Weighted Restarting Automata and Pushdown Relations

Qichao Wang, Norbert Hundeshagen, and Friedrich Otto(✉)

Fachbereich Elektrotechnik/Informatik, Universität Kassel, 34109 Kassel, Germany
{wang,hundeshagen,otto}@theory.informatik.uni-kassel.de

Abstract. Weighted restarting automata have been introduced to study quantitative aspects of computations of restarting automata. Here we study the special case of assigning words as weights from the semiring of formal languages over a given (output) alphabet, in this way generalizing the restarting transducers introduced by Hundeshagen (2013). We obtain several new classes of word relations in terms of restarting automata, which we relate to various types of pushdown relations.

Keywords: Weighted restarting automaton · Restarting transducer · Pushdown relation · Quasi-realtime pushdown relation

1 Introduction

Analysis by reduction is a linguistic technique that is used to check the correctness of sentences of natural languages through sequences of local simplifications. The *restarting automaton* was invented as a formal model for the analysis by reduction [7]. In order to study quantitative aspects of computations of restarting automata, *weighted restarting automata* were introduced in [10]. These automata are obtained by assigning an element of a given semiring S as a weight to each transition of a restarting automaton. Then the product (in S) of the weights of all transitions that are used in a computation yields a weight for that computation, and by forming the sum over the weights of all accepting computations for a given input $w \in \Sigma^*$, a value from S is assigned to w. Thus, a partial function $f : \Sigma^* \dashrightarrow S$ is obtained. Here we consider the special case that S is the semiring of formal languages over some finite (output) alphabet Δ. Then f is a transformation from Σ^* into the languages over Δ. Thus, we obtain a generalization of the notion of a *restarting transducer* as introduced in [6].

It is well known (see, e.g., [8]) that the class of languages that are accepted by monotone RWW- and RRWW-automata (see Section 2 for the definitions) coincides with the class of context-free languages. Accordingly, we are interested in the classes of transformations that are computed by various types of weighted restarting automata that are *monotone*. In this paper we compare some of these classes to each other and we relate them to the class of *pushdown relations* and some of its subclasses. In particular, we prove that monotone weighted RRWW-automata compute strictly more transformations than

A. Maletti (Ed.): CAI 2015, LNCS 9270, pp. 196–207, 2015.
DOI: 10.1007/978-3-319-23021-4_18

monotone weighted RWW-automata. The latter in turn compute a class that properly includes the quasi-realtime pushdown relations, which we will show to coincide with the transformations that are computed by monotone RWW- and RRWW-transducers.

This paper is structured as follows. In Section 2 we recall some basic notions concerning weighted restarting automata, and in Section 3 we look at the pushdown relations and some of their subclasses. Then, in Section 4 we study the classes of transformations that are computed by (monotone) restarting transducers, and in Section 5 we investigate the computational power of weighted RWW- and RRWW-automata that are monotone. The paper closes with a short summary and some problems for future work.

2 Weighted Restarting Automata

We assume that the reader is familiar with the standard notions and concepts of theoretical computer science, such as monoids, finite automata, and semirings. Throughout the paper we will use $|w|$ to denote the length of a word w and λ to denote the empty word. Further, $\mathbb{P}(X)$ denotes the power set of a set X, and $\mathbb{P}_{\text{fin}}(X)$ denotes the set of all finite subsets of X.

A *restarting automaton* (or RRWW-automaton for short) is a nondeterministic machine with a finite-state control, a flexible tape with endmarkers, and a read/write window of a fixed finite size. Formally, it is described by 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 $¢, \$ \notin \Gamma$ are used as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and δ is the (partial) *transition relation* that associates finite sets of transition steps to pairs of the form (q, w), where q is a state and w is a possible content of the read/write window. There are four types of transition steps. A *move-right step* (MVR) causes M to shift its read/write window one position to the right and to change the state. A *rewrite step* causes M to replace the content w of the read/write window by a shorter string v, thereby reducing the length of the tape, and to change the state. Further, the read/write window is placed immediately to the right of the string v. However, occurrences of the delimiters ¢ and \$ can neither be deleted nor newly created by a rewrite step. A *restart step* causes M to place its read/write window over the left end of the tape, so that the first symbol it sees is the left sentinel ¢, and to reenter the initial state q_0, and, finally, an *accept step* causes M to halt and accept.

