) = L_{copy}$. Let $M = (Q, \Sigma, \Sigma, ¢, \$, q_0, 2, \delta)$ be an R-automaton, where $Q = \{q_0\} \cup \{q_{r,x}, q_{r,\hat{x}} \mid x \in \Sigma \cup \{@\}\}$, and the transition function δ is defined as follows:

$$\begin{array}{rlcll}
t_{1,x}: & (q_0, ¢x) & \to & (q_0, \mathsf{MVR}) & \text{for all } x \in \Sigma, \\
t_{2,x}: & (q_0, ¢x) & \to & (q_{r,x}, ¢) & \text{for all } x \in \Sigma, \\
t_{2,\hat{@}}: & (q_0, ¢x) & \to & (q_{r,\hat{@}}, ¢) & \text{for all } x \in \Sigma, \\
t_{3,x_1x_2}: & (q_0, x_1x_2) & \to & (q_0, \mathsf{MVR}) & \text{for all } x_1, x_2 \in \Sigma, \\
t_{4,\hat{x}}: & (q_0, x\$) & \to & (q_{r,\hat{x}}, \$) & \text{for all } x \in \Sigma, \\
t_{4,@}: & (q_0, x\$) & \to & (q_{r,@}, \$) & \text{for all } x \in \Sigma, \\
t_5: & (q_0, ¢\$) & \to & \mathsf{Accept}, & \\
t_{6,x}: & (q_{r,x}, u) & \to & \mathsf{Restart} & \text{for all } x \in \{a, \hat{a} \mid a \in \Sigma \cup \{@\}\}, \\
 & & & & \text{and all admissible } u.
\end{array}$$

It is easily seen that M accepts all words $w \in \Sigma^*$. Further, let the alphabet $\Delta = \Sigma \cup \{\hat{a} \mid a \in \Sigma\} \cup \{@, \hat{@}\}$, and let ω be the weight function over Δ that is defined as follows:

$$\begin{array}{lcll}
\omega(t_{2,x}) & = & \{x\} & \text{for all } x \in \Sigma, \\
\omega(t_{2,\hat{@}}) & = & \{\hat{@}\}, & \\
\omega(t_{4,\hat{x}}) & = & \{\hat{x}\} & \text{for all } x \in \Sigma, \\
\omega(t_{4,@}) & = & \{@\}, & \\
\omega(t) & = & \{\lambda\} & \text{for all other transitions } t \in \delta.
\end{array}$$

Obviously, given an input word of the form $uu \in \Sigma^*$, where $u = a_1a_2 \dots a_n$ there are always two accepting computations with associated weights of the forms $\{a_1@a_2@\dots a_n@\}$ and $\{\hat{@}\hat{a}_n\hat{@}\hat{a}_{n-1}\dots\hat{@}\hat{a}_1\}$. This implies that M removes the first symbol of the first u and the last symbol of the second u, alternatingly. Let $T_1 = \{\{a_1@a_2@\dots a_n@, \hat{@}\hat{a}_n\hat{@}\hat{a}_{n-1}\dots\hat{@}\hat{a}_1\} \mid a_1, a_2, \dots, a_n \in \Sigma\}$, and then for each $w \in L_{copy}$, $f_\omega^M(w)$ contains a set from T_1. As there are also accepting computations with the weights that do not belong to any set from T_1, we define the following sets. Let $T_1' = \bigcup_{V \in T_1} V$, and let $T_2 = \Delta^* \smallsetminus T_1'$. Further, let $T = \{V_1 \cup V_2 \mid V_1 \in T_1, V_2 \in \mathbb{P}_{\text{fin}}(T_2)\}$. Obviously, $w \in L_{copy}$ if and only if $f_\omega^M(w) \in T$, and hence $L_T(\mathcal{M}) = L_{copy}$. Finally, as $\bigcup_{V \in T} V = \Delta^* \in \mathsf{REG}(\Delta)$, it follows that $T \in \mathbb{H}_{fin}^{reg(\Delta)}$, which completes this proof. □

Although it is not yet proved that $L_{copy} \notin \mathcal{L}(\mathsf{nf\text{-}RR})$, we conjecture that it is true, as without auxiliary symbols a restarting automaton cannot remember its guess for the border between the prefix and suffix. Hence, we give the following conjecture.

Conjecture 6.4.1.
For each $\mathsf{X} \in \{\mathsf{R}, \lambda\}$, $\hat{\mathcal{L}}(\mathsf{XR}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) \subsetneq \mathcal{L}(\mathsf{XR}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{reg(\Delta)})$.

Now we turn to the closure properties of the classes of languages that are accepted by word-weighted restarting automata with the stronger acceptance. We begin with the following important closure property.

Theorem 6.4.3 ([WO16b]). *Let* $\mathcal{M}_1 = (M_1, \omega_1)$ *and* $\mathcal{M}_2 = (M_2, \omega_2)$ *be* $\mathsf{w}_{\mathsf{word}}\mathsf{X}$*-automata, where* $\mathsf{X} \in \{\mathsf{RWW}, \mathsf{RRWW}\}$ *and* ω_1 *and* ω_2 *map the transitions of* M_1 *and* M_2 *to singleton sets over* Δ*, and let* $T_1, T_2 \subseteq \mathbb{P}_{\text{fin}}(\Delta^*)$*. Then there are an alphabet* Δ'*, a* $\mathsf{w}_{\mathsf{word}}\mathsf{X}$*-automaton* $\mathcal{M} = (M, \omega)$*, where* ω *maps the transitions of* M *to singleton sets over* Δ'*, and a set* $T \subseteq \mathbb{P}_{\text{fin}}(\Delta'^*)$ *such that* $L_T(\mathcal{M}) = L_{T_1}(\mathcal{M}_1) \cap L_{T_2}(\mathcal{M}_2)$.

