Theory of Computing Systems (2021) 65:1211–1242
https://doi.org/10.1007/s00224-021-10031-2

Reachability Problems on Reliable and Lossy Queue Automata

Chris Köcher[1]

Accepted: 9 January 2021 / Published online: 19 June 2021
© The Author(s) 2021

Abstract
We study the reachability problem for queue automata and lossy queue automata. Concretely, we consider the set of queue contents which are forwards resp. backwards reachable from a given set of queue contents. Here, we prove the preservation of regularity if the queue automaton loops through some special sets of transformation sequences. This is a generalization of the results by Boigelot et al. and Abdulla et al. regarding queue automata looping through a single sequence of transformations. We also prove that our construction is possible in polynomial time.

Keywords Partially lossy queue · Queue automaton · Reachability · Verification

1 Introduction

Nearly all problems in verification ask whether in a program or automaton one can reach some given configurations from other given configurations. In some computational models this question is decidable, e.g., in finite state machines, pushdown automata [2–4], or counter automata without zero-tests (resp. Petri-nets or vector addition systems) [5, 6]. In some other, mostly Turing-complete computational models, this reachability problem is undecidable. For example, the reachability problem for two-counter automata with zero-tests (so called Minsky-machines) is undecidable [7].

For queue automata reachability is undecidable [8], while this problem is decidable for so-called lossy queue automata [9] which are allowed to forget any parts of their content at any time. In this case, for a regular set of configurations, the set of reachable configurations is regular [10] but in general it is impossible to compute

This is the full version of the conference contribution [1].

✉ Chris Köcher
chris.koecher@tu-ilmenau.de

[1] Automata and Logics Group, Technische Universität Ilmenau, Ilmenau, Germany

a finite automaton accepting this set [11, 12]. Surprisingly, the set of backwards reachable configurations is effectively regular [9], even though this construction is not primitive recursive [13, 14]. This positive reachability result for lossy queue automata was generalized by Abdulla et al. [15] and by Finkel and Schnoebelen [16] to so-called well-structured transition systems, which are systems with an infinite state space and some special restrictions on their transitions.

Another variation of queue automata are automata with priority queues: here, each queue entry has a priority and entries with high priority can supersede or overtake entries with low priority. Haase et al. [17] have proven that the reachability problem is decidable for these priority queue automata with superseding semantics, but it is undecidable for priority queue automata with overtaking semantics.

In this paper we will focus on the reachability problem for reliable and lossy queue automata. Due to its undecidability resp. inefficiency, one may consider approximations of this problem. One trivial approach is to simulate the automaton's computation step by step until a given configuration (or a given set of configurations) is found. In this case, starting from a given set of configurations we simply add or remove a single letter from the queue's contents. An even better and more efficient approach is to consider so-called "meta-transformations" as described in [18, 19]. Such a meta-transformation is a combination of multiple transitions of the queue. In particular, given a loop in the queue's control component one can combine iterations of this loop to one big step of the queue automaton. With this trick it is possible to explore infinitely many contents of the queue in a small amount of time.

Considering reliable queue automata, we know from Boigelot et al. [19] that, starting from a regular language of queue contents, the set of reachable queue contents after iterated application of a single transformation sequence is effectively regular. The authors of that paper also generalized this result to automata having multiple queues. However, in this case we need some special restrictions to the considered loops. Bouajjani and Habermehl [20] extended this result to arbitrary loops. But this requires to consider some proper super-class of the regular languages representing the queues contents. Abdulla et al. also considered in [21] loops in automata having some lossy queues and proved the preservation of regularity.

From the under-approximation considering single loops, one may also obtain so-called *flat automata*. These are automata in which each state belongs to at most one loop. For example, Leroux and Sutre [22] considered flat counter automata and proved that a large class of counter automata are flat or at least flattable. Additionally, flat counter automata have an NP-complete reachability problem [23]. The reachability problem is also NP-complete if we consider flat reliable or lossy queue automata [24, 25].

Here, we consider some extensions of the results from Boigelot et al. and Abdulla et al. Concretely, we consider iterations through certain regular languages, so-called *read-write independent* sets. Such language is the product of a language consisting of write action sequences, only, with another language consisting of read action sequences. For these new meta-transformations we prove the preservation of regularity of sets of configurations. We will see that our construction is possible in polynomial time. Additionally, we show some extensions of our result. For example, we demonstrate that from our main result, one can derive the result of Boigelot et al.

To this end, we show how to turn a single transformation sequence into a read-write independent set yielding the same set of reachable queue contents.

Additionally, we consider another type of meta-transformations: sets of transformations which are closed under some special (context-sensitive) commutations of the atomic transformations. For such meta-transformations, the set of reachable configurations is also effectively regular. Moreover, if we start from a context-free set of configurations, the set of reachable configurations is effectively context-free, again. Both constructions can be carried out in polynomial time.

In the last section of this paper we consider so-called partially lossy queue automata which were first introduced in [26, 27]. This is a generalization of reliable and lossy queue automata where we can specify which letters can be forgotten at any time. These partially lossy queues can also be seen as some kind of the aforementioned priority queues with superseding semantics: here, the forgettable letters have low priority and the unforgettable letters have high priority. In this case, letters with low priority can be superseded by any letter and letters with high priority cannot be superseded by any letter.

We will see, that the sets of reachable configurations can be computed from the ones of a reliable queue automaton. Hence, all of our results do also hold for arbitrary partially lossy queue automata. In particular, we will see that the results from [19, 21] follow from our result. So, we also have a new, unified proof of these two results.

2 Preliminaries

2.1 Words and Languages

At first, we have to introduce some basic definitions. To this end, let Γ be an alphabet. A word $v \in \Gamma^*$ is a prefix of $w \in \Gamma^*$ iff $w \in v\Gamma^*$. Similarly, v is a *suffix* of w iff $w \in \Gamma^* v$ and v is an *infix* of w iff $w \in \Gamma^* v \Gamma^*$. The *complementary prefix* (resp. *suffix*) of w wrt. v is the word $w/v \in \Gamma^*$ (resp. $v\backslash w \in \Gamma^*$) with $w = w/v \cdot v$ (resp. $w = v \cdot v\backslash w$). The *right quotient* of a language $L \subseteq \Gamma^*$ wrt. $K \subseteq \Gamma^*$ is the language $L/K = \{u \in \Gamma^* \mid \exists v \in K : uv \in L\}$. Similarly, we can define the *left quotient* $K\backslash L = \{v \in \Gamma^* \mid \exists u \in K : uv \in L\}$ of L wrt. K.

For a word $w = a_1 a_2 \dots a_n \in \Gamma^*$ we define its *reversal* by $w^{\mathrm{R}} := a_n \dots a_2 a_1$. The *reversal* of a language L is $L^{\mathrm{R}} = \{w^{\mathrm{R}} \mid w \in L\}$. The *shuffle* of two languages L and K is the following language:

$$L \sqcup\!\sqcup K := \left\{ v_1 w_1 v_2 w_2 \dots v_n w_n \;\middle|\; \begin{array}{c} n \in \mathbb{N},\, v_i, w_i \in \Gamma^*, \\ v_1 v_2 \dots v_n \in L,\, w_1 w_2 \dots w_n \in K \end{array} \right\}.$$

Let $\sim$ be an equivalence relation on Γ^*. The *equivalence class* of $v \in \Gamma^*$ wrt. $\sim$ is $[v]_\sim = \{u \in \Gamma^* \mid u \sim v\}$. A language $L \subseteq \Gamma^*$ is *closed under* $\sim$ if for each $v \in L$ we have $[v]_\sim \subseteq L$.

Let $S \subseteq \Gamma$. Then the *projection* $\pi_S \colon \Gamma^* \to S^*$ to S is the monoid homomorphism induced by $\pi_S(a) = a$ for each $a \in S$ and $\pi_S(a) = \varepsilon$ for each $a \in \Gamma \backslash S$. Additionally, for $w \in \Gamma^*$ we write $|w|_S := |\pi_S(w)|$.

2.2 Finite Automata

A *(nondeterministic) finite automaton* (*NFA* for short) is a quintuple $\mathfrak{A} = (Q, \Gamma, I, \Delta, F)$ where Q is a finite set of states, $I, F \subseteq Q$ are the sets of initial and final states, and $\Delta \subseteq Q \times \Gamma \times Q$ is the transition relation. Then, the *configuration graph* of $\mathfrak{A}$ is $\mathcal{G}_\mathfrak{A} := (Q, \Delta)$ which is a finite, edge-labeled, and directed graph. For $p, q \in Q$ and $w \in \Gamma^*$ we write $p \xrightarrow{w}_\mathfrak{A} q$ if there is a w-labeled path in $\mathcal{G}_\mathfrak{A}$ from p to q. For $Q_1, Q_2 \subseteq Q$ and $w \in \Gamma^*$ we write $Q_1 \xrightarrow{w}_\mathfrak{A} Q_2$ if there are $q_1 \in Q_1$ and $q_2 \in Q_2$ with $q_1 \xrightarrow{w}_\mathfrak{A} q_2$. The *accepted language* of $\mathfrak{A}$ is $L(\mathfrak{A}) := \{w \in \Gamma^* \mid I \xrightarrow{w}_\mathfrak{A} F\}$. A language $L \subseteq \Gamma^*$ is *regular*, if there is an NFA $\mathfrak{A}$ accepting L. The class of regular languages is effectively closed under Boolean operations, left and right quotients, shuffle, reversal, and projections.

Let $\mathfrak{A} = (Q, \Gamma, I, \Delta, F)$ be an NFA, $Q_i, Q_f \subseteq Q$. Then we set $\mathfrak{A}_{Q_i \to Q_f} := (Q, \Gamma, Q_i, \Delta, Q_f)$, i.e., $\mathfrak{A}_{Q_i \to Q_f}$ is the NFA constructed from $\mathfrak{A}$ with initial states Q_i and final states Q_f. For example, we have

$$L(\mathfrak{A}) = \bigcup_{q \in Q} L(\mathfrak{A}_{I \to q}) L(\mathfrak{A}_{q \to F}) .$$

3 Queue and Pushdown Automata

In this section we will recall basic knowledge on (fifo-)queues and (lifo-)stacks. Both data structures can store entries from a given alphabet A. Then, the contents of such a queue or stack are words from A^*. For each letter $a \in A$ we have two *actions* (or *transformations*): writing of a into the structure (denoted by a) and reading of a from the structure (denoted by $\overline{a}$). We assume that the alphabet $\overline{A}$ containing each such reading operation $\overline{a}$ is a disjoint copy of A. By $\Sigma_A := A \cup \overline{A}$ we denote the set of all actions on the data structure. For $w = a_1 a_2 \dots a_n \in A^*$ we also write $\overline{w} := \overline{a_1}\,\overline{a_2} \dots \overline{a_n}$ and for $L \subseteq A^*$ we write $\overline{L} := \{\overline{w} \mid w \in L\}$.

In the following two subsections we will consider queues and stacks separately.

3.1 Queue Automata

In queues (or channels) we always write letters on one end of the queue's content and read them from the other end. Hence, writing the letter $a \in A$ into the queue with content $w \in A^*$ results in wa and reading a from aw yields the queue content w. It is impossible to read a from the empty queue or whenever the queue's content is bw with $a \neq b$.

Formally, a *queue automaton* is a tuple $\mathfrak{Q} = (Q, \Gamma, A, I, \Delta, F)$ where Q is a finite set of states, Γ and A are two (not necessarily disjoint) alphabets, $I, F \subseteq Q$ are the sets of initial and final states, respectively, and $\Delta \subseteq Q \times (\Gamma \cup \{\varepsilon\}) \times (\Sigma_A \cup \{\varepsilon\}) \times Q$ is the transition relation. A *configuration* of $\mathfrak{Q}$ is a tuple from $\mathrm{Conf}_\mathfrak{Q} := Q \times A^*$. We denote the set of initial configurations by $\mathrm{Init}_\mathfrak{Q} := I \times \{\varepsilon\}$ and the set of accepting

configurations by $\text{Final}_{\mathfrak{Q}} := F \times A^*$. For $p, q \in Q$, $v, w \in A^*$, and $\alpha \in \Gamma \cup \{\varepsilon\}$ we write $(p, v) \xrightarrow{\alpha}_{\mathfrak{Q}} (q, w)$ if one of the following holds:

(1) there is $a \in A$ with $(p, \alpha, a, q) \in \Delta$ and $va = w$,
(2) there is $a \in A$ with $(p, \alpha, \overline{a}, q) \in \Delta$ and $v = aw$, or
(3) we have $(p, \alpha, \varepsilon, q) \in \Delta$ and $v = w$.

Then $\mathcal{G}_{\mathfrak{Q}} := (\text{Conf}_{\mathfrak{Q}}, \bigcup_{\alpha \in \Gamma \cup \{\varepsilon\}} \xrightarrow{\alpha}_{\mathfrak{Q}})$ is called the *configuration graph* of $\mathfrak{Q}$. For $(p, v), (q, w) \in \text{Conf}_{\mathfrak{Q}}$ and $\gamma \in \Gamma^*$ we write $(p, v) \xrightarrow{\gamma}_{\mathfrak{Q}} (q, w)$ if there is a γ-labeled path from (p, v) to (q, w) in $\mathcal{G}_{\mathfrak{Q}}$. The accepted language of $\mathfrak{Q}$ is $L(\mathfrak{Q}) := \{\gamma \in \Gamma^* \mid \text{Init}_{\mathfrak{Q}} \xrightarrow{\gamma}_{\mathfrak{Q}} \text{Final}_{\mathfrak{Q}}\}$.