If $\delta(q, w)$ is undefined for some pair (q, w), then M necessarily halts in a corresponding situation, and we say that M *rejects*. Finally, if each rewrite step is combined with a restart step into a joint rewrite/restart operation, then M is called an *RWW-automaton*.

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 is the current state, and $\alpha\beta$ is the current content of the tape, where it is understood that the window contains

the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 ¢ w \$$. If $w \in \Sigma^*$, then $q_0 ¢ w \$$ is an *initial configuration*.

We observe that any computation of 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 single rewrite operation until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a *tail*. It is required that in each cycle M performs *exactly one* rewrite step. A word $w \in \Sigma^*$ is accepted by M, if there is an accepting computation which starts from the initial configuration $q_0 ¢ w \$$. By $L(M)$ we denote the language consisting of all (input) words that are accepted by M.

Next we come to the notion of *monotonicity*. Let $C := \alpha q \beta$ be a *rewrite configuration* of an RRWW-automaton M, that is, a configuration in which a rewrite step is to 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 $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$, 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 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.

For studying quantitative aspects of computations of restarting automata, the weighted restarting automaton has been introduced in [10]. A *weighted restarting automaton* of type X, a wX-automaton for short, is a pair (M, ω), where M is a restarting automaton of type X, and ω is a *weight function* from the transitions of M into a semiring S. This *weight function* assigns an element $\omega(t) \in S$ as a weight to each transition t of M. 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.

Let $\mathcal{M} = (M, \omega)$ be a weighted restarting automaton, where $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$. Let $\mathrm{AC}_M(w) = \{A_1, A_2, \cdots, A_m\}$ 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 $t_{i,1}, t_{i,2}, \cdots, t_{i,n_i}$ of M. Then the weight of a transition $t_{i,j}$ $(1 \leq j \leq n_i)$ 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, $f_\omega^M(w) = \hat{L}_1 \cup \hat{L}_2 \cup \cdots \cup \hat{L}_m \in \mathbb{P}(\Delta^*)$ 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

$Rel(\mathcal{M}) = \{ (u, v) \mid u \in L(M), v \in f_\omega^M(u) \} \subseteq \Sigma^* \times \Delta^*$. By $\mathcal{R}(\mathsf{wX})$ we denote the class of relations that are computed by weighted restarting automata of type wX.

As the weight of a transition of M can be any language over Δ, the general model of weighted restarting automata is quite powerful. Therefore, we introduce some more restricted types of weighted restarting automata.

Definition 1. *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 $\mathcal{R}(\mathsf{w_{word}X}) = \mathcal{R}(\mathsf{w_{FIN}X}) \subsetneq \mathcal{R}(\mathsf{wX})$ for each type $\mathsf{X} \in \{\mathsf{mon\text{-}RWW}, \mathsf{mon\text{-}RRWW}, \mathsf{RWW}, \mathsf{RRWW}\}$. We close this section with a simple example of a relation that is computed by a weighted restarting automaton.

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

$$t_1 : (q_0, ¢a) \to (q_0, \mathsf{MVR}),\ t_3 : (q_0, aa) \to (q_0, \mathsf{MVR}),$$
$$t_2 : (q_0, ¢\$) \to \mathsf{Accept}, \qquad t_4 : (q_0, a\$) \to \$.$$

Here t_4 is the only (combined) rewrite/restart operation of M_1. It is easily seen that $L(M_1) = \{ a^n \mid n \geq 0 \}$.

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 MVR transitions t_1 and t_3, and that assigns the set $\{\lambda\}$ to all other transitions, and let $\mathcal{M}_1 = (M_1, \omega_1)$. It follows easily that

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

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

Finally, we recall the notion of *restarting transducer* from [3]. 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. Hence, restarting transducers are a special type of word-weighted restarting automata. By $\mathcal{R}(\mathsf{X\text{-}Td})$ we denote the class of relations that are computed by restarting transducers of type X.

3 Pushdown Relations

A *pushdown transducer* (PDT for short) is defined by an 8-tuple $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, E)$, 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 $E \subset Q \times (\Sigma \cup \{\lambda\}) \times X \times Q \times X^* \times \Delta^*$ is a finite transition relation that produces a (possible empty) output word in each step (see, e.g., [2]). The output produced during a computation is then simply the concatenation of all outputs produced during that computation.