Proof. For $i = 1, 2$, let $\mathcal{M}_i = (M_i, \omega_i)$, where $M_i = (Q_i, \Sigma, \Gamma_i, ¢, \$, q_0^{(i)}, k_i, \delta_i)$ is an RRWW-automaton, ω_i is a weight function that maps the transitions of M_i to singleton sets over Δ, and let $T_i \subseteq \mathbb{P}_{\text{fin}}(\Delta^*)$. We construct an alphabet Δ', a $\mathsf{w}_{\mathsf{word}}\mathsf{RRWW}$-automaton $\mathcal{M} = (M, \omega)$, where ω maps the transitions of M to singleton sets over Δ', and a subset $T \subseteq \mathbb{P}_{\text{fin}}(\Delta'^*)$ such that $L_T(\mathcal{M}) = L_{T_1}(\mathcal{M}_1) \cap L_{T_2}(\mathcal{M}_2)$, that is, for all $w \in \Sigma^*$, $w \in L_T(\mathcal{M})$ iff $w \in L_{T_1}(\mathcal{M}_1)$ and $w \in L_{T_2}(\mathcal{M}_2)$.

On input a word $w \in \Sigma^*$, the automaton M will be able to simulate M_1 as well as M_2. Essentially, the simulation of an accepting computation of M_1 on input w should give the same weight as the corresponding computation of M_1, and analogously, the simulation of an accepting computation of M_2 on input w should give the same weight as the corresponding computation of M_2. However, as the elements of T_1 and T_2 are subsets of Δ^*, it could happen that $f_{\omega_1}^{M_1}(w)$ is an element of T_2, although $w \notin L_{T_1}(M_1)$. Thus, we must ensure that the set of weights of the simulations of all accepting computations of M_1 for an input word w cannot be an element of T_2, and analogously for simulations of accepting computations of M_2 and T_1.

For this purpose, we define the following sets and mappings. Let $\Delta_1 = \Delta \cup \{@\}$, $\hat{\Delta} = \{\, \hat{a} \mid a \in \Delta \,\}$, $\Delta_2 = \hat{\Delta} \cup \{\hat{@}\}$, and let $\Delta' = \Delta_1 \cup \Delta_2$. Further, let σ_1, σ_2, and σ' be the mappings that are given through

$$\begin{array}{lcll}
\sigma_1(w) & = & w@ & \text{for } w \in \Delta^*, \\
\sigma_2(w) & = & \hat{a}_1\hat{a}_2 \ldots \hat{a}_n\hat{@} & \text{for } w = a_1a_2 \ldots a_n \in \Delta^*, \\
\sigma'(\lambda) & = & \lambda, & \\
\sigma'(w) & = & \hat{a}_1\hat{a}_2 \ldots \hat{a}_n & \text{for } w = a_1a_2 \ldots a_n \in \Delta^+,
\end{array}$$

which are extended to sets by simply applying them to all elements of a given set. Finally, let ω'_1 and ω'_2 be the weight functions that are defined as follows:

$$\begin{array}{rcll}
\omega'_1(t) & = & \{u@\} & \text{for each accept transition } t \in \delta_1, \text{where } \omega_1(t) = \{u\}, \\
\omega'_1(t) & = & \omega_1(t) & \text{for all other transitions } t \in \delta_1, \\
\omega'_2(t) & = & \{\sigma'(u)\hat{@}\} & \text{for each accept transition } t \in \delta_2, \text{where } \omega_2(t) = \{u\}, \\
\omega'_2(t) & = & \sigma'(\omega_2(t)) & \text{for all other transition } t \in \delta_2.
\end{array}$$

Now let $\mathcal{M}'_1 = (M_1, \omega'_1)$ and $\mathcal{M}'_2 = (M_2, \omega'_2)$, and let $T'_1 = \{\, \sigma_1(V) \mid V \in T_1 \,\}$ and $T'_2 = \{\, \sigma_2(V) \mid V \in T_2 \,\}$. Then $L_{T_1}(\mathcal{M}_1) = L_{T'_1}(\mathcal{M}'_1)$ and $L_{T_2}(\mathcal{M}_2) = L_{T'_2}(\mathcal{M}'_2)$. It is easily seen that $f^{M_1}_{\omega'_1}(w) \subseteq \Delta_1^+$ and $f^{M_2}_{\omega'_2}(w) \subseteq \Delta_2^+$ for each $w \in \Sigma^*$.

The RRWW-automaton $M = (Q, \Sigma, \Gamma, \text{¢}, \$, q_0, k, \delta)$ and the weight function ω are defined as follows. If $\max\{k_1, k_2\} = 1$, we take $k = 2$; otherwise, we take $k = \max\{k_1, k_2\}$. Starting from the initial configuration on input $w \in \Sigma^*$, M first guesses whether to simulate M_1 or M_2. In order to remember its guess, δ contains some transitions that allow M to combine the first two symbols a_1 and a_2 of w into a special auxiliary symbol of the form $[a_1, a_2, i]$, where $i \in \{1, 2\}$ is the above guess. The weight function ω assigns the set $\{\lambda\}$ to these transitions. In the subsequent cycles, M simulates the machine M_i on seeing the symbol $[a_1, a_2, i]$. Of course, the symbol $[a_1, a_2, i]$ leads to some adjustments in the construction of the transitions of M that simulate M_1 and M_2. However, this technique has already been presented in detail in the proof of Theorem 4.3.1. An accepting computation of M with the weight $\omega'_i(AC)$ can be obtained by simulating an accepting computation AC of M_i on input w. Finally, let $T = \{\, V_1 \cup V_2 \mid V_1 \in T'_1 \text{ and } V_2 \in T'_2 \,\}$. Then, for each $w \in \Sigma^*$, $w \in L_T(\mathcal{M})$ iff $w \in L(M_1) \cup L(M_2)$ and it holds that $f^M_\omega(w) \in T$. The latter means that there exist a subset $V_1 \in T'_1$ and a subset $V_2 \in T'_2$ such that $f^M_\omega(w) = V_1 \cup V_2$. This implies that $f^{M_1}_{\omega'_1}(w) = V_1$ and $f^{M_2}_{\omega'_2}(w) = V_2$, which means in particular that both, M_1 and M_2, accept on input w. It follows that $L_T(\mathcal{M}) = L_{T'_1}(\mathcal{M}'_1) \cap L_{T'_2}(\mathcal{M}'_2)$. For RWW-automata, the result can be proved in exactly the same way. □