It is well-known that queue automata can simulate Turing-machines (cf. [8]). Hence, queue automata accept exactly the class of recursively enumerable languages.

In the following, we remove the input tape of our queue automata and focus on the behavior of the memory component (i.e. the queue). Formally, we describe a queue's behavior by a function $\circ$ associating a word $v \in A^*$ and a sequence of transformations $t \in \Sigma_A^*$ with another word $v \circ t \in A^*$ which is the queue's content after application of t on the content v. Since it is impossible to read a from a queue with content ε or bw with $a \neq b$, the function $\circ$ is not total. However, we may introduce a new content $\perp \notin A^*$ (the so-called *error state*) and set $v \circ t = \perp$ whenever the application of t on v is not possible.

Definition 3.1 Let A be an alphabet and $\perp \notin A$. Then the map $\circ\colon (A^* \cup \{\perp\}) \times \Sigma_A^* \to (A^* \cup \{\perp\})$ is defined for each $v \in A^*$, $a, b \in A$ with $a \neq b$, and $t \in \Sigma_A^*$ as follows:

(i) $v \circ \varepsilon = v$
(ii) $v \circ at = va \circ t$
(iii) $av \circ \overline{a}t = v \circ t$
(iv) $bv \circ \overline{a}t = \varepsilon \circ \overline{a}t = \perp \circ t = \perp$

We will say "$v \circ t$ is undefined" if $v \circ t = \perp$.

Let $\mathfrak{Q} = (Q, \Gamma, A, I, \Delta, F)$ be some queue automaton. Construct the following NFA (with ε-transitions) $\mathfrak{T} = (Q, \Sigma_A, I, \Delta', F)$ with

$$\Delta' = \{(p, \alpha, q) \mid \exists \gamma \in \Gamma \cup \{\varepsilon\}\colon (p, \gamma, \alpha, q) \in \Delta\}.$$

Then $(\varepsilon \circ L(\mathfrak{T})) \setminus \{\perp\}$ is exactly the set of all queue contents after any computation of $\mathfrak{Q}$. Note that $\mathfrak{Q}$ will not end up in an error state if it is impossible to read the letter from the queue's head position. Instead the queue automaton will stop in such situation. Hence, we exclude $\perp$ from $\varepsilon \circ L(\mathfrak{T})$.

More generally, we will consider sets $L \circ T$ for some set of initial queue contents $L \subseteq A^*$ and some set of transformation sequences $T \subseteq \Sigma_A^*$. At this juncture, it suffices to regard only regular languages $T \subseteq \Sigma_A^*$ since the control component $\mathfrak{T}$ of a queue automaton is always an NFA. All in all, we may define our reachability problems as follows:

Problem 3.2 Let A be an alphabet, $L \subseteq A^*$ be a set of queue contents, and $T \subseteq \Sigma_A^*$ be a regular set of transformation sequences. The set of queue contents that are reachable from L via T is

$$\textsc{Reach}(L, T) := (L \circ T) \setminus \{\bot\}$$

and the set of queue contents that can reach L via T is

$$\textsc{BackReach}(L, T) := \{v \in A^* \mid (v \circ T) \cap L \neq \emptyset\}.$$

From the definition of $\circ$ we already know that $v \circ a = va$ and $av \circ \overline{a} = v$ holds. In this sense, we may see some duality between the write and read actions a and $\overline{a}$. This duality can be extended as follows: by $d\colon \Sigma_A^* \to \Sigma_A^*$ we denote the map defined by

$$d(\varepsilon) = \varepsilon, \quad d(av) = d(v)\overline{a}, \text{ and } \quad d(\overline{a}v) = d(v)a$$

for each $a \in A$ and $v \in \Sigma_A^*$. We can see that d is a bijective antimorphism and an involution. Additionally, from [28, Lemma 3.3] we know that $v \circ t = w$ holds if, and only if, $w^{\mathrm{R}} \circ d(t) = v^{\mathrm{R}}$ for each $v, w \in A^*$ and $t \in \Sigma_A^*$. From this equivalence we also obtain the following duality between Reach and BackReach:

Theorem 3.3 *Let A be an alphabet, $K, L \subseteq A^*$, and $T \subseteq \Sigma_A^*$. Then we have*

$$\textsc{BackReach}(L, T) = \textsc{Reach}(L^{\mathrm{R}}, d(T))^{\mathrm{R}}.$$

Proof Let $v \in \textsc{BackReach}(L, T)$. By definition we have $(v \circ T) \cap L \neq \emptyset$. So, there are $w \in L$ and $t \in T$ with $v \circ t = w$. By [28, Lemma 3.3] we also have $w^{\mathrm{R}} \circ d(t) = v^{\mathrm{R}}$ implying $v^{\mathrm{R}} \in \textsc{Reach}(L^{\mathrm{R}}, d(T))$, i.e., $v \in \textsc{Reach}(L^{\mathrm{R}}, d(T))^{\mathrm{R}}$.

Conversely, let $v \in \textsc{Reach}(L^{\mathrm{R}}, d(T))^{\mathrm{R}}$. We know $v^{\mathrm{R}} \in \textsc{Reach}(L^{\mathrm{R}}, d(T))$. Hence, there is $w \in L$ and $t \in T$ such that $w^{\mathrm{R}} \circ d(t) = v^{\mathrm{R}}$. Due to [28, Lemma 3.3] we also have $v \circ t = w$ implying $w \in (v \circ T) \cap L$. In other words, we have $v \in \textsc{BackReach}(L, T)$. □

So, with the help of Theorem 3.3 it suffices to consider forwards reachability from now on.

Now, let $L \subseteq A^*$ be a recursively enumerable language of queue contents and $T \subseteq \Sigma_A^*$ a regular language of queue transformations. Then the language $\textsc{Reach}(L, T)$ is (effectively) recursively enumerable. However, since queue automata can simulate Turing-machines, the language $\textsc{Reach}(L, T)$ can be any recursively enumerable language - even if L and T are somewhat "simple" languages:

Remark 3.4 Let $K \subseteq \Gamma^*$ be a recursively enumerable language. Then there is a (type-0) grammar $\mathfrak{G} = (N, \Gamma, P, S)$ with $K = L(\mathfrak{G})$. Let $\# \notin N \cup \Gamma$ be some new letter. We set our alphabet $A := N \cup \Gamma \cup \{\#\}$ (recall that this is the set of possible queue entries). We construct the language of transformations $T \subseteq \Sigma_A^*$ as follows:

$$T := \left(\{\overline{\ell} r \mid (\ell, r) \in P\} \cup \{\overline{a}a \mid a \in N \cup \Gamma \cup \{\#\}\}\right)^* \overline{\#},$$

i.e., the queue automaton can apply any rule from $\mathfrak{G}$ and move any letter from the head to its end. Then we have

$$\textsc{Reach}(\{\#S\}, T) \cap \Gamma^* = L(\mathfrak{G}) = K\,.$$

In other words, $\textsc{Reach}(L, T)$ can be any recursively enumerable language K even if L is a singleton and T is essentially the Kleene-closure of a finite set of transformation sequences.

Due to Remark 3.4 there are regular languages L and T such that $\textsc{Reach}(L, T)$ is undecidable. Therefore, we need an approximation to decide whether a given regular set of configurations can be reached from the regular language L of queue inputs by application of the transformation sequences from T. A trivial approach is to simulate the computation of the queue automaton step-by-step. That means, starting with L we iteratively compute the set of all queue contents which are reachable from L after n steps. Formally, for the set $T_n \subseteq \Sigma_A^*$ of prefixes of length at most n of T we compute $L_n = \textsc{Reach}(L, T_n)$ for increasing $n \in \mathbb{N}$. Unfortunately, this algorithm is not very efficient: consider a finite language of queue contents $L \subseteq A^*$ and a regular language of transformation sequences $T \subseteq \Sigma_A^*$. Then $T_n \subseteq \Sigma_A^{\leq n}$ is finite for any $n \in \mathbb{N}$ and, hence, $L_n = \textsc{Reach}(L, T_n)$ is finite.

Boigelot et al. improved this trivial approximation in [18, 19] by the introduction of so-called *meta-transformations*. These are certain regular languages $S \subseteq \Sigma_A^*$ such that the sets $\textsc{Reach}(L, S)$ (for any regular set $L \subseteq A^*$) are effectively regular. Then the trivial approximation can be modified as follows: whenever we compute L_{n+1} from L_n we search for such meta-transformation in the queue automaton's control component starting from T_n and apply it on L_n. In [19] the authors considered meta-transformations of the form $S = \{t\}^*$ for some $t \in \Sigma_A^*$. In fact, this approach is more efficient than the trivial one, since we can explore an infinite state space in just one step of the algorithm.

From the following proposition we can obtain some further simple meta-transformations. Concretely, we consider the case that T contains only sequences of write actions or only sequences of read actions.

Proposition 3.5 *Let A be an alphabet and $L, T \subseteq A^*$. Then the following statements hold:*

(1) $\textsc{Reach}(L, T) = LT$
(2) $\textsc{Reach}(L, \overline{T}) = T \backslash L$

Note that this proposition is a generalization of Theorems 1 and 2 in [18]. In that paper, Boigelot and Godefroid have proven the effective recognizability of $\textsc{Reach}(L, t)$ and $\textsc{Reach}(L, \bar{t})$ where $L \subseteq A^*$ is regular and $t \in A^*$ is a single transformation sequence.

Proof First, consider equation (1): let $w \in \textsc{Reach}(L, T) = (L \circ T) \setminus \{\bot\}$. Then there are $v \in L$ and $t \in T$ with $w = v \circ t$. Since $t \in T \subseteq A^*$ we have, by iterated application of (ii) in Definition 3.1, $w = v \circ t = vt \in LT$. Now, let $w \in LT$. Then there are $v \in L$ and $t \in T$ with $w = vt$. Again, by application of (ii) in

Definition 3.1 we have $w = vt = v \circ t \in L \circ T$. Since $L, T \subseteq A^*$ we infer that $w \in (L \circ T) \setminus \{\bot\} = \textsc{Reach}(L, T)$ holds.

Using (iii) in Definition 3.1, we can similarly prove equation (2). □

Combining Theorem 3.3 and Proposition 3.5 we obtain the following two equations:

$$\textsc{BackReach}(L, T) = L/T \quad \text{and} \quad \textsc{BackReach}(L, \overline{T}) = TL$$

for each pair of languages $L, T \subseteq A^*$.

Now, let $L, T \subseteq A^*$ be two regular languages accepted by the NFAs $\mathfrak{L}$ and $\mathfrak{T}$, respectively. Then, using the classical constructions, we can construct an NFA accepting $\textsc{Reach}(L, T)$ in quadratic time. An NFA accepting $\textsc{Reach}(L, \overline{T})$ can be constructed in cubic time. The number of states of these NFAs is linear in the number of states in $\mathfrak{L}$ and $\mathfrak{T}$.

If we require these languages to be accepted by a DFA, then we additionally need to determinize the constructed NFAs resulting in exponential size and time. The complexities of BackReach are similar to the ones of Reach due to the duality stated in Proposition 3.5. However, if $\mathfrak{L}$ is a DFA, we still can compute a DFA accepting $\textsc{BackReach}(L, T)$ in cubic time having a linear number of states (in this case we only modify the accepting states of $\mathfrak{L}$).

Later in this paper we consider two further types of meta-transformations T having mappings $L \mapsto \textsc{Reach}(L, T)$ and $L \mapsto \textsc{BackReach}(L, T)$ which preserve regularity efficiently.

3.2 Pushdown Automata

Recall that stacks (or pushdowns) have the same set of actions on their content as queues. In other words, we are able to write a letter into the stack's content or read a letter from the stack. While queues apply their read and write actions on different ends of their content, stacks execute these actions always on the same end. Formally, writing a letter $a \in A$ into the stack $w \in A^*$ results in the content aw and reading a from aw makes w. Note that it is impossible to read a from bw where $a \neq b$. Similarly, the stack blocks when reading a from ε.

In this paper, a *pushdown automaton* (or *PDA* for short) is defined similarly to queue automata. Also the definitions of their configuration graphs and accepted languages are similar. The only exception is Transition (1) which has to be rephrased as follows:

(1') there is $a \in A$ with $(p, \alpha, a, q) \in \Delta$ and $av = w$,

Note that our definition of pushdown automata slightly differs from the classical definitions in textbooks (cf. for example [29]). While our automata apply at most one action on their stack on each transition, in the classical definition the PDAs always read one letter and write a sequence of letters afterwards. However, both definitions are equivalent. A classical PDA can be transformed into our model by splitting transitions into a sequence of transitions applying exactly one action. Conversely, a

transition writing a from our definition can be translated into transitions applying $\overline{b}ba$ for each $b \in A$.

Due to the equivalence of these models, we are allowed to call a language $L \subseteq \Gamma^*$ *context-free* if there is a PDA $\mathfrak{P}$ with $L = L(\mathfrak{P})$.

Let $C \subseteq \mathrm{Conf}_{\mathfrak{P}}$ be a set of configurations of $\mathfrak{P}$. Then we denote the set of configurations of $\mathfrak{P}$ reachable from C by

$$\mathrm{post}^*(C) := \{d \in \mathrm{Conf}_{\mathfrak{P}} \mid \exists \gamma \in \Gamma^* \colon C \xrightarrow{\gamma}_{\mathfrak{P}} d\}\,.$$

The following result is well-known in verification:

Theorem 3.6 ([3, 4, 30]) *Let $\mathfrak{P} = (Q, \Gamma, A, I, \Delta, F)$ be a PDA, $p \in Q$, and $\mathfrak{A}$ be an NFA over A. Then we can compute an NFA $\mathfrak{B}_q$ over A for each $q \in Q$ such that*

$$post^*(\{p\} \times L(\mathfrak{A})) = \bigcup_{q \in Q} \{q\} \times L(\mathfrak{B}_q)$$

holds. The construction of the NFAs $\mathfrak{B}_q$ is possible in polynomial time.