A configuration of 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, and $v \in \Delta^*$ is the output produced so far. The relation $Rel(T)$ computed by T is defined as

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

A relation $R \subseteq \Sigma^* \times \Delta^*$ is called a *pushdown relation* if $R = Rel(T)$ holds for some PDT T. By PDR we denote the class of all 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 $(u, v) \in R$. 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 $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, E)$ that does not perform any λ-steps, that is, its set of transitions E satisfies the condition $E \subset Q \times \Sigma \times X \times Q \times X^* \times \Delta^*$. By rtPDR we denote the class of all realtime pushdown relations.

Finally, a pushdown relation R is called *quasi-realtime* if it is computed by a PDT $T = (Q, \Sigma, \Delta, X, q_0, Z_0, F, E)$ for which each λ-step pops a symbol from the pushdown, that is, if $(q, \lambda, x, q', x', v) \in E$, then $x' = \lambda$. By qrtPDR we denote the class of all quasi-realtime pushdown relations.

Proposition 3. $\mathsf{rtPDR} \subsetneq \mathsf{qrtPDR} \subsetneq \mathsf{lbPDR} \subsetneq \mathsf{PDR}$.

Proof. The first and the third inclusions are obvious. Concerning the second inclusion, assume that R is computed by a quasi-realtime PDT T. Let $(u, v) \in R$. On reading an input symbol, T can push a string of length c (for some constant $c \geq 1$) onto its pushdown, and so altogether at most $c \cdot |u|$ symbols are pushed. Hence, T can execute at most $c \cdot |u|$ λ-transitions, which means that, on input u, T executes at most $(c+1) \cdot |u|$ steps. Thus, the output v produced during this computation satisfies the inequality $|v| \leq d \cdot (c+1) \cdot |u|$, where d is the maximal length of any output string produced by T in a single step.

It remains to prove that all the inclusions above are proper. The transduction $R_{uu^R} = \{\, (u, uu^R) \mid u \in \{a, b\}^* \,\}$ is quasi-realtime: a PDT T can output its input u letter by letter, also pushing each letter onto the pushdown. At the end of the input, which T can guess, it empties its pushdown letter by letter, producing the output u^R. On the other hand, this transduction is not realtime, as in a realtime pushdown relation the final output syllable is produced when the last input symbol is being read, which is not possible for the relation R_{uu^R}.

The relation $R_{a^m b^m c^n} = \{\, (a^m b^m c^n, c^n a^m b^m) \mid m, n \geq 1 \,\}$ is a linearly bounded pushdown relation. A PDT T can first guess c^n, outputting this factor and pushing it onto the pushdown. Then it compares the syllables a^m and b^m,

producing the output $a^m b^m$. Finally, it checks the syllable c^n against the c-syllable on its pushdown. However, $R_{a^m b^m c^n}$ is not quasi-realtime. The output syllable c^n must be produced first, but the pushdown must be used for comparing a^m to b^m, which are the first two syllables of the input. In addition, when this comparison is made, then the output $a^m b^m$ must be produced. Hence, the output syllable c^n must already be produced before the input syllable c^n is being read, that is, the output c^n is produced through λ-transitions that do not pop from the pushdown.

Finally, the relation $R_+ = \{ (a^m, b^n a^m b^n) \mid m, n \geq 1 \}$ is a pushdown relation. Obviously, it is not linearly bounded. □

The class of pushdown relations can be characterized in terms of context-free languages and morphisms. For that we recall the following concept from [1].

Definition 4. *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 [1] it was shown that the pushdown relations are characterized by the context-free languages. For the case that Σ and Δ are disjoint, an even stronger result was shown that assumes that $\Gamma = \Sigma \cup \Delta$ and that h_1 (h_2) is the projection from Γ^* onto Σ^* (Δ^*). In terms of [1] this is expressed by saying that the pushdown relations are *strongly characterized* by the context-free languages. In the following we extend this result to lbPDR.

Lemma 5. *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 4 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 $h_1'(x) = \begin{cases} h_1(x), & x \in \Gamma \\ x, & x \in \Gamma' \end{cases}$ and $h_2'(x') = \begin{cases} h_2(x), & x' \in \Gamma' \\ x', & x' \in \Sigma \end{cases}$.