This is the first result that shows that a class of languages defined in terms of a quite general class of restarting automata is closed under the operation of intersection. Using essentially the same technique also the following closure property can be derived.

Theorem 6.4.4 ([WO16b]). *Let $\mathcal{M}_1 = (M_1, \omega_1)$ and $\mathcal{M}_2 = (M_2, \omega_2)$ be* $\mathsf{w_{word}X}$*-automata, where $\mathsf{X} \in \{\mathsf{RWW}, \mathsf{RRWW}\}$ and ω_1 and ω_2 map the transitions of M_1 and M_2 to singleton sets over Δ, and let $T_1, T_2 \subseteq \mathbb{P}_{\text{fin}}(\Delta^*)$. Then there are an alphabet Δ', a* $\mathsf{w_{word}X}$*-automaton $\mathcal{M} = (M, \omega)$, where ω maps the transitions of M to singleton sets over Δ', and a set $T \subseteq \mathbb{P}_{\text{fin}}(\Delta'^*)$ such that $L_T(\mathcal{M}) = L_{T_1}(\mathcal{M}_1) \cup L_{T_2}(\mathcal{M}_2)$.*

In fact, we have also the following result.

Theorem 6.4.5 ([WO16b]). *Let $\mathcal{M}_1 = (M_1, \omega_1)$ and $\mathcal{M}_2 = (M_2, \omega_2)$ be $\mathsf{w_{word}X}$-automata, where $\mathsf{X} \in \{\mathsf{RWW}, \mathsf{RRWW}\}$ and ω_1 and ω_2 map the transitions of M_1 and M_2 to singleton sets over Δ, and let $T_1, T_2 \subseteq \mathbb{P}_{\text{fin}}(\Delta^*)$. Then there are an alphabet Δ', a $\mathsf{w_{word}X}$-automaton $\mathcal{M} = (M, \omega)$, where ω maps the transitions of M to singleton sets over Δ', and a set $T \subseteq \mathbb{P}_{\text{fin}}(\Delta'^*)$ such that $L_T(\mathcal{M}) = L_{T_1}(\mathcal{M}_1) \cdot L_{T_2}(\mathcal{M}_2)$.*

Proof. It is known that the language classes $\mathcal{L}(\mathsf{RWW})$ and $\mathcal{L}(\mathsf{RRWW})$ are closed under the operation of concatenation [JLNO04]. The central idea of the proof is to guess a factorization $w = uv$ for an input w, to combine the last symbol a of u and the first symbol b of v into a special symbol $[a, b]$ in order to fix this guess, and to simulate the first automaton on u and the second on v. Using the alphabets and mappings from the proof of Theorem 6.4.3, and by taking $T = \{\, V_1 \cdot V_2 \mid V_1 \in T_1' \text{ and } V_2 \in T_2' \,\}$, the simulation technique from [JLNO04] can be used. □

Finally, we close this section with the following result.

Theorem 6.4.6. *Let $\mathcal{M} = (M, \omega)$ be a $\mathsf{w_{word}X}$-automata, where $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$ and ω maps the transitions of M to singleton sets over Δ, and let $T \subseteq \mathbb{P}(\Delta^*)$. Then there are an alphabet Δ', a $\mathsf{w_{word}X}$-automaton $\mathcal{M}' = (M', \omega')$, where ω' maps the transitions of M' to singleton sets over Δ', and a set $T' \subseteq \mathbb{P}_{\text{fin}}(\Delta'^*)$ such that $L_T(\mathcal{M}) = \overline{L_{T'}(\mathcal{M}')}$.*

Proof. We can prove the above result by using the technique from [Wan17]. Let $\mathcal{M} = (M, \omega)$ be a $\mathsf{w_{word}X}$-automaton, where $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is a restarting automaton of type X and ω maps the transitions of M to singleton sets over Δ, and let $T \subseteq \mathbb{P}(\Delta^*)$. For each input $w \in \Sigma^*$, if $w \notin L_T(\mathcal{M})$, then this means that either $w \notin L(M)$, or $w \in L(M)$ but $f_\omega^M(w) \notin T$. Let M' be a restarting automaton of type X that performs each transition t exactly as M if $t \in \delta$. Further, for each $q \in Q$ and each admissible window content u, if $\delta(q, u) = \emptyset$, then M' contains the accept transition $(q, u) \to \mathsf{Accept}$. Therefore, M' accepts all inputs, and for each input all computations are accepting. Let $\Delta' = \Delta \cup \{@\}$, and let ω' be the weight function that is defined as follows

$$\omega'(t) = \begin{cases} \omega(t), & \text{for each transition } t \in \delta, \\ @, & \text{otherwise.} \end{cases}$$