Concretely, using the construction from [3], we obtain NFAs $\mathfrak{B}_q$ having $\mathcal{O}(n_{\mathfrak{P}} + n_{\mathfrak{A}})$ states, where [...] $n_{\mathfrak{P}}$ and $n_{\mathfrak{A}}$ are the numbers of states of a given PDA $\mathfrak{P}$ and a given NFA $\mathfrak{A}$.

4 Behavioral Equivalence of Queue Transformations

The first type of meta-transformations we want to consider are languages that are closed under the so-called behavioral equivalence. To this end, let $v \in A^*$ be an arbitrary queue content and $a \in A$. Then we have

$$v \circ aa\overline{a} = vaa \circ \overline{a} = (va \circ \overline{a}) \cdot a = v \circ a\overline{a}a\,.$$

In other words, the queue transformation sequences $aa\overline{a}$ and $a\overline{a}a$ have the same effect on any queue content. Then we say that these two sequences behave equivalently. On the other hand, we have $\varepsilon \circ a\overline{a} = \varepsilon \neq \bot = \varepsilon \circ \overline{a}a$ which witnesses that $a\overline{a}$ and $\overline{a}a$ do not behave equivalently.

Formally, this equivalence is defined as follows:

Definition 4.1 Let A be an alphabet and $s, t \in \Sigma_A^*$. Then s and t *behave equivalently* (denoted by $s \equiv t$) if $v \circ s = v \circ t$ for each $v \in A^*$. The induced relation $\equiv$ is called the *behavioral equivalence*.

In other words, we have $s \equiv t$ if for each queue input the application of s and t lead to the same output of the queue automaton. As we have seen above we know $aa\overline{a} \equiv a\overline{a}a$ and $a\overline{a} \not\equiv \overline{a}a$.

This equivalence relation was first introduced by Huschenbett et al. in [28]. It was used in that paper to define the transformation monoid of a queue (the so-called queue monoid). This monoid consists of the equivalence classes of the behavioral

equivalence with composition. Moreover, they proved that $\equiv$ is a congruence on Σ_A^* which is described by a finite set of equations. We recall these context-sensitive commutations in the following theorem:

Theorem 4.2 ([28, Theorem 4.3]) *Let A be an alphabet. Then $\equiv$ is the least congruence on Σ_A^* satisfying the following equations for each $a, b \in A$:*

(1) $a\overline{b} \equiv \overline{b}a$ *if* $a \neq b$,
(2) $a\overline{a}\overline{b} \equiv \overline{a}a\overline{b}$, *and*
(3) $ba\overline{a} \equiv b\overline{a}a$.

In this section we want to prove that, for regular languages which are closed under the behavioral equivalence, the mappings $L \mapsto \textsc{Reach}(L, T)$ and $L \mapsto \textsc{BackReach}(L, T)$ effectively and even efficiently preserve regularity. We prove this with the help of the following corollary of Theorem 4.2 stating that each transformation sequence $t \in \Sigma_A^*$ has some equivalently behaving transformation sequence $s \in \Sigma_A^*$ which is in some sense "simple":

Proposition 4.3 ([28, Lemma 5.2]) *Let A be an alphabet and $t \in \Sigma_A^*$. Then there is $s \in \overline{A}^* A^* \overline{A}^*$ with $s \equiv t$. From a given word t we can compute such a word s in polynomial time.*

Proof (idea) We construct some confluent and terminating semi-Thue system $\mathfrak{R}$ by ordering the equations in Theorem 4.2 from left to right. Then from $t \in \Sigma_A^*$ we can compute a unique word $r \in A^*\big(\bigcup_{a\in A} a\overline{a}\big)^* A^*$. This word r identifies the whole equivalence class of t. Assume that $r = \overline{r_1}\, a_1\overline{a_1}a_2\overline{a_2}\ldots a_n\overline{a_n}\, r_2$ (where $r_1, r_2 \in A^*$, $a_1, \ldots, a_n \in A$) is this unique word. Then we can set $s := \overline{r_1}\, a_1 a_2 \ldots a_n r_2\, \overline{a_1 a_2 \ldots a_n}$ and we have $t \equiv r \equiv s$ by [28, Lemma 5.2]. □

Remark 4.4 The algorithm from Proposition 4.3 returns a word $s \in \overline{A}^* A^* \overline{A}^*$ which is unique for each word from the equivalence class $[t]_\equiv$. However, there are words $t \in A^*$ having multiple $s \in \overline{A}^* A^* \overline{A}^*$ which behave equivalent. For example, let $a, b \in A$ be two distinct letters and $t = a\overline{a}\overline{b}a$. Then the algorithm outputs $\overline{a}\overline{b}aa$ but we also have $\overline{a}aa\overline{b}$, $aa\overline{a}\overline{b} \in [t]_\equiv$.

The behavioral equivalence of queue transformations was further considered in [27, 28]. Concretely, in that papers the authors studied those regular languages which are closed under the behavioral equivalence $\equiv$. In [27, Theorem 4.1] we defined some kind of rational expressions constructing these sets as well as an MSO-logic describing them. In particular, let $T \subseteq \Sigma_A^*$ be a language that is closed under $\equiv$. Then, we know that T is regular if, and only if, $T \cap \overline{A}^* A^* \overline{A}^*$ is regular due to [28, Theorem 9.4].

Example 4.5 Let $R, W \subseteq A^*$ be regular languages. Then R can be accepted by an NFA $\mathfrak{R} = (Q, A, I, \Delta, F)$. In this case, we have

$$[\overline{R} \sqcup\!\sqcup W]_{\equiv} \cap \overline{A}^* A^* \overline{A}^* = \bigcup_{q \in Q} \overline{L(\mathfrak{R}_{I \to q})} W \overline{L(\mathfrak{R}_{q \to F})},$$

which is regular (recall that $[L]_{\equiv}$ is the closure of $L \subseteq \Sigma_A^*$ under $\equiv$ and $\sqcup\!\sqcup$ is the shuffle of two languages). Hence, $[\overline{R} \sqcup\!\sqcup W]_{\equiv}$ is regular and closed under $\equiv$.

Now, let $a \in A$. Then $[(a\overline{a})^*]_{\equiv}$ is *not* regular since (by Theorem 4.2) we can prove $[(a\overline{a})^*]_{\equiv} \cap \overline{A}^* A^* \overline{A}^* = \{a^n \overline{a}^n \mid n \in \mathbb{N}\}$ which is not regular.

The following lemma states that closure under behavioral equivalence is decidable for regular languages:

Lemma 4.6 *Let A be an alphabet. Given a regular language $T \subseteq \Sigma_A^*$ of queue transformations, it is decidable whether T is closed under behavioral equivalence $\equiv$.*

Proof (idea) We can check this question with the help of a rational transduction (cf. [31], Chapter III): let τ be the transduction with the graph

$$G := I_{\Sigma_A}^* \{(\ell, r), (r, \ell) \mid \ell \equiv r \text{ is an equation in Theorem 4.2}\} I_{\Sigma_A}^* \cup I_{\Sigma_A}^*$$

where $I_{\Sigma_A} = \{(\alpha, \alpha) \mid \alpha \in \Sigma_A\}$ is the identity on Σ_A. It is a simple task to prove that G is rational in $\Sigma_A^* \times \Sigma_A^*$. Then $T \subseteq \Sigma_A^*$ is closed under $\equiv$ if, and only if, $T = \tau(T)$ holds. We can check this equation since $\tau(T)$ is effectively regular. □

However, it is undecidable whether the closure of a given regular language T under $\equiv$ is regular as well [28, Theorem 8.4].

Finally, we are able to prove the main theorem in this section:

Theorem 4.7 *Let A be an alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma_A^*$ be regular and closed under $\equiv$. Then* $\textsc{Reach}(L, T)$ *and* $\textsc{BackReach}(L, T)$ *are effectively regular. In particular, from NFAs accepting L and T we can construct NFAs accepting* $\textsc{Reach}(L, T)$ *and* $\textsc{BackReach}(L, T)$ *in polynomial time.*

Proof We first prove that

$$\textsc{Reach}(L, T) = \textsc{Reach}(L, T \cap \overline{A}^* A^* \overline{A}^*)$$

holds. The inclusion "$\supseteq$" is trivial. Towards the converse inclusion, let $w \in \textsc{Reach}(L, T)$. Then there are $v \in L$ and $t \in T$ with $v \circ t = w$. Due to Proposition 4.3 there is $s \in \overline{A}^* A^* \overline{A}^*$ with $s \equiv t$. By definition of $\equiv$ we also have $v \circ s = v \circ t = w$. Since T is closed under $\equiv$ we have $s \in T \cap \overline{A}^* A^* \overline{A}^*$. This finally implies $w \in \textsc{Reach}(L, T \cap \overline{A}^* A^* \overline{A}^*)$.

Next, we compute $\textsc{Reach}(L, T \cap \overline{A}^* A^* \overline{A}^*)$. To this end, let $\mathfrak{T} = (Q, \Sigma_A, I, \Delta, F)$ be an NFA with $L(\mathfrak{T}) = T$. We partition $T \cap \overline{A}^* A^* \overline{A}^*$ as follows:

let $p, q \in Q$ be any pair of states. Then we can compute the following three regular languages in polynomial time:

$$\overline{K_1^{p,q}} = L(\mathfrak{T}_{I\to p}) \cap \overline{A}^*, \quad K_2^{p,q} = L(\mathfrak{T}_{p\to q}) \cap A^*, \text{ and } \quad \overline{K_3^{p,q}} = L(\mathfrak{T}_{q\to F}) \cap \overline{A}^*.$$

Then it is easy to see that $T \cap \overline{A}^* A^* \overline{A}^* = \bigcup_{p,q\in Q} \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}$ holds. Hence, we have

$$\begin{aligned} \textsc{Reach}(L, T) &= \textsc{Reach}\left(L, \bigcup_{p,q\in Q} \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}\right) \\ &= \bigcup_{p,q\in Q} \textsc{Reach}(L, \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}). \end{aligned}$$

So, let $p, q \in Q$. By Proposition 3.5 reading from the queue corresponds to taking the left-quotient and writing into the queue corresponds to concatenation. Therefore, we have:

$$\textsc{Reach}(L, \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}) = K_3^{p,q} \backslash ((K_1^{p,q} \backslash L) \cdot K_2^{p,q}).$$

Hence, due to closure properties of the class of regular languages, $\textsc{Reach}(L, T)$ is effectively regular. Since all of the needed closure properties are also efficient and since we are considering only $\mathcal{O}(|Q|^2)$ many languages $K_i^{p,q}$, an NFA accepting $\textsc{Reach}(L, T)$ can be computed in polynomial time. This NFA has a cubic number of states.

Finally, we have to show that $\textsc{BackReach}(L, T)$ is effectively and even efficiently regular. Recall that $\textsc{BackReach}(L, T) = \textsc{Reach}(L^{\mathrm{R}}, d(T))^{\mathrm{R}}$ holds. Due to [28, Proposition 3.4] the language $d(T)$ is still closed under behavioral equivalence. Additionally, $d(T)$ is effectively regular since we only have to replace a by $\overline{a}$ and vice versa and to invert the edges of the automaton accepting T. Since the class of regular languages is efficiently closed under reversal we can compute an automaton accepting $\textsc{BackReach}(L, T)$ in polynomial time which has a cubic number of states. □

Note that, due to the proof of this theorem, the map $L \mapsto \textsc{Reach}(L, T)$ also preserves regularity if $T \subseteq \overline{A}^* A^* \overline{A}^*$ holds. It is also possible to extend the result of the previous theorem to context-free languages:

Theorem 4.8 *Let A be an alphabet, $L \subseteq A^*$ be context-free, and $T \subseteq \Sigma_A^*$ be regular and closed under $\equiv$. Then $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ are effectively context-free. In particular, from a PDA accepting L and an NFA accepting T, we can construct PDAs accepting $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$, respectively, in polynomial time.*

Proof (idea) This is similar to the proof of Theorem 4.7 due to the effective and efficient closure properties of context-free languages (note that the left or right quotient of a context-free language wrt. a regular language is context-free, again). □

5 Read-Write Independence

In this section we want to consider another kind of meta-transformations: cyclic regular languages. In other words, given two regular languages $L \subseteq A^*$ and $T \subseteq \Sigma_A^*$, we want to compute an NFA accepting $\textsc{Reach}(L, T^*)$. The case, where T is a singleton, was first considered by Boigelot et al. in [18, 19] (and similarly by Abdulla et al. [21] for lossy queues). In these papers the authors proved that $\textsc{Reach}(L, T^*)$ is effectively regular in this case. So, a natural question would be to ask, whether this result also holds if T is no singleton. Unfortunately, we have seen in Remark 3.4 that $\textsc{Reach}(L, T^*)$ can be undecidable if T is a finite language. The following example proves that $\textsc{Reach}(L, T^*)$ is not necessarily regular anymore even if T consists of two words:

Example 5.1 Let A be an alphabet and $a, b \in A$ be distinct letters. Then we have

$$\textsc{Reach}(\{a\}, \{\overline{a}bb, \overline{b}a\}^*) \cap \{a\}^* = \{a^{2^n} \mid n \in \mathbb{N}\}$$

which is not even context-free.