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. □

4 Pushdown Relations and Restarting Transducers

Every relation that is computed by a restarting transducer is *linearly bounded* in the sense of the class lbPDR, 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. It follows that restarting transducers cannot compute all pushdown relations. Naturally, the question arises of whether they can at least compute all linearly bounded pushdown relations. In [3] it was claimed that monotone RWW- and RRWW-transducers do exactly compute these relations, but actually, only a weaker result was proven there. Here we show that these transducers actually characterize the class qrtPDR. By Proposition 3 this means that they cannot realize all relations from the class lbPDR.

Theorem 6. $\mathcal{R}(\text{mon-RWW-Td}) = \mathcal{R}(\text{mon-RRWW-Td}) = \text{qrtPDR}$.

To prove this result we present two lemmas.

Lemma 7. $\text{qrtPDR} \subseteq \mathcal{R}(\text{mon-RWW-Td})$.

Proof. Let $R \subseteq \Sigma^* \times \Delta^*$ be the relation that is computed by the quasi-realtime PDT $T = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$. We now simulate T by a mon-RWW-Td using a construction from [9].

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

In each cycle M simulates two steps of T. Assume that an accepting computation of T on input $w = a_0 a_1 \cdots a_n$ begins by first applying the transition $(q_1, B_1 \cdots B_{m_1} C_1, v_1) \in \delta(q_0, a_0, Z_0)$ and then the transition $(q_2, B_{m_1+1} \cdots B_{m_1+m_2} C_2, v_2) \in \delta(q_1, a_1, C_1)$. As $m_1 < l$ and $m_2 < l$, $|B_1 \cdots B_{m_1+m_2} C_2| < 2l$ holds. Accordingly, starting with the input configuration corresponding to input w, M can execute the rewrite step $¢a_0a_1a_2 \to ¢(xC_2)a_2$, where $x := B_1 \cdots B_{m_1+m_2}$, producing the output $v_1 v_2$.

Assume that by executing the next two steps, the PDT T reaches the configuration $(q_4, a_4 \cdots a_n, B_1 \cdots B_{m_1+m_2-1} x_1)$, that is, the factor a_2a_3 is read from the input tape, the internal state changes to q_4, the two topmost symbols $B_{m_1+m_2} C_2$ on the pushdown are rewritten into the string $x_1 \in \Gamma^*$, and the output $v_3 v_4$ is produced. If $m_1 + m_2 - 1 + |x_1| \leq 2l$, then M rewrites $(xC_2)a_2a_3a_4$ into $(x')a_4$, where $x' = B_1 \cdots B_{m_1+m_2-1} x_1$, and if $m_1 + m_2 - 1 + |x_1| > 2l$, then M rewrites $(xC_2)a_2a_3a_4$ into $(x')(x'')a_4$, where $x'x'' = B_1 \cdots B_{m_1+m_2-1} x_1$ and $|x'| = 2l$.

In addition, if T executes a λ-step, then it changes its state, pops a symbol from the pushdown, and produces an output syllable. In order for M to simulate this in a length-reducing fashion, we must combine up to $2l$ λ-steps of T (or several λ-steps together with the next non-λ-step) into a single simulation step of M. This is rather technical, but nevertheless fairly standard.

Continuing in this way it follows that the tape content of M is always of the form $\alpha(u)a_j \cdots a_n$, where $(u) \in \Gamma_1'$, and $\alpha \in \Gamma_2'^*$. Here αu encodes the current content of the pushdown of T, and $a_j \cdots a_n$ is the suffix of the input that T still has to read. As long as $j < n-1$, M can simulate the next two steps of T by rewriting the four symbols $(x_i)(x_{i+1})a_j a_{j+1}$ either into $(x_i)(x_{i+1})(x_{i+2})$, into $(x_i)(x_{i+2})$, or into (x_{i+2}), depending on the way in which the contents of the pushdown of T is modified by these 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 8. $\mathcal{R}(\mathsf{mon\text{-}RRWW\text{-}Td}) \subseteq \mathsf{qrtPDR}$.

Proof. Let M be a mon-RRWW-Td. Using the simulation technique from [8] it can be shown that M can be simulated by a PDT T. Let $\mathrm{¢}uqvw\$$ be a rewrite configuration within 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 $q_0\mathrm{¢}uv'w\$$, 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. As an RRWW-transducer, M moves to the right after performing the above rewrite step, and (without loss of generality) it only restarts and produces its output at the right end of the tape, provided the state reached leads to a restart operation. 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 further letters from 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 the corresponding output strings.