As for each input word $w \in \Sigma^*$, the value of $f_\omega^M(w)$ is a finite set over Δ, it suffices to consider subsets of $\mathbb{P}_{\text{fin}}(\Delta^*)$. Then, we take

$$T' = \{\, V_1 \cup V_2 \mid V_1 \in \{L \cdot @ \mid L \in \mathbb{P}_{\text{fin}}(\Delta^*)\}, V_2 \in \mathbb{P}_{\text{fin}}(\Delta^*) \smallsetminus T \,\}.$$

For each input $w \in \Sigma^*$, if $w \in L_T(\mathcal{M})$, then it follows that $w \in L(M)$, and $f^M_\omega(w) \in T$. Further, by the above definition of the weight function ω', it is easily seen that $f^{M'}_{\omega'}(w)$ does not belong to the subset T'. Hence, $w \notin L_{T'}(\mathcal{M}')$.

If $w \notin L_T(\mathcal{M})$, then it follows that $w \notin L(M)$, or $w \in L(M)$, but $f^M_\omega(w) \notin T$. First, we consider the case that $w \notin L(M)$, that is, there is no accepting computation of M on the input w. It is rather clear that $w \in L(M')$, as M' accepts all inputs, and the weight of each computation of M' on the input w ends with the symbol @. By the definition of the subset T' it follows that $f^{M'}_{\omega'}(w) \in T'$, and thus $w \in L_{T'}(\mathcal{M}')$. Further, we consider the case that $w \in L(M)$, but $f^M_\omega(w) \notin T$. It is easily seen that $f^{M'}_{\omega'}(w) = V_1 \cup V_2$, where V_1 is a finite set of words that end with the symbol @, and $V_2 = f^M_\omega(w)$. Note that V_1 can also be empty. By the definition of the subset T' we obtain that the union $V_1 \cup V_2 \in T'$, and thus $w \in L_{T'}(\mathcal{M}')$.

From the arguments above, it follows that $L_T(\mathcal{M}) = \overline{L_{T'}(\mathcal{M}')}$. □

6.5 Concluding Remarks

We have introduced the notion of acceptance relative to a subset of a semiring using the weight function of a weighted restarting automaton to specify a language. This language is obtained from the language that is accepted by the underlying (unweighted) restarting automaton by restricting the weight associated to a given input word through an additional requirement. Here we have considered the case of the semirings over integers, such as the tropical semiring and the semiring of natural numbers with addition and multiplication, and the case of semirings of formal languages such as the regular languages $\mathsf{REG}(\Delta)$ over a finite alphabet Δ. First, we have seen that by using the semiring $\mathbb{Z}_\infty$, we can simulate the computations of a restarting automaton with auxiliary symbols by an automaton without auxiliary symbols, and by using the semiring $\mathbb{N}$, some #P-complete problem is solvable even by wR-automata. Further, for the notion of acceptance relative to a regular language, we have shown that our notion of relative acceptance just corresponds to that of non-forgetting restarting automata. In addition, we have also studied the closure properties and membership problems of these language classes. In particular, using a stronger acceptance notion, the classes of languages that are computed by general RWW- and RRWW-automata are closed under the operation of intersection. Finally, we summarize the inclusion relations between the language classes above in Figure 6.1.

However, many problems remain open. First, we do not yet have a characterization for the classes of languages that are accepted by the various types

$$\begin{tikzcd}
\hat{\mathcal{L}}(\mathsf{X}, \mathsf{REG}(\Delta), \mathsf{REG}(\Delta)) \arrow[r] \arrow[d] & \hat{\mathcal{L}}(\mathsf{X}, \mathsf{CFL}(\Delta), \mathsf{CFL}(\Delta)) \arrow[d] \\
\mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{reg(\Delta)}) \arrow[r] & \mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{cfl(\Delta)}) \\
\mathcal{L}(\mathsf{X}, \mathbb{N}_\infty, \mathbb{H}_{fin}^{N}) \arrow[r] \arrow[u] & \mathcal{L}(\mathsf{X}, \mathbb{Z}_\infty, \mathbb{H}_{fin}^{z}) \arrow[u]
\end{tikzcd}$$

Figure 6.1: Hierarchy of classes of languages that are accepted by weighted restarting automata relative to subsets of various semirings. An arrow denotes a (proper) inclusion.

of weighted restarting automata relative to subsets of the associated semiring. Further, the properness of most of the inclusions above is still unknown. Most importantly, we do yet not have an upper bound for the expressive power of these weighted restarting automata that we considered above, and it is still open whether the class $\mathcal{L}(\mathsf{X}, \mathbb{P}_{\text{fin}}(\Delta^*), \mathbb{H}_{fin}^{cfl(\Delta)})$ contains a language that is not context-sensitive. Notice that the languages $L_{3SAT,half}$ belongs to the language class DCSL, as a deterministic linearly space-bounded Turing machine can certainly accept them by just checking all assignments, respectively.