In both cases, Example 5.1 and Remark 3.4, the write actions of any sequence $t \in T$ depend on the read actions in t. So, we are able to copy data from the head of the queue into its tail. Then, we can see the queue as a Turing-tape and we are able to move the head on this tape in any direction. Hence, we consider languages $T \subseteq \Sigma_A^*$ in which for each pair $s, t \in T$ there is another word $r \in T$ consisting of the write actions from s and the read actions from t. In this case, independently of the word from $\pi_A(T)$ we write into the queue, we can read any word from $\pi_{\overline{A}}(T)$. Formally, we are considering the following sets of sequences of transformations:

Definition 5.2 Let A be an alphabet. A set $T \subseteq \Sigma_A^*$ is *read-write independent* if, for each $s, t \in T$, we have $\pi_A(s)\pi_{\overline{A}}(t) \in T$. In other words, T is read-write independent if, and only if, $\pi_A(T)\pi_{\overline{A}}(T) \subseteq T$ holds.

We may see read-write independent sets T as some kind of a Cartesian product of a set of sequences of write actions $W \subseteq A^*$ with a set of read action sequences $\overline{R} \subseteq \overline{A}^*$ where for each element $(w, \overline{r}) \in W \times \overline{R}$ we have the transformation $w\overline{r} \in T$. Some simple read-write independent sets are listed in the following example:

Example 5.3 Let $W, R \subseteq A^*$. Then $W\overline{R}$ and $W \sqcup\!\sqcup \overline{R}$ are read-write independent sets.

Since the class of regular languages is closed under projections and concatenation and due to the decidability of the inclusion problem, we can decide whether a given regular language $T \subseteq \Sigma_A^*$ is read-write independent.

For our further considerations of read-write independent sets we need the following lemma. It states that we can "de-shuffle" those languages:

Lemma 5.4 ([26, Lemma 3.11]) *Let A be an alphabet, $L \subseteq A^*$, and $T \subseteq \Sigma_A^*$ be read-write independent. Then we have*

$$\textsc{Reach}(L, T) = \textsc{Reach}(L, \pi_A(T)\pi_{\overline{A}}(T)) . \qquad \square$$

Note that Lemma 5.4 does not hold for arbitrary languages $T \subseteq \Sigma_A^*$. For example, consider $L = \{\varepsilon\}$ and $T = \{\overline{a}a\}$. Then we know $\varepsilon \circ \overline{a}a = \bot$ and $\varepsilon \circ a\overline{a} = \varepsilon$ resulting in $\textsc{Reach}(\varepsilon, \overline{a}a) = \emptyset \subsetneq \{\varepsilon\} = \textsc{Reach}(\varepsilon, a\overline{a})$. However, the following inequation holds for any language $T \subseteq \Sigma_A^*$ - even if T is not read-write independent:

$$\textsc{Reach}(L, T) \subseteq \textsc{Reach}(L, \pi_A(T)\pi_{\overline{A}}(T)) .$$

Now, consider $T := \pi_A(t) \sqcup\!\sqcup \pi_{\overline{A}}(t)$ for a word $t \in \Sigma_A^*$. This language is read-write independent. Due to Theorem 4.2 T is also closed under behavioral equivalence, i.e., we can compute $\textsc{Reach}(L, T)$ in polynomial time. However, T^* is not necessarily closed. Hence, we cannot apply Theorem 4.7 to compute $\textsc{Reach}(L, T^*)$. By Lemma 5.4 we infer

$$\textsc{Reach}(L, T^*) = \textsc{Reach}(L, (\pi_A(T)\pi_{\overline{A}}(T))^*) = \textsc{Reach}(L, (\pi_A(t)\pi_{\overline{A}}(t))^*) .$$

Since $\pi_A(t)\pi_{\overline{A}}(t) \in \Sigma_A^*$ the map $L \mapsto \textsc{Reach}(L, T^*)$ preserves regularity efficiently by [19].

In the following, we will prove that, provided T is any regular and read-write independent language, the mapping $L \mapsto \textsc{Reach}(L, T^*)$ preserves regularity effectively and efficiently (cf. Theorem 5.11). By Lemma 5.4 it suffices to consider languages $T = W\overline{R}$ where $W, R \subseteq A^*$ are two regular languages. But before we show this general case, we make some additional assumptions on the languages W and R. Afterwards we derive the general case from this particular case. Concretely, we consider regular subsets $W\overline{R} \subseteq A^*\overline{A}^*$ where A is an alphabet having a special letter \$ which marks the beginning of each word from W and is used for synchronization between writing and reading actions. In this connection, we have to ensure that the \$ can be read whenever it occurs on the queue's head. We do this by insertion of arbitrarily many \$'s at any position in R. In other words, we require $R = \$^* \sqcup\!\sqcup R$.

Theorem 5.5 *Let A be an alphabet and $\$ \in A$ be some letter. Additionally, let $L \subseteq (A \setminus \{\$\})^*$, $W \subseteq \$(A \setminus \{\$\})^*$, and $R \subseteq A^*$ be regular languages such that $R = \$^* \sqcup\!\sqcup R$ holds. Then $\textsc{Reach}(L, (W\overline{R})^*)$ is effectively regular. In particular, from NFAs accepting L, W, and R we can construct an NFA accepting $\textsc{Reach}(L, (W\overline{R})^*)$ in polynomial time.*

The proof of this result can be found on page 17.

5.1 The Reduction to Pushdown Automata

We prove Theorem 5.5 by reduction to the reachability problem in pushdown automata. A first, trivial idea would be a simple replacement of the queue by a stack, i.e., from the queue's content v we reach w if, and only if, the PDA reaches w from

v. Unfortunately, this construction is not possible since our queue automaton modifies its content at both ends which cannot be simulated with a single stack. Hence, we need a more abstract presentation of the queue's contents.

To this end, we consider some non-failing computation $t \in (W\overline{R})^*$ of the queue with initial content $v \in L$, i.e., $v \circ t \neq \bot$. So, let $v_0, \ldots, v_m \in A^*$ and $\alpha_1, \ldots, \alpha_m \in \Sigma_A$ with $v_0 = v$, $v_{i+1} = v_i \circ \alpha_{i+1} \neq \bot$ for each $0 \leq i < m$, and $t = \alpha_1 \ldots \alpha_m \in (W\overline{R})^*$. By $v_{i+1} = v_i \circ \alpha_{i+1} \neq \bot$ we have $\overline{v_i}\overline{\pi_A(\alpha_{i+1})} = \pi_{\overline{A}}(\alpha_{i+1})\overline{v_{i+1}}$ for each $0 \leq i < m$. Hence, we have $\overline{v_0}\overline{\pi_A(t)} = \pi_{\overline{A}}(t)\overline{v_m}$. Since $t \in (W\overline{R})^*$ holds, we have $\pi_A(t) \in W^*$ and, therefore, $v_0\pi_A(t) \in LW^*$. Let $\mathfrak{C}$ be an NFA accepting the regular language LW^* (this is the set of all possible queue **C**ontents). Then there is an accepting run $p_0, \ldots, p_\ell$ in $\mathfrak{C}$ labeled with $v_0\pi_A(t)$.

Due to closure properties, the language $(W\overline{R})^*$ is regular. Let $\mathfrak{T}$ be an NFA accepting this language (i.e., $\mathfrak{T}$ accepts all possible **T**ransformation sequences). Then there is an accepting run $s_0, \ldots, s_m$ in $\mathfrak{T}$ labeled with $t = \alpha_1 \ldots \alpha_m$.

Now, we want to abstract any configuration (s_i, v_i) of our queue automaton with the help of the following information:

1. the state s_i from $\mathfrak{T}$ which corresponds to the control state of our queue automaton,
2. two states p_{x_i} and p_{y_i} from $\mathfrak{C}$ such that $p_{x_i}, \ldots, p_{y_i}$ is a run in $\mathfrak{C}$ labeled with v_i, and
3. the natural number $|v_i|_\$$.

Initially, we abstract (s_0, v_0) by $(p_0, p_{|v|}, s_0, 0)$ since $p_0, \ldots, p_{|v|}$ is a run in $\mathfrak{C}$ labeled with $v_0 = v$ and $|v_0|_\$ = |v|_\$ = 0$ by $v \in L \subseteq (A \setminus \{\$\})^*$. Next, we can obtain the abstraction of (s_{i+1}, v_{i+1}) from (s_i, v_i) as follows: let $(p_{x_i}, p_{y_i}, s_i, n_i)$ be the abstraction of (s_i, v_i). By the choice of our run in $\mathfrak{T}$ we have some edge $s_i \xrightarrow{\alpha_{i+1}}_{\mathfrak{T}} s_{i+1}$. Additionally, we have to distinguish the following two cases (as depicted in Fig. 1):

1. If $\alpha_{i+1} = a \in A$, we can extend the run $p_{x_i}, \ldots, p_{y_i}$ by the edge $p_{y_i} \xrightarrow{a}_{\mathfrak{C}} p_{y_i+1}$. Additionally, if $\alpha_{i+1} = \$$ then the number of \$'s in v_i will be increased. Hence, we abstract (s_{i+1}, v_{i+1}) in this case by $(p_{x_i}, p_{y_i+1}, s_{i+1}, n_i + |a|_\$)$.

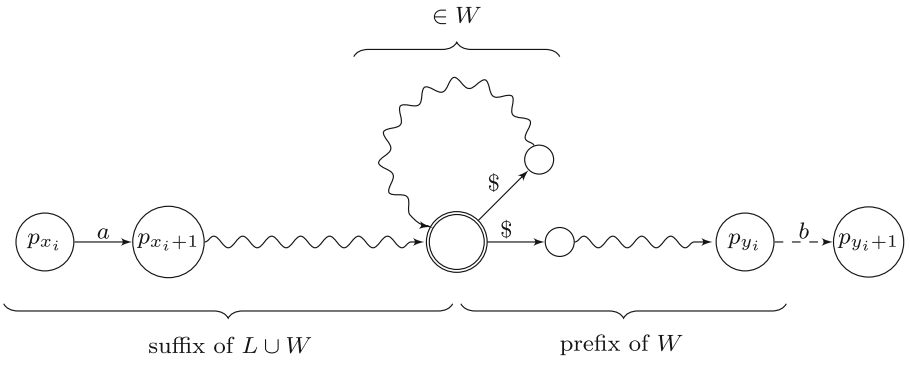

Fig. 1 A run labeled with v_i from p_{x_i} to p_{y_i} in $\mathfrak{C}$

2. If $\alpha_{i+1} = \overline{a} \in \overline{A}$, the run $p_{x_i}, \ldots, p_{y_i}$ starts with the edge $p_{x_i} \xrightarrow{a}_{\mathfrak{C}} p_{x_i+1}$. If $a = \$$ then the number of \$'s in v_i decreases. The resulting abstraction of (s_{i+1}, v_{i+1}) in this case is $(p_{x_i+1}, p_{y_i}, s_{i+1}, n_i - |a|_{\$})$.

All in all, $(p_{x_i}, p_{y_i}, s_i, n_i)$ is an abstraction of the queue automaton's configuration (s_i, v_i). These information can be simulated with the help of a pushdown automaton $\mathfrak{P}$. In this case, the control states of $\mathfrak{P}$ are composed of the states p_{x_i}, p_{y_i}, and s_i and the stack contains $\$^{n_i}$. Note that this PDA is essentially a (partially blind) one-counter automaton, but due to technical reasons we will utilize this more powerful automata model.

To this end, let $\mathfrak{C} = (Q_{\mathfrak{C}}, A, I_{\mathfrak{C}}, \Delta_{\mathfrak{C}}, F_{\mathfrak{C}})$ be an NFA accepting LW^* and $\mathfrak{T} = (Q_{\mathfrak{T}}, \Sigma_A, I_{\mathfrak{T}}, \Delta_{\mathfrak{T}}, F_{\mathfrak{T}})$ be an NFA accepting $(W\overline{R})^*$. W.l.o.g., we can assume that both, $\mathfrak{C}$ and $\mathfrak{T}$, are trim in the sense that each state is reachable from the initial state and can reach some final state. Additionally, we assume that $\mathfrak{C}$ and $\mathfrak{T}$ have exactly one final state called $f_{\mathfrak{C}}$ resp. $f_{\mathfrak{T}}$. Note that we can compute these two automata in polynomial time from NFAs accepting L, W, and R.

Recall that the queue's configuration is abstracted by states from $\mathfrak{C}$ and $\mathfrak{T}$ and by some natural number. Then the PDA $\mathfrak{P} = (Q_{\mathfrak{P}}, \Sigma_A, \{\$\}, I_{\mathfrak{P}}, \Delta_{\mathfrak{P}}, F_{\mathfrak{P}})$ is defined as follows:

- $Q_{\mathfrak{P}} := Q_{\mathfrak{C}} \times Q_{\mathfrak{C}} \times Q_{\mathfrak{T}}$. Here, the first and second component represent the two states characterizing the queue's content as described above. The third component represents the control state of the queue automaton.
- $I_{\mathfrak{P}} := I_{\mathfrak{C}} \times Q_L \times I_{\mathfrak{T}}$ where $Q_L := \{q \in Q_{\mathfrak{C}} \mid \exists v \in L \colon I_{\mathfrak{C}} \xrightarrow{v}_{\mathfrak{C}} q\}$ is the set of states being reachable via L
- $F_{\mathfrak{P}} := Q_{\mathfrak{C}} \times F_{\mathfrak{C}} \times F_{\mathfrak{T}}$
- $\Delta_{\mathfrak{P}}$ contains exactly the following transitions for $a \in A$, $p, p', q, q' \in Q_{\mathfrak{C}}$, and $s, s' \in Q_{\mathfrak{T}}$:

  (W) *Simulate writing of the letter a into the queue*:
  $((p, q, s), a, \pi_{\$}(a), (p, q', s')) \in \Delta_{\mathfrak{P}}$ if $(q, a, q') \in \Delta_{\mathfrak{C}}$ and $(s, a, s') \in \Delta_{\mathfrak{T}}$.