In fact, as M is monotone, it can be checked quite easily that T is quasi-realtime, that is, whenever T executes a λ-transition, then it pops a symbol 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 $\mathcal{R}(\mathsf{mon\text{-}RWW\text{-}Td}) \subseteq \mathcal{R}(\mathsf{mon\text{-}RRWW\text{-}Td})$, Lemmas 7 and 8 imply the characterization in Theorem 6. Next it can be shown that all linearly bounded pushdown relations are accepted by (non-monotone) RRWW-transducers.

Theorem 9. $\mathsf{lbPDR} \subseteq \mathcal{R}(\mathsf{RRWW\text{-}Td})$.

Proof. Let $R \subseteq \Sigma^* \times \Delta^*$ be a linearly bounded pushdown relation. W.l.o.g. we assume that Σ and Δ are disjoint. By Lemma 5, R is strongly characterized by a context-free language $L \subseteq (\Sigma \cup \Delta)^*$ and the two projections $h_i : (\Sigma \cup \Delta)^* \to \Sigma^*$

and $h_o : (\Sigma \cup \Delta)^* \to \Delta^*$. Furthermore, there is a constant k such that, for all $(u, v) \in R$, there exists a word $w \in L$ such that $|w| \leq k \cdot |u|$ and $(u, v) = (h_i(w), h_o(w))$. Let M be a PDA for L. Now an RRWW-Td T for R can be constructed that proceeds in two steps. For a given pair $(u, v) \in R$,

1. T guesses a characterizing word w of (u, v) and produces the output $h_o(w)$,
2. T verifies that $w \in L$ by simulating the PDA M on w.

The main problem in constructing T is the fact that we have to ensure that these steps are realized in a length-reducing manner. □

Actually, the inclusion in Theorem 9 has already been stated in [4] and its journal version [5] by relating restarting transducers to transducing observer systems. The proof above can easily be converted to the latter, in this way correcting the proof given in these papers, which only proves a weaker result.

5 Relations Computed by Monotone Weighted RWW- and RRWW-Automata

In the previous section we have shown that monotone RWW- and RRWW-transducers compute the relations in qrtPDR. Are (word-weighted) RWW- and RRWW-automata that are monotone more expressive?

We begin this investigation by studying the relation between the classes $\mathcal{R}(\mathsf{mon\text{-}wRWW})$ and $\mathcal{R}(\mathsf{mon\text{-}wRRWW})$. Let $\tau_1 \subseteq \{a, b, c\}^* \times \{d, e\}^*$ be the relation

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

Lemma 10. $\tau_1 \notin \mathcal{R}(\mathsf{mon\text{-}wRWW})$.

Proof. Assume that $\tau_1 \in \mathcal{R}(\mathsf{mon\text{-}wRWW})$, that is, there exists a weighted mon-RWW-automaton M and a weight function ω' that maps the transitions of M into subsets of $\{d, e\}^*$ such that $\tau_1 = Rel((M, \omega'))$. As τ_1 is actually a (partial) function, we see that ω' can be replaced by a weight function ω that maps each transition of M into a singleton, which means that $\mathcal{M} = (M, \omega)$ is a word-weighted mon-RWW-automaton. Interpreting the weight $\omega(t)$ of a transition as output, we see that, for an input of the form $a^k b^k c^m$, $\mathcal{M}$ first outputs the symbol d m-times, which is the number of c-symbols in the input, and then it outputs k e-symbols, which is the number of a- and b-symbols in the input.

As the language $L = \{\, a^k b^k c^m \mid k, m \geq 1 \,\}$ is not regular, M needs to execute rewrite steps in all its accepting computations on input $a^k b^k c^m$, if k is sufficiently large. At what position can the first of these rewrite steps be applied?

(1) Assume that the first rewrite step is applied within the suffix c^m. While processing this suffix, M can easily produce the output d^m. Then M must compare the prefix a^k to the infix b^k, and while doing so it should produce the output e^k. 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. Accordingly, the infix b^k must be reduced by rewrites to a word that fits into the window of M, which means that M cannot distinguish between b^k and b^{k+r} for some positive integer r. Thus, together with $a^k b^k c^m$, M would also accept the word $a^k b^{k+r} c^m$, contradicting our assumption on M.