Chapter 7

Conclusion

In this work we have introduced weighted restarting automata and studied their computational power. We have seen that the class of functions that are computed by weighted restarting automata is quite rich, and each polynomial can be realized by a wRWW-automaton. In addition, it is shown that the function classes $\mathbb{F}(\mathsf{RWW}, \Sigma, S)$ and $\mathbb{F}(\mathsf{RRWW}, \Sigma, S)$ are closed under the operations of addition, scalar multiplication, and Cauchy product. Further, we have proved that the class of relations that are computed by weighted restarting automata of any type is incomparable to the relation classes lbPDR and PDR with respect to inclusion. Most importantly, we have obtained that the relations class $\mathcal{R}(\mathsf{mon\text{-}wRWW})$ is strictly included in the relation class $\mathcal{R}(\mathsf{mon\text{-}wRRWW})$, which is the first result that establishes a difference in the computational power between a model of the $\mathsf{mon\text{-}RWW}$-automaton and the corresponding model of the $\mathsf{mon\text{-}RRWW}$-automaton. Finally, the classes of language accepted by weighted restarting automata over various semirings as well as their closure properties and membership problems have been studied. We have seen that word-weighted restarting automata with the weak acceptance condition turn out to be equivalent to non-forgetting restarting automata. In particular, using a stronger acceptance condition, the classes of languages accepted by $\mathsf{w_{word}RWW}$- and $\mathsf{w_{word}RRWW}$-automata are closed under the operation of intersection. This is the first result that shows that a class of languages defined in terms of a quite general class of restarting automata is closed under the operation of intersection.

However, the following related questions remain open:

1. Are the classes $\mathbb{F}(\mathsf{X}, \Sigma, S)$ closed under addition and/or Cauchy product also for those types of restarting automata that cannot use auxiliary symbols?

2. For all types of restarting automata, it remains to characterize the classes of functions $\mathbb{F}(\mathsf{X}, \Sigma, S)$ and $\hat{\mathbb{F}}(\mathsf{X}, \Sigma, S)$ in a syntactic manner.

3. What is the inclusion relation between the classes $\mathcal{R}_{lb}((\mathsf{det\text{-}})\mathsf{mon\text{-}wRR}(1))$ and lbPDR?

4. It remains to characterize the classes of languages that are accepted by the various types of weighted restarting automata relative to subsets of the associated semiring.

5. For many types of restarting automata, the properness of the inclusions between the classes of languages that are accepted by weighted restarting automata over various semirings still remains proper.

6. Does the class $\mathcal{L}(\mathsf{X}, \mathbb{P}_{\mathrm{fin}}(\Delta^*), \mathbb{H}_{fin}^{cfl(\Delta)})$ contain a language that is not context-sensitive?

In order to explore these open problems, further study is called for.

Bibliography

[AAM07] Reza Ebrahimi Atani, Shahabaddin Ebrahimi Atani, and Sattar Mirzakuchaki. A novel public key crypto system based on semi-modules over quotient semi-rings. *IACR Cryptology ePrint Archive*, 2007:391, 2007.

[AU69] Alfred V. Aho and Jeffrey D. Ullman. Properties of Syntax Directed Translations. *J. Comput. Syst. Sci.*, 3(3):319–334, 1969.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[AVFY00] Serge Abiteboul, Victor Vianu, Bradley S. Fordham, and Yelena Yesha. Relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 61(2):236–269, 2000.

[Ber79] Jean Berstel. Transductions and context-free languages. *Ed. Teubner*, pages 1–278, 1979.

[BFGM05] Björn Borchardt, Zoltán Fülöp, Zsolt Gazdag, and Andreas Maletti. Bounds for tree automata with polynomial costs. *Journal of Automata, Languages and Combinatorics*, 10(2/3):107–157, 2005.

[BG70] Ronald V. Book and Sheila A. Greibach. Quasi-realtime languages. *Math. Syst. Theory*, 4(2):97–111, 1970.

[BO93] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems.* Texts and Monographs in Computer Science. Springer, New York, 1993.

[BO98] Gerhard Buntrock and Friedrich Otto. Growing context-sensitive languages and Church-Rosser languages. *Inf. Comput.*, 141(1):1–36, 1998.

[Bou06] Patricia Bouyer. Weighted timed automata: Model-checking and games. *Electr. Notes Theor. Comput. Sci.*, 158:3–17, 2006.

[CB14] Monika Cerinsek and Vladimir Batagelj. Semirings and matrix analysis of networks. In *Encyclopedia of Social Network Analysis and Mining*, pages 1681–1687. 2014.

[CC83] Christian Choffrut and Karel Culik II. Properties of Finite and Pushdown Transducers. *SIAM J. Comput.*, 12(2):300–315, 1983.

[CDH09] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Probabilistic weighted automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *Proceedings of the 20th International Conference on Concurrency Theory*, volume 5710 of *Lecture Notes in Computer Science*, pages 244–258, Heidelberg, 2009. Springer.

[CL04] Matteo Cavaliere and Peter Leupold. Evolution and observation: A non-standard way to accept formal languages. In Maurice Margenstern, editor, *Proceedings of the 4th International Conference on Machines, Computations, and Universality*, volume 3354 of *Lecture Notes in Computer Science*, pages 153–163, Heidelberg, 2004. Springer.

[CL06] Matteo Cavaliere and Peter Leupold. Observation of string-rewriting systems. *Fundam. Inform.*, 74(4):447–462, 2006.

[DHI$^+$93] Pál Dömösi, Sándor Horváth, Masami Ito, László Kászonyi, and Masashi Katsura. Formal languages consisting of primitive words. In Zoltán Ésik, editor, *Proceedings of Fundamentals of Computation Theory, 9th International Symposium*, volume 710 of *Lecture Notes in Computer Science*, pages 194–203, Heidelberg, 1993. Springer.

[DHV15] Manfred Droste, Doreen Heusel, and Heiko Vogler. Weighted unranked tree automata over tree valuation monoids and their characterization by weighted logics. In Andreas Maletti, editor, *Proceedings of the 6th International Conference on Algebraic Informatics*, volume 9270 of *Lecture Notes in Computer Science*, pages 90–102, Heidelberg, 2015. Springer.

[DK09] Manfred Droste and Werner Kuich. Semirings and formal power series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 3–28. Springer, Heidelberg, 2009.