  (R) *Simulate reading of the letter a from the queue*:
  $((p, q, s), \overline{a}, \overline{\pi_{\$}(a)}, (p', q, s')) \in \Delta_{\mathfrak{P}}$ if $(p, a, p') \in \Delta_{\mathfrak{C}}$ and $(s, \overline{a}, s') \in \Delta_{\mathfrak{T}}$.

In other words, we have the following four cases:

1. $((p, q, s), \$^n) \xrightarrow{a}_{\mathfrak{P}} ((p, q', s'), \$^n)$ iff $a \in A \setminus \{\$\}$, $q \xrightarrow{a}_{\mathfrak{C}} q'$, and $s \xrightarrow{a}_{\mathfrak{T}} s'$.
2. $((p, q, s), \$^n) \xrightarrow{\$}_{\mathfrak{P}} ((p, q', s'), \$^{n+1})$ iff $q \xrightarrow{\$}_{\mathfrak{C}} q'$ and $s \xrightarrow{\$}_{\mathfrak{T}} s'$.
3. $((p, q, s), \$^n) \xrightarrow{\overline{a}}_{\mathfrak{P}} ((p', q, s'), \$^n)$ iff $a \in A \setminus \{\$\}$, $p \xrightarrow{a}_{\mathfrak{C}} p'$, and $s \xrightarrow{\overline{a}}_{\mathfrak{T}} s'$.
4. $((p, q, s), \$^n) \xrightarrow{\overline{\$}}_{\mathfrak{P}} ((p', q', s'), \$^{n-1})$ iff $n > 0$, $p \xrightarrow{\$}_{\mathfrak{C}} p'$, and $s \xrightarrow{\overline{\$}}_{\mathfrak{T}} s'$.

Now, we assign to the configuration $((p, q, s), \$^n)$ the set of all words being the labeling of a run from p to q in $\mathfrak{C}$ and containing n appearances of the letter \$ (which

marks the beginning of a word from W). Formally, our assignment is the mapping $[\![.]\!]\colon \mathrm{Conf}_{\mathfrak{P}} \to 2^{A^*}$ with

$$[\![((p,q,s),\$^n)]\!] = L(\mathfrak{C}_{p\to q}) \cap (\$^n \sqcup\!\sqcup (A \setminus \{\$\})^*)$$

for each $p, q \in Q_{\mathfrak{C}}$, $s \in Q_{\mathfrak{T}}$, and $n \in \mathbb{N}$. This language represents the set of all configurations (s, v) whose abstraction (as explained above) is $((p, q, s), n)$.

Next, we prove that the set of reachable queue contents coincides with this semantics of the reachable, accepting configurations of the PDA $\mathfrak{P}$.

Proposition 5.6 *We have* $\textsc{Reach}(L, (W\overline{R})^*) = \bigcup_{\sigma \in post^*(Init_{\mathfrak{P}}) \cap Final_{\mathfrak{P}}} [\![\sigma]\!]$.

With the help of Proposition 5.6 we are able to prove Theorem 5.5. So, we will first prove this theorem and afterwards we show the correctness of the PDA $\mathfrak{P}$ and its semantics.

Proof (of Theorem 5.5) Due to Theorem 3.6 we can compute NFAs describing the set of configurations $\mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}})$ in polynomial time. So, for $(p, q, s) \in Q_{\mathfrak{P}}$ let $\mathfrak{A}_{(p,q,s)}$ be an NFA such that

$$\mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}}) = \bigcup_{(p,q,s)\in Q_{\mathfrak{P}}} \{(p,q,s)\} \times L(\mathfrak{A}_{(p,q,s)})$$

holds. Since, \$ is the only stack symbol in $\mathfrak{P}$, we get $L(\mathfrak{A}_{(p,q,s)}) \subseteq \$^*$. Furthermore, $\$^n \in L(\mathfrak{A}_{(p,q,s)})$ holds if, and only if, $((p, q, s), \$^n) \in \mathrm{Conf}_{\mathfrak{P}}$ is reachable from an initial configuration of $\mathfrak{P}$.

The following language is regular as well:

$$K := \bigcup_{(p,q,s)\in F_{\mathfrak{P}}} \Big(L(\mathfrak{C}_{p\to q}) \cap \big(L(\mathfrak{A}_{(p,q,s)}) \sqcup\!\sqcup (A \setminus \{\$\})^* \big) \Big).$$

Later we will see $K = \textsc{Reach}(L, (W\overline{R})^*)$. But before, we want to give some intuition on the definition of K. This language contains all words $v \in A^*$ such that

- v is the label of a run in $\mathfrak{C}$ from p to q, where q is accepting in $\mathfrak{C}$ (note that p is not necessarily initial),
- v contains n \$'s (for an $n \in \mathbb{N}$), and
- the configuration $((p, q, s), \$^n)$ is reachable in $\mathfrak{P}$ from some initial configuration.

Since each intermediate step of our computation is possible in polynomial time, we can compute an NFA accepting K in polynomial time as well.

Finally, we prove $K = \textsc{Reach}(L, (W\overline{R})^*)$. Applying Proposition 5.6 it suffices to prove $K = \bigcup_{\sigma \in \mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}} [\![\sigma]\!]$.

First, let $v \in [\![\sigma]\!]$ for some $\sigma = ((p, q, s), \$^n) \in \mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$. Then we have $(p, q, s) \in F_{\mathfrak{P}}$ and $\$^n \in L(\mathfrak{A}_{(p,q,s)})$. Hence, we have

$$\begin{aligned} v \in [\![\sigma]\!] &= L(\mathfrak{C}_{p\to q}) \cap \left(\$^n \sqcup\!\sqcup\ (A \setminus \{\$\})^*\right) \\ &\subseteq L(\mathfrak{C}_{p\to q}) \cap \left(L(\mathfrak{A}_{(p,q,s)}) \sqcup\!\sqcup\ (A \setminus \{\$\})^*\right) \subseteq K\,. \end{aligned}$$

Conversely, let $v \in K$. Then there is $(p, q, s) \in F_{\mathfrak{P}}$ with

$$v \in \left(L(\mathfrak{C}_{p\to q}) \cap \left(L(\mathfrak{A}_{(p,q,s)}) \sqcup\!\sqcup\ (A \setminus \{\$\})^*\right)\right).$$

Set $n := |v|_{\$}$. Then we have $\$^n \in L(\mathfrak{A}_{(p,q,s)})$ implying that $((p, q, s), \$^n)$ is reachable from an initial configuration of $\mathfrak{P}$. Additionally, this configuration is final since $(p, q, s) \in F_{\mathfrak{P}}$. In other words, we have $((p, q, s), \$^n) \in \mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$. Finally, we have

$$v \in L(\mathfrak{C}_{p\to q}) \cap \left(\$^n \sqcup\!\sqcup\ (A \setminus \{\$\})^*\right) = [\![((p, q, s), \$^n)]\!]\,. \qquad \square$$

5.2 The Correctness of the Proposition 5.6

Next, we want to prove the correctness of Proposition 5.6. We prove this with the help of two lemmas each proving one inclusion. First, we show that each reachable queue content belongs to the semantics of some reachable configuration of $\mathfrak{P}$. To this end, we consider some $v \in L$ and $t \in (W\overline{R})^*$ with $v \circ t \neq \bot$. We construct a t-labeled run of $\mathfrak{P}$ such that the i^{th} intermediate result v_i is covered by the semantics of the i^{th} step in our constructed run.

Concretely, we have runs $p_0, \ldots, p_\ell$ and $s_0, \ldots, s_m$ in $\mathfrak{C}$ and $\mathfrak{T}$ labeled with $v\pi_A(t)$ and t, respectively, from an initial state to an accepting state. The i^{th} configuration σ_i on our run of $\mathfrak{P}$ consists of two states p_{x_i} and p_{y_i} (where $0 \le x_i \le y_i \le \ell$), s_i, and the number of \$'s on the sub-path $p_{x_i}, \ldots, p_{y_i}$. Then we will see that $v_i \in [\![\sigma_i]\!]$, which finally implies $v_m = v \circ t \in [\![\sigma_m]\!]$.

Example 5.7 Consider $L = \{\varepsilon\}$, $W = \{\$a, \$b\}$, and $R = \$^* \sqcup\!\sqcup\ b = \$^*b\$^*$. Then the languages LW^* and $(W\overline{R})^*$ are accepted by the NFAs $\mathfrak{C}$ and $\mathfrak{T}$ in Fig. 2. Let $t := \$b\$\overline{b} \in (W\overline{R})^*$. Then we have

$$\varepsilon \circ \$b\$\overline{b} = \$ \circ b\$\overline{b} = \$b \circ \$\overline{b} = b \circ \overline{b} = \varepsilon \neq \bot\,.$$

Consider the accepting runs $p_1 \xrightarrow{\$}_{\mathfrak{C}} p_2 \xrightarrow{b}_{\mathfrak{C}} p_1$ and $s_1 \xrightarrow{\$}_{\mathfrak{T}} s_2 \xrightarrow{b}_{\mathfrak{T}} s_3 \xrightarrow{\overline{\$}}_{\mathfrak{T}} s_3 \xrightarrow{\overline{b}}_{\mathfrak{T}} s_4$ in $\mathfrak{C}$ and $\mathfrak{T}$ labeled with $v\pi_A(t) = \$b$ and t, respectively. Then we construct the following run in $\mathfrak{P}$:

$$\begin{aligned} \mathrm{Init}_{\mathfrak{P}} \ni ((p_1, p_1, s_1), 0) &\xrightarrow{\$}_{\mathfrak{P}} ((p_1, p_2, s_2), 1) \xrightarrow{b}_{\mathfrak{P}} ((p_1, p_1, s_3), 1) \\ &\xrightarrow{\overline{\$}}_{\mathfrak{P}} ((p_2, p_1, s_3), 0) \xrightarrow{\overline{b}}_{\mathfrak{P}} ((p_1, p_1, s_4), 0) \in \mathrm{Final}_{\mathfrak{P}}\,. \end{aligned}$$

Then we can see $v \circ t = \varepsilon \in [\![((p_1, p_1, s_4), 0)]\!]$.

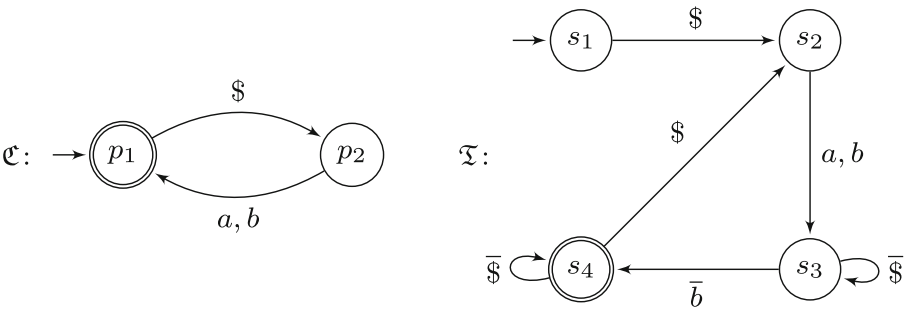

Fig. 2 NFAs $\mathfrak{C}$ and $\mathfrak{T}$ accepting LW^* and $(W\overline{R})^*$, respectively, from Example 5.7

Lemma 5.8 *Let $t \in (W\overline{R})^*$ and $v \in L$ with $v \circ t \neq \bot$. Then there is $\sigma \in post^*(Init_{\mathfrak{P}}) \cap Final_{\mathfrak{P}}$ with $(v \circ t) \in [\![\sigma]\!]$.*

Proof Let $t = w_1\overline{r_1} \dots w_k\overline{r_k}$ with $w_1, \dots, w_k \in W$ and $r_1, \dots, r_k \in R$. We have $vw_1 \dots w_k \in LW^* = L(\mathfrak{C})$. Hence, there is a run $p_0, \dots, p_\ell$ labeled with $vw_1 \dots w_k$ in $\mathfrak{C}$ with $p_0 \in I_{\mathfrak{C}}$ and $p_\ell \in F_{\mathfrak{C}}$. Additionally, we have $t \in (W\overline{R})^* = L(\mathfrak{T})$ and therefore a run $s_0, \dots, s_m$ with labeling t in $\mathfrak{T}$ with $s_0 \in I_{\mathfrak{T}}$ and $s_m \in F_{\mathfrak{T}}$.