(2) From the arguments above, it follows that the first rewrite step must be executed within the prefix a^k or at the border between the prefix a^k and the infix b^k. This means that M must first compare the syllables a^k and b^k, and since by this process the information on the exponent k is being destroyed, it must produce the output e^k during this process. However, as the output syllable e^k is preceded by the prefix d^m, M must already output the syllable d^m before it starts to output e-symbols. As shown in [10], the length of any computation of M on an input of length n is at most $\frac{1}{2}(n+2)(n+3)-1$. This means that during the processing of the prefix $a^k b^k$, M can perform at most $\frac{1}{2}(2k+2)(2k+3)-1$ steps. Choose $l \geq 1$ to be a constant such that $|\omega(t)| \leq l$ for all transitions t of M, and choose m such that $m > (\frac{1}{2}(2k+2)(2k+3)-1) \cdot l$. Then $\mathcal{M}$ is not able to produce m d-symbols, while it is processing the prefix $a^k b^k$. Thus, it either stops producing d-symbols before it has erased all information on the number k, which means that not enough d-symbols are produced, or it keeps on producing d-symbols while erasing all information on k. In the latter case it will then not be able to produce the correct number of e-symbols. □

Obviously, $\mathcal{R}(\mathsf{mon\text{-}wRWW})$ is contained in $\mathcal{R}(\mathsf{mon\text{-}wRRWW})$. We now prove that this inclusion is proper.

Theorem 11. $\mathcal{R}(\mathsf{mon\text{-}xRWW}) \subsetneq \mathcal{R}(\mathsf{mon\text{-}xRRWW})$ *for all* $\mathsf{x} \in \{\mathsf{w}, \mathsf{w}_{\mathsf{FIN}}, \mathsf{w}_{\mathsf{word}}\}$.

Proof. By Lemma 10, $\tau_1 \notin \mathcal{R}(\mathsf{mon\text{-}wRWW})$. On the other hand, it is easy to construct a monotone word-weighted RRWW-automaton $\mathcal{M} = (M, \omega)$ such that $Rel(\mathcal{M}) = \tau_1$. This automaton $\mathcal{M}$ proceeds as follows. Let $w = a^k b^k c^m$ be given as input. In the first cycle, $\mathcal{M}$ places a marking on the prefix of w by encoding the first two symbols into a combined (new) symbol, and then it moves to the suffix c^m of w. While scanning this suffix, it outputs a d-symbol for each c-symbol that it encounters, and at the right delimiter, it restarts. In the subsequent cycles, on seeing the marking at the left end of the tape, $\mathcal{M}$ realizes that it has already produced the d-symbols. Hence, it now moves to the boundary between the prefix a^k and the infix b^k to compare them. In each subsequent rewrite step, it removes a single a-symbol and a single b-symbol, producing a single e-symbol as output (via ω). It follows that $Rel(\mathcal{M}) = \tau_1$, which completes the proof. □

We remark that Theorem 11 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 $\tau_1 = \{ (a^k b^k c^m, d^m e^k) \mid k, m \geq 1 \}$ considered above is a linearly bounded pushdown relation that is not computed by any monotone weighted RWW-automaton. On the other hand, the relation considered in Example 2 is computed by a monotone word-weighted RWW-automaton. Its domain a^* is context-free, while its range $\{ c^{\frac{1}{2}(n+1)n} \mid n \geq 0 \}$ is not. Hence, this relation is not a pushdown relation. Thus, we have the following incomparability result.

Theorem 12
For each prefix $\mathsf{x} \in \{\mathsf{w}, \mathsf{w}_{\mathsf{FIN}}, \mathsf{w}_{\mathsf{word}}\}$*, the class of relations* $\mathcal{R}(\mathsf{mon\text{-}xRWW})$ *is incomparable to the classes* lbPDR *and* PDR *with respect to inclusion.*

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

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

Lemma 13. $\tau_2 \notin \mathcal{R}(\mathsf{mon\text{-}wRRWW})$.

Proof. The relation τ_2 is a partial function. Thus, if τ_2 is computed by a monotone wRRWW-automaton, then it is also computed by a monotone word-weighted RRWW-automaton $\mathcal{M} = (M, \omega)$. Interpreting the weight $\omega(t) \in \{d, e\}^*$ of a transition t as output, $\mathcal{M}$ first outputs the syllable d^m, then e^k, then d^m again, and finally e^l given the word $a^k b^k c^{m+l} a^l$ as input.