[DK13] Manfred Droste and Werner Kuich. Weighted finite automata over hemirings. *Theor. Comput. Sci.*, 485:38–48, 2013.

[DK17] Manfred Droste and Werner Kuich. Weighted omega-restricted one counter automata. *CoRR*, abs/1701.08703, 2017.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, Heidelberg, 2009.

[DM11] Manfred Droste and Ingmar Meinecke. Weighted automata and regular expressions over valuation monoids. *Int. J. Found. Comput. Sci.*, 22(8):1829–1844, 2011.

[DW86] Elias Dahlhaus and Manfred K. Warmuth. Membership for growing context-sensitive grammars is polynomial. *J. Comput. System Sci.*, 33:456–472, 1986.

[Eil74] Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, Inc., Orlando, FL, USA, 1974.

[Eve63] Robert James Evey. *The Theory and Applications of Pushdown Store Machines*. PhD thesis, Harvard University, 1963.

[Gol99] Jonathan S. Golan. *Semirings and their applications*. Kluwer Academic Publishers, Dordrecht, 1999.

[GR66] Seymour Ginsburg and Gene F. Rose. Preservation of languages by transducers. *Information and Control*, 9(2):153–176, 1966.

[GR68] Seymour Ginsburg and Gene F. Rose. A note on preservation of languages by transducers. *Information and Control*, 12(5/6):549–552, 1968.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2 edition, 2001.

[HO12] Norbert Hundeshagen and Friedrich Otto. Characterizing the rational functions by restarting transducers. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Proceedings of the 6th International Conference on Language and Automata Theory and Applications*, volume 7183 of *Lecture Notes in Computer Science*, pages 325–336, Heidelberg, 2012. Springer.

[HO15] Norbert Hundeshagen and Friedrich Otto. Restarting transducers, regular languages, and rational relations. *Theory Comput. Syst.*, 57(1):195–225, 2015.

[HOV10] Norbert Hundeshagen, Friedrich Otto, and Marcel Vollweiler. Transductions computed by pc-systems of monotone deterministic restarting automata. In Michael Domaratzki and Kai Salomaa, editors, *Proceedings of the 15th International Conference on Implementation and Application of Automata*, volume 6482 of *Lecture Notes in Computer Science*, pages 163–172, Heidelberg, 2010. Springer.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Hun13] Norbert Hundeshagen. Relations and Transductions Realized by Restarting Automata. Doctoral thesis, Universität Kassel, 2013.

[ICJ14] Jelena Ignjatovic, Miroslav Ciric, and Zorana Jancic. Weighted finite automata with output. *CoRR*, abs/1410.2415, 2014.

[IK93] Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. *Computers & Graphics*, 17(3):305–313, 1993.

[JLNO04] Tomasz Jurdziński, Krzysztof Lorys, Gundula Niemann, and Friedrich Otto. Some results on RWW- and RRWW-automata and their relation to the class of growing context-sensitive languages. *J. Autom. Lang. Comb.*, 9(4):407–437, October 2004.

[JMPV95] Petr Jancar, Frantisek Mráz, Martin Plátek, and Jörg Vogel. Restarting Automata. In Horst Reichel, editor, *Proceedings of the 11th International Symposium on Fundamentals of Computation Theory*, volume 965 of *Lecture Notes in Computer Science*, pages 283–292, Heidelberg, 1995. Springer.

[JMPV97] Peter Jančar, František Mráz, Martin Plátek, and Jörg Vogel. On restarting automata with rewriting. In Gheorghe Păun and Arto Salomaa, editors, *New Trends in Formal Languages*, volume 1218 of *Lecture Notes in Computer Science*, pages 119–136. Springer, Heidelberg, 1997.

[JMPV98] Petr Jancar, Frantisek Mráz, Martin Plátek, and Jörg Vogel. Different types of monotonicity for restarting automata. In Vikraman Arvind and Ramaswamy Ramanujam, editors, *Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *Lecture Notes in Computer Science*, pages 343–354, Heidelberg, 1998. Springer.

[JMPV99] Petr Jancar, Frantisek Mráz, Martin Plátek, and Jörg Vogel. On Monotonic Automata with a Restart Operation. *J. Auto. Lang. Comb.*, 4(4):287–312, 1999.

[Kam09] Mark Kambites. Formal languages and groups as memory. *Communications in Algebra*, 37(1):193–208, 2009.

[Kir09] Daniel Kirsten. The support of a recognizable series over a zero-sum free, commutative semiring is recognizable. In Volker Diekert and Dirk Nowotka, editors, *Proceedings of the 13th International Conference on Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 326–333, Heidelberg, 2009. Springer.

[Kir11] Daniel Kirsten. The support of a recognizable series over a zero-sum free, commutative semiring is recognizable. *Acta Cybern.*, 20(2):211–221, 2011.

[KMO10a] Martin Kutrib, Hartmut Messerschmidt, and Friedrich Otto. On stateless deterministic restarting automata. *Acta Inf.*, 47(7-8):391–412, 2010.

[KMO10b] Martin Kutrib, Hartmut Messerschmidt, and Friedrich Otto. On Stateless Two-Pushdown Automata and Restarting Automata. *Int. J. Found. Comput. Sci.*, 21(5):781–798, 2010.

[KO12] Martin Kutrib and Friedrich Otto. On the descriptional complexity of the window size for deterministic restarting automata. In Nelma Moreira and Rogério Reis, editors, *Proceedings of the 17th International Conference on Implementation and Application of Automata*, volume 7381 of *Lecture Notes in Computer Science*, pages 253–264, Heidelberg, 2012. Springer.