By $t \in \Sigma_A^*$ there are $\alpha_0, \dots, \alpha_{m-1} \in \Sigma_A$ with $t = \alpha_0 \dots \alpha_{m-1}$. Since $v \circ t \neq \bot$ there are $v_0, \dots, v_m \in A^*$ with $v_0 = v$ and $v_{i+1} = v_i \circ \alpha_i$ for each $0 \leq i < m$. This implies $v_i = v \circ \alpha_0 \dots \alpha_{i-1}$ and, hence, $\overline{v\pi_A(\alpha_0 \dots \alpha_{i-1})} = \pi_{\overline{A}}(\alpha_0 \dots \alpha_{i-1})\overline{v_i}$. Since $v\pi_A(\alpha_0 \dots \alpha_{i-1})$ is a prefix of $vw_1 \dots w_k$ we infer that v_i is a factor of the word $vw_1 \dots w_k$. Therefore, v_i is the labeling of some fragment of the run $p_0, \dots, p_\ell$ in $\mathfrak{C}$.

Now, we want to construct a run $\sigma_0, \dots, \sigma_m$ in $\mathfrak{P}$ from an initial configuration to a final configuration with labeling t. To this end, we define

- $x_i := |\alpha_0 \dots \alpha_{i-1}|_{\overline{A}}$,
- $y_i := |v| + |\alpha_0 \dots \alpha_{i-1}|_A$, and
- $n_i := |v_i|_{\$}$.

By definition we have $0 \leq x_i \leq |t|_{\overline{A}} \leq m$ and $|v| \leq y_i \leq |vt|_A = \ell$ for each $0 \leq i \leq m$. Set $\sigma_i := ((p_{x_i}, p_{y_i}, s_i), \$^{n_i}) \in \mathrm{Conf}_{\mathfrak{P}}$ for each $0 \leq i \leq m$. We will prove that $\sigma_0, \dots, \sigma_m$ is a run in $\mathfrak{P}$ with labeling t from $\mathrm{Init}_{\mathfrak{P}}$ to $\mathrm{Final}_{\mathfrak{P}}$ such that $v_i \in [\![\sigma_i]\!]$. But first, we have to show $n_i = |\alpha_0 \dots \alpha_{i-1}|_{\$} - |\alpha_0 \dots \alpha_{i-1}|_{\overline{\$}}$ for each $0 \leq i \leq m$. We do this by induction on i. The case $i = 0$ is obvious since $n_0 = 0$ by $v_0 = v \in L \subseteq (A \setminus \{\$\})^*$ and $\alpha_0 \dots \alpha_{i-1} = \varepsilon$. Now, let $i \geq 0$. The induction hypothesis holds for i and we prove the equation for $i + 1$. Then we have to consider three cases:

1. $\alpha_i \notin \{\$, \overline{\$}\}$. Then we have

$$\begin{aligned} n_{i+1} = |v_{i+1}|_{\$} = |v_i|_{\$} = n_i &= |\alpha_0 \dots \alpha_{i-1}|_{\$} - |\alpha_0 \dots \alpha_{i-1}|_{\overline{\$}} \\ &= |\alpha_0 \dots \alpha_i|_{\$} - |\alpha_0 \dots \alpha_i|_{\overline{\$}}. \end{aligned}$$

2. $\alpha_i = \$$. Then we have

$$\begin{aligned} n_{i+1} &= |v_{i+1}|_{\$} = |v_i|_{\$} + 1 = n_i + 1 = |\alpha_0 \dots \alpha_{i-1}|_{\$} - |\alpha_0 \dots \alpha_{i-1}|_{\overline{\$}} + 1 \\ &= |\alpha_0 \dots \alpha_{i-1}\$|_{\$} - |\alpha_0 \dots \alpha_{i-1}\$|_{\overline{\$}} = |\alpha_0 \dots \alpha_i|_{\$} - |\alpha_0 \dots \alpha_i|_{\overline{\$}}. \end{aligned}$$

3. $\alpha_i = \overline{\$}$. Then, by $v_{i+1} = v_i \circ \overline{\$} \neq \bot$ we have $v_i = \$v_{i+1}$. Hence, we have

$$\begin{aligned} n_{i+1} &= |v_{i+1}|_{\$} = |v_i|_{\$} - 1 = n_i - 1 = |\alpha_0 \dots \alpha_{i-1}|_{\$} - |\alpha_0 \dots \alpha_{i-1}|_{\overline{\$}} - 1 \\ &= |\alpha_0 \dots \alpha_{i-1}\overline{\$}|_{\$} - |\alpha_0 \dots \alpha_{i-1}\overline{\$}|_{\overline{\$}} = |\alpha_0 \dots \alpha_i|_{\$} - |\alpha_0 \dots \alpha_i|_{\overline{\$}}. \end{aligned}$$

To prove that $\sigma_m \in \mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ and $v \circ t \in [\![\sigma_m]\!]$ we demonstrate that $\sigma_0 \in \mathrm{Init}_{\mathfrak{P}}$, $\sigma_i \xrightarrow{\alpha_i}_{\mathfrak{P}} \sigma_{i+1}$, $v_{i+1} \in [\![\sigma_{i+1}]\!]$, and $\sigma_m \in \mathrm{Final}_{\mathfrak{P}}$. This is done by induction on i.

First, we show $\sigma_0 \in \mathrm{Init}_{\mathfrak{P}}$. By definition, we have $x_0 = n_0 = 0$ and $y_0 = |v|$. Due to the choice of the run $p_0, \dots, p_\ell$ we have $p_0 \in I_{\mathfrak{C}}$ and $p_0 \xrightarrow{v}_{\mathfrak{C}} p_{|v|}$ and therefore $p_{|v|} \in \{q \in Q_{\mathfrak{C}} \mid \exists u \in L \colon I_{\mathfrak{C}} \xrightarrow{u}_{\mathfrak{C}} q\} = Q_L$. Additionally, by the choice of $s_0, \dots, s_m$ we have $s_0 \in I_{\mathfrak{T}}$. Hence, $\sigma_0 = ((p_0, p_{|v|}, s_0), \varepsilon) \in \mathrm{Init}_{\mathfrak{P}}$. By $v \in L \subseteq (A \setminus \{\$\})^*$ we can also infer $v \in L(\mathfrak{C}_{p_0 \to p_{|v|}}) \cap (\$^0 \sqcup\!\sqcup (A \setminus \{\$\})^*) = [\![\sigma_0]\!]$.

Next, let $i \geq 0$. We have to consider two cases:

(W) $\alpha_i \in A$. Then we have $x_{i+1} = x_i$, $y_{i+1} = y_i + 1$, and $n_{i+1} = n_i + |\alpha_i|_{\$}$. By the choice of the run $p_0, \dots, p_\ell$ we have $(p_{y_i}, \alpha_i, p_{y_{i+1}}) \in \Delta_{\mathfrak{C}}$ and by choice of $s_0, \dots, s_m$ we have $(s_i, \alpha_i, s_{i+1}) \in \Delta_{\mathfrak{T}}$. Hence, there is a transition

$$((p_{x_i}, p_{y_i}, s_i), \alpha_i, \pi_{\$}(\alpha_i), (p_{x_{i+1}}, p_{y_{i+1}}, s_{i+1})) \in \Delta_{\mathfrak{P}}$$

and, therefore, $\sigma_i \xrightarrow{\alpha_i}_{\mathfrak{P}} \sigma_{i+1}$. Furthermore, we have

$$\begin{aligned} & v_{i+1} = v_i \circ \alpha_i = v_i \alpha_i \overset{\text{i.h.}}{\in} [\![\sigma_i]\!] \cdot \alpha_i \\ = {} & \big(L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) \cap (\$^{n_i} \sqcup\!\sqcup (A \setminus \{\$\})^*)\big) \cdot \alpha_i \\ \subseteq {} & \big(L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) \cap (\$^{n_i} \sqcup\!\sqcup (A \setminus \{\$\})^*)\big)\big(L(\mathfrak{C}_{p_{y_i} \to p_{y_{i+1}}}) \cap (\pi_{\$}(\alpha_i) \sqcup\!\sqcup (A \setminus \{\$\})^*)\big) \\ \subseteq {} & \big(L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) L(\mathfrak{C}_{p_{y_i} \to p_{y_{i+1}}})\big) \cap \big(\$^{n_i} \pi_{\$}(\alpha_i) \sqcup\!\sqcup (A \setminus \{\$\})^*\big) \\ \subseteq {} & L(\mathfrak{C}_{p_{x_i} \to p_{y_{i+1}}}) \cap (\$^{n_i + |\alpha_i|_{\$}} \sqcup\!\sqcup (A \setminus \{\$\})^*) = [\![\sigma_{i+1}]\!]. \end{aligned}$$

(R) $\alpha_i = \overline{a} \in \overline{A}$. Here, we have $x_{i+1} = x_i + 1$, $y_{i+1} = y_i$, and $n_{i+1} = n_i - |a|_{\$} \geq 0$. Due to $v_{i+1} = v_i \circ \overline{a} \neq \bot$ we have $v_i = a v_{i+1}$. Since $p_{x_i}, \dots, p_{y_i}$ is a run labeled with v_i and a is a prefix of v_i, this run begins with an a-edge. This implies $(p_{x_i}, a, p_{x_{i+1}}) \in \Delta_{\mathfrak{C}}$. Additionally, since $s_0, \dots, s_m$ is a run labeled with t, we have $(s_i, \alpha_i, s_{i+1}) \in \Delta_{\mathfrak{T}}$. Hence, we have

$$((p_{x_i}, p_{y_i}, s_i), \overline{a}, \overline{\pi_{\$}(a)}, (p_{x_{i+1}}, p_{y_{i+1}}, s_{i+1})) \in \Delta_{\mathfrak{P}}$$

implying $\sigma_i \xrightarrow{\alpha_i}_{\mathfrak{P}} \sigma_{i+1}$. By the induction hypothesis we have $v_i \in [\![\sigma_i]\!] = L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) \cap (\$^{n_i} \sqcup\!\sqcup (A \setminus \{\$\})^*)$. Since $v_i = a v_{i+1}$ and $(p_{x_i}, a, p_{x_{i+1}}) \in$

$\Delta_{\mathfrak{C}}$ we know that

$$v_i \in a \cdot \left(L(\mathfrak{C}_{p_{x_{i+1}} \to p_{y_i}}) \cap (\$^{n_i - |a|_\$} \sqcup\!\sqcup (A \setminus \{\$\})^*)\right) = a \cdot [\![\sigma_{i+1}]\!]$$

which implies $v_{i+1} = a \backslash v_i \in [\![\sigma_{i+1}]\!]$ (recall that $v_{i+1} = a v_i$ holds).

Finally, we have $y_m = \ell$ implying $p_{y_m} = p_\ell \in F_{\mathfrak{C}}$ and $s_m \in F_{\mathfrak{T}}$. Hence, we have $\sigma_m \in \mathrm{Final}_{\mathfrak{P}}$ which finishes our proof. □

Let $v \in L$ and $t \in (W\overline{R})^*$ with $v \circ t \neq \bot$. Recall that we have proven Lemma 5.8 by combining runs in $\mathfrak{C}$ and $\mathfrak{T}$ to a t-labeled run in $\mathfrak{P}$ which simulates the computation $v \circ t$. A first approach to prove the converse inclusion would be the following: let $\sigma \in \mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ and $w \in [\![\sigma]\!]$. Then there is a run in $\mathfrak{P}$ from $\mathrm{Init}_{\mathfrak{P}}$ to σ labeled with $t \in \Sigma_A^*$. From this run we obtain an accepting run in $\mathfrak{T}$ labeled with t implying $t \in L(\mathfrak{T}) = (W\overline{R})^*$. Unfortunately, we cannot infer $v \circ t \neq \bot$ as the following example proves:

Example 5.9 We continue Example 5.7. Consider the following accepting run of $\mathfrak{P}$:

$$\begin{aligned}\mathrm{Init}_{\mathfrak{P}} \ni ((p_1, p_1, s_1), 0) &\xrightarrow{\$}_{\mathfrak{P}} ((p_1, p_2, s_2), 1) \xrightarrow{a}_{\mathfrak{P}} ((p_1, p_1, s_3), 1) \\ &\xrightarrow{\overline{\$}}_{\mathfrak{P}} ((p_2, p_1, s_3), 0) \xrightarrow{\overline{b}}_{\mathfrak{P}} ((p_1, p_1, s_4), 0) =: \sigma \in \mathrm{Final}_{\mathfrak{P}}\,.\end{aligned}$$

Then we have $[\![\sigma]\!] = \{\varepsilon\}$ and, indeed, $\varepsilon \in \textsc{Reach}(L, (W\overline{R})^*)$. However, $t = \$a\overline{\$}\overline{b}$ is not an allowed computation of our queue automaton since

$$\varepsilon \circ \$a\overline{\$}\overline{b} = \$ \circ a\overline{\$}\overline{b} = \$a \circ \overline{\$}\overline{b} = a \circ \overline{b} = \bot\,.$$

The reason of this problem is the lack of memory of our pushdown automaton $\mathfrak{P}$ which allows that the subsequences of write and read actions, respectively, do not match. However, we can avoid this problem by a modification of write actions in our run t. Since the application of a read action in $\mathfrak{P}$ always requires a step in the NFA $\mathfrak{C}$, we can obtain a transformation sequence $t' \in (W\overline{R})^*$ in which we only read letters that have been written into the queue before. This will finally result in $w = v \circ t' \in \textsc{Reach}(L, (W\overline{R})^*)$.