As $\mathcal{M}$ is monotone, we see that $\mathcal{M}$ must first compare the prefix a^k to the infix b^k (see the proof of Lemma 10). Since the information about the exponent k is lost during this process, $\mathcal{M}$ must produce the output syllable e^k during this process. Hence, 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^k of the input while it produces the output d^m.

The exact value of m is unknown, that is, while moving right across the input syllable c^{m+l}, $\mathcal{M}$ must guess it. After comparing the numbers of a- and b-symbols and outputting correspondingly many e-symbols, $\mathcal{M}$ must again produce m d-symbols, that is, it must somehow remember this number. However, as $\mathcal{M}$ must not perform any rewrite steps on the suffix $c^{m+l} a^l$ before a^k has been compared to b^k, it must encode the number m within the prefix a^k. However, if m is sufficiently large, then this is not possible. Hence, it follows that τ_2 cannot be computed by any weighted RRWW-automaton that is monotone. □

Clearly τ_2 is a linearly bounded pushdown relation, too. Hence, from Example 2 and Lemma 13 the following incomparability result follows.

Theorem 14
For each prefix $\mathsf{x} \in \{\mathsf{w}, \mathsf{w}_{\mathsf{FIN}}, \mathsf{w}_{\mathsf{word}}\}$*, the class of relations* $\mathcal{R}(\mathsf{mon\text{-}xRRWW})$ *is incomparable to the classes* lbPDR *and* PDR *with respect to inclusion.*

6 Conclusion

We have studied the classes of (binary) relations that are computed by weighted RWW- and RRWW-automata that are monotone, relating them to the classes of relations that are computed by monotone RWW- and RRWW-transducers and to some classes of pushdown relations. The inclusion results obtained are summarized in the diagram in Figure 1. In particular, we have shown that the monotone RWW- and RRWW-transducers characterize the class of quasi-realtime

pushdown relations, and we have seen that monotone (word-) weighted RWW-automata are strictly weaker in computational power than monotone (word-) weighted RRWW-automata. The latter is the first known case where it 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.

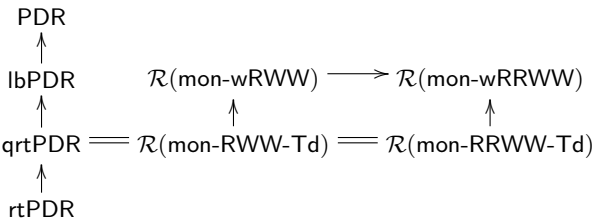

Fig. 1. Hierarchy of classes of (binary) relations that are computed by monotone R(R)WW-transducers and (word-)weighted R(R)WW-automata. An arrow denotes a proper inclusion, and classes that are not connected are incomparable with respect to inclusion.

References

1. Aho, A.V., Ullman, J.D.: The Theory of Parsing, Translation, and Compiling. Prentice-Hall Inc., Upper Saddle River (1972)
2. Choffrut, C., Culik II, K.: Properties of Finite and Pushdown Transducers. SIAM J. Comput. **12**(2), 300–315 (1983)
3. Hundeshagen, N.: Relations and Transductions Realized by Restarting Automata. Ph.D. thesis, Fachbereich Elektrotechnik/Informatik, Universität Kassel (2013)
4. Hundeshagen, N., Leupold, P.: Transducing by observing and restarting transducers. In: Freund, R., Holzer, M., Truthe, B., Ultes-Nitsche, U. (eds.) NCMA 2012. books@ocg.at, vol. 290, pp. 93–106. Österreichische Computer Gesellschaft, Vienna (2012)
5. Hundeshagen, N., Leupold, P.: Transducing by Observing Length-Reducing and Painter Rules. RAIRO - Theor. Inform. Appl. **48**(1), 85–105 (2014)
6. Hundeshagen, N., Otto, F.: Characterizing the rational functions by restarting transducers. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 325–336. Springer, Heidelberg (2012)
7. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Restarting automata. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 965, pp. 283–292. Springer, Heidelberg (1995)
8. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: On Monotonic Automata with a Restart Operation. J. Auto. Lang. Comb. **4**(4), 287–312 (1999)
9. Kutrib, M., Messerschmidt, H., Otto, F.: On Stateless Two-Pushdown Automata and Restarting Automata. Int. J. Found. Comp. Sci. **21**, 781–798 (2010)
10. Otto, F., Wang, Q.: Weighted Restarting Automata. The results of this paper have been announced at WATA 2014 in Leipzig, May 2014 (submitted)