[Lau88] C. Lautemann. One pushdown and a small tape. In Klaus W. Wagner, editor, *Dirk Siefkes zum 50. Geburtstag*, pages 42–47. Technische Universität Berlin and Universität Augsburg, 1988.

[LH15] Peter Leupold and Norbert Hundeshagen. A hierarchy of transducing observer systems. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Proceedings of the 9th International Conference on Language and Automata Theory and Applications*, volume 8977 of *Lecture Notes in Computer Science*, pages 727–738, Heidelberg, 2015. Springer.

[LPS07] Markéta Lopatková, Martin Plátek, and Petr Sgall. Towards a formal model for functional generative description:analysis by reduction and restarting automata. *The Prague Bulletin of Mathematical Linguistics*, (87):7–26, 2007.

[MNO88] Robert McNaughton, Paliath Narendran, and Friedrich Otto. Church-Rosser Thue systems and formal languages. *J. ACM*, 35(2):324–344, 1988.

[MO06] Hartmut Messerschmidt and Friedrich Otto. On nonforgetting restarting automata that are deterministic and/or monotone. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *Computer Science - Theory and Applications, First International Computer Science Symposium in Russia, Proceedings*, Lecture Notes in Computer Science, pages 247–258, Heidelberg, 2006. Springer.

[MO11] Hartmut Messerschmidt and Friedrich Otto. A hierarchy of monotone deterministic non-forgetting restarting automata. *Theory Comput. Syst.*, 48(2):343–373, 2011.

[Moh97] Mehryar Mohri. Finite-state transducers in language and speech processing. *Comput. Linguist.*, 23(2):269–311, June 1997.

[Mon02] Christopher J. Monico. *Semirings and Semigroup Actions in Public-Key Cryptography*. PhD thesis, Department of Mathematics, University of Notre Dame, 2002.

[MOP09] Frantisek Mráz, Friedrich Otto, and Martin Plátek. The degree of word-expansion of lexicalized RRWW-automata - A new measure for the degree of nondeterminism of (context-free) languages. *Theor. Comput. Sci.*, 410(37):3530–3538, 2009.

[MPJV97] Frantisek Mráz, Martin Plátek, Petr Jancar, and Jörg Vogel. Monotonic rewriting automata with a restart operation. In Frantisek Plasil and Keith G. Jeffery, editors, *Proceedings of the 24th Seminar on Current Trends in Theory and Practice of Informatics*, volume 1338 of *Lecture Notes in Computer Science*, pages 505–512, Heidelberg, 1997. Springer.

[MPR00] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. The design principles of a weighted finite-state transducer library. *Theor. Comput. Sci.*, 231(1):17–32, 2000.

[MPR02] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[Mrá01] Frantisek Mráz. Lookahead hierarchies of restarting automata. *Journal of Automata, Languages and Combinatorics*, 6(4):493–506, 2001.

[MS04] H. Messerschmidt and H. Stamer. Restart-Automaten mit mehreren Restart-Zuständen. In H. Bordihn, editor, *Workshop "Formale Methoden in der Linguistik" und 14. Theorietag "Automaten und Formale Sprachen"*, pages 111–116, Potsdam, 2004. Institut für Informatik, Universität Potsdam.

[Nar84] Paliath Narendran. *Church-Rosser and Related Thue Systems.* PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 1984.

[Nie03] Gundula Niemann. *Church-Rosser languages and related classes.* PhD thesis, University of Kassel, 2003.

[Niv09] Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 351–359, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[NO00a] Gundula Niemann and Friedrich Otto. Further results on restarting automata. In Masami Ito and Teruo Imaoka, editors, *Proceedings of the International Colloquium on Words, Languages & Combinatorics III*, pages 352–369. World Scientific, 2000.

[NO00b] Gundula Niemann and Friedrich Otto. Restarting automata, Church-Rosser languages, and representations of r.e. languages. In G. Rozenberg and W. Thomas, editors, *Developments in Language Theory - Foundations, Applications, and Perspectives, DLT 1999, Proc.*, pages 103–114. World Scientific, Singapore, 2000.

[NO01] Gundula Niemann and Friedrich Otto. On the power of RRWW-automata. In Masami Ito, Gheorghe Paun, and Sheng Yu, editors, *Words, Semigroups, and Transductions - Festschrift in Honor of Gabriel Thierrin*, pages 341–355. World Scientific, 2001.

[NO05] Gundula Niemann and Friedrich Otto. The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages. *Inf. Comput.*, 197(1-2):1–21, 2005.

[NO12] Benedek Nagy and Friedrich Otto. On CD-systems of stateless deterministic R-automata with window size one. *J. Comput. Syst. Sci.*, 78(3):780–806, 2012.

[Ogd68] William F. Ogden. A helpful result for proving inherent ambiguity. *Mathematical Systems Theory*, 2(3):191–194, 1968.

[Ott06] Friedrich Otto. Restarting Automata. In Zoltán Ésik, Carlos Martín-Vide, and Victor Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 269–303. Springer, Heidelberg, 2006.

[Ott10] F. Otto. On proper languages and transformations of lexicalized types of automata. In Masami Ito, Yuji Kobayashi, and Kunitaka Shoji, editors, *Automata, Formal Languages and Algebraic Systems - Proceedings of AFLAS 2008*. World Scientific, 2010.

[OW16] Friedrich Otto and Qichao Wang. Weighted Restarting Automata. *Soft Computing*, 2016. DOI: 10.1007/s00500-016-2164-4. The results of this paper have been announced at WATA 2014 in Leipzig.

[PB16] Selena Praprotnik and Vladimir Batagelj. Semirings for temporal network analysis. *CoRR*, abs/1603.08261, 2016.