Lemma 5.10 *Let $\sigma \in post^*(Init_{\mathfrak{P}}) \cap Final_{\mathfrak{P}}$. Then we have*

$$[\![\sigma]\!] \subseteq \textsc{Reach}(L, (W\overline{R})^*)\,.$$

Proof Let $\sigma = ((p, q, s), \$^n) \in \mathrm{post}^*(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ and $v \in [\![\sigma]\!]$. Since $v \in [\![\sigma]\!] = L(\mathfrak{C}_{p \to q}) \cap (\$^n \sqcup\!\sqcup (A \setminus \{\$\})^*)$, $L(\mathfrak{C}) = LW^*$, and $q \in F_{\mathfrak{C}}$ there are a suffix y of a word from $L \cup W$ and $\$z_1, \ldots, \$z_n \in W$ such that $v = y\$z_1\$z_2 \ldots \$z_n$. By $v \in [\![\sigma]\!]$ we have $y \in L(\mathfrak{C}_{p \to f_{\mathfrak{C}}}) \cap (A \setminus \{\$\})^*$. Furthermore, there is some word $t \in \Sigma_A^*$ labeling a path from $\mathrm{Init}_{\mathfrak{P}}$ to σ. Since every transition of the PDA $\mathfrak{P}$ simulates a transition of the NFA $\mathfrak{T}$, we obtain $t \in L(\mathfrak{T}) = (W\overline{R})^*$. Hence, there are $k \geq 0$, $\$w_1, \ldots, \$w_k \in W$, and $r_1, \ldots, r_k \in R$ with $t = \$w_1\overline{r_1} \ldots \$w_k\overline{r_k}$.

The PDA $\mathfrak{P}$ lacks a memory of the concrete paths in $\mathfrak{C}$ and $\mathfrak{T}$ and, hence, lacks a memory of the letters that have been written into the queue before. Therefore, it is possible that the transformation t cannot be applied to any word from L (i.e., $L \circ t = \{\bot\}$). But due to this lack of memory we can replace the infixes $\$w_1, \ldots, \$w_k$ in t by arbitrary words from W. Hence, we construct a new transformation $t' \in L(\mathfrak{T}) = (W\overline{R})^*$ which is a labeling of some other path in $\mathfrak{P}$ from $\mathrm{Init}_{\mathfrak{P}}$ to σ which corresponds to some valid computation of the queue automaton (i.e., $L \circ t' \neq \{\bot\}$).

Recall that \$-transitions in $\mathfrak{P}$ increase the number of \$s in the stack and $\overline{\$}$-transitions in $\mathfrak{P}$ decrease the number of \$s in the stack. Therefore, since t is some labeling of a path in $\mathfrak{P}$ each prefix of t contains at least as many \$s as $\overline{\$}$. Hence, we have $|r_1 \ldots r_i|_{\$} \leq |\$w_1 \ldots \$w_i|_{\$} = i$ for each $1 \leq i \leq k$.

Due to $r_1, \ldots, r_k \in R \subseteq A^*$ there is $\ell \in \mathbb{N}$ and words $x_0, \ldots, x_\ell \in (A \setminus \{\$\})^*$ with $r_1 \ldots r_k = x_0\$ \ldots \$x_\ell$. Hence, since $|r_1 \ldots r_i|_{\$} \leq i$ we know that $r_1 \ldots r_i$ is a prefix of $x_0\$ \ldots \$x_i$ for each $1 \leq i \leq \ell$. In particular, we have $k = |t|_{\$}$ and $\ell = |t|_{\overline{\$}}$ implying $k - \ell = n$.

Now, we distinguish two cases: first, suppose $\ell = 0$, i.e., $k = n$ and $x_0 = r_1 \ldots r_k \in A^*$. Therefore, a path in $\mathfrak{P}$ with labeling t from $\mathrm{Init}_{\mathfrak{P}}$ to σ requires some \$-free path in $\mathfrak{C}$ labeled with $r_1 \ldots r_k$ from $I_{\mathfrak{C}}$ to $p \in Q_{\mathfrak{C}}$ (this path is represented in the first component of $\mathfrak{P}$'s states). Due to $L(\mathfrak{C}) = LW^*$, $L \subseteq (A \setminus \{\$\})^*$, and $W \subseteq \$(A \setminus \{\$\})^*$, the word $r_1 \ldots r_k$ is some prefix of a word from L. Hence, we have $x_0 \in L(\mathfrak{C}_{I_{\mathfrak{C}} \to p}) \cap (A \setminus \{\$\})^*$ and $y \in L(\mathfrak{C}_{p \to f_{\mathfrak{C}}}) \cap (A \setminus \{\$\})^*$ implying $x_0 y \in L$. Now, we prove

$$v = x_0 y \circ \$z_1\overline{r_1}\$ \ldots \$z_k\overline{r_k} \in \textsc{Reach}(L, (W\overline{R})^*) .$$

Then we can prove the following

$$\begin{aligned} x_0 y \circ \$z_1\overline{r_1}\$ \ldots \$z_k\overline{r_k} &= r_1 \ldots r_k y \circ \$z_1\overline{r_1}\$ \ldots \$z_k\overline{r_k} \\ &= r_2 \ldots r_k y \$z_1 \circ \$z_2\overline{r_2}\$ \ldots \$z_k\overline{r_k} \\ &\vdots \\ &= r_{i+1} \ldots r_k y \$z_1 \ldots \$z_i \circ \$z_{i+1}\overline{r_{i+1}}\$ \ldots \$z_k\overline{r_k} \\ &\vdots \\ &= y\$z_1 \ldots \$z_k = v . \end{aligned}$$

Since $\$z_1, \ldots, \$z_k \in W$ and $r_1, \ldots, r_k \in R$ we have $\$z_1\overline{r_1}\$ \ldots \$z_k\overline{r_k} \in (W\overline{R})^*$. Then, from $x_0 y \in L$ we can infer $v \in \textsc{Reach}(L, (W\overline{R})^*)$.

Next, we assume $\ell \geq 1$. Since t is the labeling of some path from $\mathrm{Init}_{\mathfrak{P}}$ to σ in $\mathfrak{P}$ we can prove (by observing the first component of $\mathfrak{P}$'s state) that there is a path in $\mathfrak{C}$ from $I_{\mathfrak{C}}$ to p labeled with $r_1 \ldots r_k = x_0\$ \ldots \$x_\ell$. By definition of $L(\mathfrak{C}) = LW^*$, $L \subseteq (A \setminus \{\$\})^*$, and $W \subseteq \$(A \setminus \{\$\})^*$ we have $x_0 \in L$, $\$x_1, \ldots, \$x_{\ell-1} \in W$, and $\$x_\ell$ is a prefix of a word in W.

Since $x_0\$ \ldots \$x_\ell \in L(\mathfrak{C}_{I_{\mathfrak{C}} \to p})$ and $v = y\$z_1\$ \ldots \$z_n \in L(\mathfrak{C}_{p \to F_{\mathfrak{C}}}) \subseteq [\![\sigma]\!]$ we have

$$x_0\$ \ldots \$x_\ell y \$z_1\$ \ldots \$z_n \in L(\mathfrak{C}) = LW^* ,$$

i.e., $\$x_\ell y \in W$. We want to prove now

$$v = x_0 \circ \$x_1\overline{r_1}\$x_2\overline{r_2}\ldots\$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k} \in \text{REACH}(L, (W\overline{R})^*)\,.$$

First, we prove by induction on $1 \leq i < \ell$ that

$$x_0 \circ \$x_1\overline{r_1}\$x_2\overline{r_2}\ldots\$x_i\overline{r_i} = r_1\ldots r_i\backslash(x_0\$\ldots\$x_i)$$

holds. To this end, let $i = 1$. Then we have

$$x_0 \circ \$x_1\overline{r_1} = x_0\$x_1 \circ \overline{r_1} = r_1\backslash(x_0\$x_1)$$

which is defined since r_1 is a prefix of $x_0\$x_1$ as mentioned above. Now, let $1 < i < \ell$. Then we have

$$\begin{aligned}
x_0 \circ \$x_1\overline{r_1}\$x_2\overline{r_2}\ldots\$x_i\overline{r_i} &= r_1\ldots r_{i-1}\backslash(x_0\$\ldots\$x_{i-1}) \circ \$x_i\overline{r_i} \qquad \text{(by ind. hyp.)}\\
&= r_1\ldots r_{i-1}\backslash(x_0\$\ldots\$x_{i-1}) \cdot \$x_i \circ \overline{r_i}\\
&= r_1\ldots r_{i-1}\backslash(x_0\$\ldots\$x_{i-1}\$x_i) \circ \overline{r_i}\\
&= r_i\backslash\big(r_1\ldots r_{i-1}\backslash(x_0\$\ldots\$x_i)\big)\\
&= r_1\ldots r_i\backslash(x_0\$\ldots\$x_i)\,.
\end{aligned}$$

The last two equations hold since $r_1\ldots r_i$ is a prefix of $x_0\$\ldots\$x_i$ as we have mentioned above. Next, we can prove the following equalities:

$$\begin{aligned}
&x_0 \circ \$x_1\overline{r_1}\$x_2\overline{r_2}\ldots\$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k}\\
&= r_1\ldots r_{\ell-1}\backslash(x_0\$\ldots\$x_{\ell-1}) \circ \$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k}\\
&= r_1\ldots r_{\ell-1}\backslash(x_0\$\ldots\$x_{\ell-1}\$x_\ell y) \circ \overline{r_\ell}\$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k}\\
&= r_1\ldots r_{\ell-1}\backslash(r_1\ldots r_k y) \circ \overline{r_\ell}\$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k}\\
&= r_\ell\ldots r_k y \circ \overline{r_\ell}\$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k}\\
&= r_{\ell+1}\ldots r_k y \circ \$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k}\\
&= r_{\ell+2}\ldots r_k y\$z_1 \circ \$z_2\overline{r_{\ell+2}}\ldots\$z_n\overline{r_k}\\
&\vdots\\
&= y\$z_1\$\ldots\$z_n = v
\end{aligned}$$

From $\$x_1, \ldots, \$x_\ell y, \$z_1, \ldots, \$z_n \in W$ and $r_1, \ldots, r_k \in R$ we can infer

$$\$x_1\overline{r_1}\$x_2\overline{r_2}\ldots\$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\ldots\$z_n\overline{r_k} \in (W\overline{R})^*\,.$$

With $x_0 \in L$ we can finally infer $v \in \text{REACH}(L, (W\overline{R})^*)$. □

5.3 The Main Result

Until now we have seen the effective preservation of regularity if the read-write independent set $T \subseteq \Sigma_A^*$ satisfies a special condition, namely, $T = W\overline{R}$ where $W \subseteq \$(A\backslash\{\$\})^*$ and $R = \$^* \sqcup\!\sqcup\ R$. From this special case we infer now the effective preservation of regularity for arbitrary read-write independent sets.

Theorem 5.11 (Main Theorem) *Let A be an alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma_A^*$ be read-write independent and regular. Then* $\text{REACH}(L, T^*)$ *and*

BACKREACH(L, T^*) *are effectively regular. In particular, from NFAs accepting* L *and* T *we can compute NFAs accepting* REACH(L, T^*) *and* BACKREACH(L, T^*) *in polynomial time.*

We first consider the effective and efficient regularity of REACH(L, T^*). Recall that we are able to de-shuffle T by Lemma 5.4, i.e., we have

$$\text{REACH}(L, T) = \text{REACH}(L, \pi_A(T)\pi_{\overline{A}}(T)) .$$

So, when computing REACH(L, T^*) it suffices to only consider the de-shuffled words in T. The following lemma states that we are allowed to insert the synchronizing letter \$ into our de-shuffled words:

Lemma 5.12 *Let* A *be an alphabet,* $\$ \notin A$ *be another symbol, and* $L, W, R \subseteq A^*$. *Set* $W' := \$W$ *and* $R' := R \sqcup\!\sqcup \$^*$. *Then we have* $\text{REACH}(L, W\overline{R}) = \pi_A(\text{REACH}(L, W'\overline{R'}))$.

Proof First, let $x \in \text{REACH}(L, W\overline{R})$. Then there are $v \in L$, $w \in W$, and $r \in R$ with $v \circ w\overline{r} = x \neq \bot$. Due to Definition 3.1 we have $rx = vw$. We can construct $r' \in \{r\} \sqcup\!\sqcup \$^* \subseteq R \sqcup\!\sqcup \$^* = R'$ and $x' \in \{x\} \sqcup\!\sqcup \$^*$ satisfying $r'x' = v\$w$, i.e., we have $x' = r'\backslash v\$w$. Hence, the following holds:

$$\bot \neq x' = r'\backslash v\$w = v\$w \circ \overline{r'} = v \circ \$w\overline{r'} \in \text{REACH}(L, W'\overline{R'})$$

implying $x = \pi_A(x') \in \pi_A(\text{REACH}(L, W'\overline{R'}))$.

Now, let $x \in \pi_A(\text{REACH}(L, W'\overline{R'}))$. Then there is $x' \in \text{REACH}(L, W'\overline{R'})$ with $x = \pi_A(x')$, i.e., we have $v \in L$, $w' \in W'$, and $r' \in R'$ with $v \circ w'\overline{r'} = x' \neq \bot$. Again, by Definition 3.1 we have $r'x' = vw'$. Since π_A is a homomorphism, we can infer $\pi_A(r')\pi_A(x') = \pi_A(v)\pi_A(w')$ and, therefore, $\pi_A(x') = \pi_A(r')\backslash\pi_A(v)\pi_A(w')$. Hence, the following equations hold:

$$\bot \neq x = \pi_A(x') = \pi_A(r')\backslash\pi_A(v)\pi_A(w') = \pi_A(v) \circ \pi_A(w')\overline{\pi_A(r')} .$$

Since $\pi_A(v) = v, \pi_A(w') \in \pi_A(W') = W$, and $\pi_A(r') \in \pi_A(R') = R$ we can finally infer $x \in \text{REACH}(L, W\overline{R})$. □

Now we can prove our main theorem in this section:

Proof (*of Theorem 5.11*) Let $W := \pi_A(T)$ and $R := \pi_{\overline{A}}(T)$ which are both regular by closure properties of regular languages. We introduce a new letter $\$ \notin A$. Then we can compute NFAs accepting $W' := \$W$ and $R' := R \sqcup\!\sqcup \$^*$. By Theorem 5.5 we know that REACH$(L, (W'\overline{R'})^*)$ is effectively regular as well. By iterated application of the Lemmas 5.4 and 5.12 we can infer that

$$\text{REACH}(L, T^*) = \text{REACH}(L, (W\overline{R})^*) = \pi_A(\text{REACH}(L, (W'\overline{R'})^*))$$

holds. Hence, due to the closure properties of the class of regular languages, REACH(L, T^*) is effectively regular. Note that the modifications of W and R as well as the projection to A are possible in linear time and space. Hence, an NFA accepting REACH(L, T^*) can be computed still in polynomial time.