[Sb61] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

[Spi00] Marc Spielmann. Verification of relational transducers for electronic commerce. In Victor Vianu and Georg Gottlob, editors, *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 92–103. ACM, 2000.

[SS78] Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, New York, 1978.

[Str00] Markéta Straňáková. Selected types of pg-ambiguity: Processing based on analysis by reduction. In Petr Sojka, Ivan Kopeček, and Karel Pala, editors, *Text, Speech and Dialogue*, volume 1902 of *Lecture Notes in Computer Science*, pages 139–144. Springer, Heidelberg, 2000.

[Tes59] Lucien Tesnière. *Éléments de syntaxe structurale.* Klincksieck, 1959.

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[VO12] Marcel Vollweiler and Friedrich Otto. Systems of parallel communicating restarting automata. In Rudolf Freund, Markus Holzer, Bianca Truthe, and Ulrich Ultes-Nitsche, editors, *Proceedings of the Fourth Workshop on Non-Classical Models for Automata and Applications*, volume 290 of *books@ocg.at*, pages 197–212. Österreichische Computer Gesellschaft, 2012.

[Wan17] Qichao Wang. On the expressive power of weighted restarting automata. In Rudolf Freund, Frantisek Mráz, and Daniel Prusa, editors, *Proceedings of the Ninth Workshop on Non-Classical Models of Automata and Applications*, volume 329 of *books@ocg.at*, pages 227–241. Österreichische Computer Gesellschaft, 2017.

[WHO15] Qichao Wang, Norbert Hundeshagen, and Friedrich Otto. Weighted restarting automata and pushdown relations. In Andreas Maletti, editor, *Proceedings of the 6th International Conference on Algebraic Informatics*, volume 9270 of *Lecture Notes in Computer Science*, pages 196–207. Springer, Heidelberg, 2015.

[WO16a] Qichao Wang and Friedrich Otto. Weighted restarting automata and pushdown relations. *Theor. Comput. Sci.*, 635:1–15, 2016.

[WO16b] Qichao Wang and Friedrich Otto. Weighted restarting automata as language acceptors. In Yo-Sub Han and Kai Salomaa, editors, *the 21st International Conference on Implementation and Application of Automata, Proceedings*, volume 9705 of *Lecture Notes in Computer Science*, pages 298–309, Switzerland, 2016. Springer.

[Yu89] Sheng Yu. A pumping lemma for deterministic context-free languages. *Inf. Process. Lett.*, 31(1):47–51, April 1989.

List of Abbreviations

3-SAT	3-statisfiability problem
artDPDF	class of linearly bounded functions the are computed by well-behaved deterministic pushdown transducers that must pop from their pushdowns during λ-steps
artPDR	class of almost-realtime pushdown relations
CFL	class of context-free languages
CRL	class of Church-Rosser languages
CSL	class of context-sensitive languages
DCFL	class of deterministic context-free languages
DCSL	class of deterministic context-sensitive languages
det	deterministic
DFA	deterministic finite state automaton
DPDA	deterministic pushdown automaton
DPDF	class of functions that are computed by well-behaved deterministic pushdown transducers
DPDR	class of deterministic pushdown relations
DPDT	deterministic pushdown transducer
DTPDA	deterministic two-pushdown automaton
FST	finite state transducer
GCSL	class of growing context-sensitive languages
lb	linearly bounded
lbDPDF	class of linearly bounded functions the are computed by well-behaved deterministic pushdown transducers
lbPDR	class of linearly bounded pushdown relations
mon	monotone
MVR	move-right
nf	non-forgetting
NFA	nondeterministic finite state automaton
NP	nondeterministic polynomial time
P	deterministic polynomial time
PC	parallel communicating

PDA	nondeterministic pushdown automaton
PDR	class of pushdown relations
PDT	nondeterministic pushdown transducer
RAT	class of rational relations
RE	class of recursively enumerable languages
REG	class of regular languages
rtDPDF	class of linearly bounded functions the are computed by well-behaved deterministic pushdown transducers without λ-steps
rtPDR	class of realtime pushdown relations
stl	stateless
Td	transducer
TPDA	nondeterministic two-pushdown automaton
w	weighted
wbDPDT	well-behaved deterministic pushdown transducer
$\mathsf{w}_{\mathsf{word}}$	word-weighted
$\mathsf{w}_{\mathsf{FIN}}$	finitely weighted

List of Symbols

$\mathbb{N}$	set of all natural numbers
$\mathbb{N}_+$	set of all positive natural numbers
$\mathbb{N}_\infty$	$\mathbb{N} \cup \{\infty\}$
$\overline{\mathbb{N}}$	$\mathbb{N} \cup \{-\infty\}$
$\mathbb{Q}$	set of all rational numbers
$\mathbb{R}$	set of all real numbers
R	type of restarting automaton that can only remove some symbols from its read/write window in a rewrite step, and that must restart immediately after performing a rewrite step
RR	type of restarting automaton that can only remove some symbols from its read/write window in a rewrite step
RRW	type of restarting automaton that cannot use auxiliary symbols in a rewrite step
RRWW	general type of restarting automaton
RW	type of restarting automaton that cannot use auxiliary symbols in a rewrite step, and that must restart immediately after performing a rewrite step
RWW	type of restarting automaton that must restart immediately after performing a rewrite step
$\mathbb{Z}$	set of all integers
$\mathbb{Z}_\infty$	$\mathbb{Z} \cup \{\infty\}$
$\overline{\mathbb{Z}}$	$\mathbb{Z} \cup \{-\infty\}$

ISBN 978-3-7376-0446-8

9 783737 604468 >