Finally, we have to consider $\textsc{BackReach}(L, T^*)$. Due to Lemma 5.4 and Theorem 3.3 we have

$$\textsc{BackReach}(L, T^*) = \textsc{BackReach}(L, (W\overline{R})^*) = \textsc{Reach}(L^{\mathrm{R}}, (R^{\mathrm{R}}\overline{W^{\mathrm{R}}})^*)^{\mathrm{R}}.$$

By closure properties and the statement above we obtain the effective and efficient regularity of $\textsc{BackReach}(L, T^*)$. □

We can use Theorem 5.11 to prove the effective preservation of regularity of some other language classes. The following corollary lists some of them:

Corollary 5.13 *Let A be an alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma_A^*$ be regular. Then $\textsc{Reach}(L, T^*)$ and $\textsc{BackReach}(L, T^*)$ are effectively regular if*

(1) $T = \overline{R_1}W\overline{R_2}$ *for some regular sets* $W, R_1, R_2 \subseteq A^*$,
(2) $T = \{t\}$ *for some* $t \in \Sigma_A^*$ *(cf. [19]), or*
(3) $T \subseteq A^* \cup \overline{A}^*$.

In all of these cases the computation of NFAs accepting $\textsc{Reach}(L, T^*)$ and $\textsc{BackReach}(L, T^*)$, respectively, is possible in polynomial time.

Proof First, we prove (1). Then we have

$$(\overline{R_1}W\overline{R_2})^* = \{\varepsilon\} \cup \overline{R_1}(W\overline{R_2R_1})^*W\overline{R_2}\,.$$

Then, due to Proposition 3.5 and Theorem 5.11 $\textsc{Reach}(L, (\overline{R_1}W\overline{R_2})^*)$ is effectively regular.

Next, we consider (2). Due to Proposition 4.3 we can compute a word $s \in \overline{A}^*A^*\overline{A}^*$ with $s \equiv t$. Using (1) we know that $\textsc{Reach}(L, s^*)$ is effectively regular. Hence $\textsc{Reach}(L, t^*)$ is regular as well.

Finally, we consider (3). Let $W, R \subseteq A^*$ with $T = W \cup \overline{R}$. Then we have $T^* = (W^*\overline{R}^*)^*$. Hence, due to Theorem 5.11 $\textsc{Reach}(L, T^*)$ is effectively regular. □

As we have seen, Theorem 5.11 implies the effective preservation of regularity for a large class of sets of transformation sequences. However, we think our result can be generalized to an even larger class of languages. Recall that $T \subseteq \Sigma_A^*$ is read-write independent if for each pair s, t of words in T there is some particular de-shuffled combination $\pi_A(s)\pi_{\overline{A}}(t)$ of these words in T. A possible generalization is to drop the requirement that this combination is de-shuffled:

Conjecture 5.14 *Let A be an alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma_A^*$ be regular such that for each $s, t \in T$ there is $r \in T$ with $\pi_A(r) = \pi_A(s)$ and $\pi_{\overline{A}}(r) = \pi_{\overline{A}}(t)$ holds. We conjecture that in this case $\textsc{Reach}(L, T^*)$ is effectively regular.*

The proof of Theorem 5.11 does not work in this case. At least the utilization of Lemma 5.4, where we de-shuffle the words from T, is impossible in certain cases. For example, we have $\textsc{Reach}(\{\varepsilon\}, (\overline{a}aa)^*) = \{\varepsilon\} \neq a^* = \textsc{Reach}(\{\varepsilon\}, (aa\overline{a})^*)$. However, possibly the construction of our PDA $\mathfrak{P}$ in the proof of Theorem 5.5 can be modified to this more general case.

6 Partially Lossy Queues

Until now we have only considered queues which are reliable. We can also prove the results from the previous sections for (partially) lossy queue automata. These partially lossy queue automata are queue automata with an additional uncontrollable action which is forgetting parts of its contents that are specified by a so-called lossiness alphabet.

Definition 6.1 A *lossiness alphabet* is a tuple $\mathcal{L} = (F, U)$ where F and U are two finite sets with $F \cap U = \emptyset$. We call F the set of *forgettable* letters and U the set of *unforgettable* letters.

From a given lossiness alphabet $\mathcal{L} = (F, U)$ we also obtain the alphabet $A_{\mathcal{L}} = F \cup U$ of all possible queue contents and the alphabet $\Sigma_{\mathcal{L}} = A_{\mathcal{L}} \cup \overline{A_{\mathcal{L}}}$ of all queue actions.

In fact, a partially lossy queue is allowed to forget any letter from F in its content at any time. Here, we first consider partially lossy queues with restricted lossiness. Concretely, we consider only the computations in which the queue forgets letters when necessary. That is, if the queue tries to read a letter which is preceded by some other, forgettable letters.

Formally, the computations of such restricted partially lossy queues are defined as follows:

Definition 6.2 Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $\bot \notin A_{\mathcal{L}}$. Then the map $\circ_{\mathcal{L}} \colon (A_{\mathcal{L}}^* \cup \{\bot\}) \times \Sigma_{\mathcal{L}}^* \to (A_{\mathcal{L}}^* \cup \{\bot\})$ is defined for each $v \in A_{\mathcal{L}}^*$, $a, b \in A_{\mathcal{L}}$, and $t \in \Sigma_{\mathcal{L}}^*$ as follows:

1. $v \circ_{\mathcal{L}} \varepsilon = v$
2. $v \circ_{\mathcal{L}} at = va \circ_{\mathcal{L}} t$
3. $$bv \circ_{\mathcal{L}} \overline{a}t = \begin{cases} v \circ_{\mathcal{L}} t & \text{if } a = b \\ v \circ_{\mathcal{L}} \overline{a}t & \text{if } b \in F \setminus \{a\} \\ \bot & \text{otherwise} \end{cases}$$
4. $\varepsilon \circ_{\mathcal{L}} \overline{a}t = \bot \circ_{\mathcal{L}} t = \bot$

Let A be an alphabet and $v, w \in A^*$ be two words. Then v is a *subword* of w (denoted by $v \preceq w$) if we have $w \in \{v\} \sqcup A^*$. The induced relation $\preceq \subseteq (A^*)^2$ is a partial ordering on A^*. Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $v, w \in A_{\mathcal{L}}^*$. We say that v is an $\mathcal{L}$-*subword* of w (denoted by $v \preceq_{\mathcal{L}} w$) if $\pi_U(w) \preceq v \preceq w$ holds, i.e., v is a subword of w which contains at least all unforgettable letters from w. It is easy to see, that $\preceq_{(\emptyset,U)}$ is the equality relation and $\preceq_{(F,\emptyset)}$ is the subword relation.

Next, we want to describe the computations of non-restricted partially lossy queues. In [26] we have proven that the set of reachable queue contents after application of $t \in \Sigma_{\mathcal{L}}^*$ on some content $v \in A_{\mathcal{L}}^*$ is the set of all $\mathcal{L}$-subwords of $v \circ_{\mathcal{L}} t$. To

this end, we define up- and downclosures of a language $L \subseteq A_{\mathcal{L}}^*$ with respect to $\preceq_{\mathcal{L}}$: the *downclosure* of L is

$$\downarrow_{\mathcal{L}} L := \{v \in A_{\mathcal{L}}^* \mid \exists w \in L \colon v \preceq_{\mathcal{L}} w\}.$$

Similarly, the *upclosure* of L is

$$\uparrow_{\mathcal{L}} L := \{w \in A_{\mathcal{L}}^* \mid \exists v \in L \colon v \preceq_{\mathcal{L}} w\}.$$

Hence, the set of reachable contents of a (non-restricted) partially lossy queue after application of t on the input v is $\downarrow_{\mathcal{L}} (v \circ_{\mathcal{L}} t)$. Therefore, we define our reachability problems as follows:

Problem 6.3 Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $L \subseteq A_{\mathcal{L}}^*$ be a set of queue contents, and $T \subseteq \Sigma_{\mathcal{L}}^*$ be a regular set of transformation sequences. The set of queue contents that are reachable from L via T is

$$\textsc{Reach}_{\mathcal{L}}(L, T) := \uparrow_{\mathcal{L}} ((L \circ_{\mathcal{L}} T) \setminus \{\bot\})$$

and the set of queue contents that can reach L via T is

$$\textsc{BackReach}_{\mathcal{L}}(L, T) := \uparrow_{\mathcal{L}} \{v \in A_{\mathcal{L}}^* \mid (v \circ_{\mathcal{L}} T) \cap L \neq \emptyset\}.$$

Partially lossy queue automata with lossiness alphabets $\mathcal{L} = (\emptyset, U)$ are reliable. Hence, we have $\textsc{Reach}_{\mathcal{L}} = \textsc{Reach}$ and $\textsc{BackReach}_{\mathcal{L}} = \textsc{BackReach}$ in this case. In this reliable case we have a strong duality between forwards and backwards reachability. However, this duality does not hold for arbitrary lossiness alphabets: if $\mathcal{L} = (F, U)$ is a lossiness alphabet with $a \in F$ we have $\textsc{Reach}_{\mathcal{L}}(\{\varepsilon\}, a) = \{\varepsilon, a\}$, which cannot be transformed into $\textsc{BackReach}_{\mathcal{L}}(\{\varepsilon\}.\overline{a}) = F^* a F^*$ using reversal. Hence, we have to consider forwards and backwards reachability in this case. Anyway, we will see later in this section that we can reduce forwards and backwards reachability for arbitrary partially lossy queues to reachability in reliable queues.

Now, we consider fully lossy queues: let $\mathcal{L} = (F, \emptyset)$ be a lossiness alphabet. Then, for regular languages $L \subseteq A_{\mathcal{L}}^* = F^*$ and $T \subseteq \Sigma_{\mathcal{L}}^*$, the set $\textsc{Reach}_{\mathcal{L}}(L, T)$ has a decidable membership problem [9] and, since it is downwards closed under the subword ordering $\preceq$ [10], it is regular. Though, we cannot compute an NFA accepting this set [11, 12] and this holds even if we start from the empty queue, only (i.e., $L = \{\varepsilon\}$). Surprisingly, the set $\textsc{BackReach}_{\mathcal{L}}(L, T)$ is effectively regular [9], but the computation of an NFA accepting this language is not primitive recursive [13, 14].

Hence, again we try to approximate the reachability problem with the help of meta-transformations. To this end, we need another definition:

Definition 6.4 Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $w = a_1 a_2 \dots a_n \in A_{\mathcal{L}}^*$ with $a_1, a_2, \dots, a_n \in A_{\mathcal{L}}$. The set of the *reduced $\mathcal{L}$-superwords* of w is

$$\text{redsup}_{\mathcal{L}}(w) := \{w_1 a_1 w_2 a_2 \dots w_n a_n \mid \forall 1 \leq i \leq n \colon w_i \in (F \setminus \{a_i\})^*\}.$$

Let $w \in A_{\mathcal{L}}^*$. Then it is easy to see, that a reduced $\mathcal{L}$-superword $v \in \text{redsup}_{\mathcal{L}}(w)$ also is an $\mathcal{L}$-superword of w, i.e., $w \preceq_{\mathcal{L}} v$. However, in general $w \preceq_{\mathcal{L}} v$ does not imply that $v \in \text{redsup}_{\mathcal{L}}(w)$ since, e.g., for $w \preceq_{\mathcal{L}} v$ it is allowed to add some

forgettable letters at the end of w. If $F = \emptyset$ holds, then there is exactly one reduced $\mathcal{L}$-superword $\mathrm{redsup}_{\mathcal{L}}(w) = \{w\}$. Note that $\mathrm{redsup}_{\mathcal{L}}(w)$ is effectively regular. We can also extend this notion to languages:

Lemma 6.5 *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $L \subseteq A_{\mathcal{L}}^*$ be regular. Then the language $\mathrm{redsup}_{\mathcal{L}}(L) := \bigcup_{w \in L} \mathrm{redsup}_{\mathcal{L}}(w)$ is effectively regular. An NFA accepting $\mathrm{redsup}_{\mathcal{L}}(L)$ can be computed in polynomial time.*

Proof (idea) Let $\mathfrak{A} = (Q_{\mathfrak{A}}, A_{\mathcal{L}}, I_{\mathfrak{A}}, \Delta_{\mathfrak{A}}, F_{\mathfrak{A}})$ be an NFA accepting L. We can compute an NFA $\mathfrak{B} = (Q_{\mathfrak{B}}, A_{\mathcal{L}}, I_{\mathfrak{B}}, \Delta_{\mathfrak{B}}, F_{\mathfrak{B}})$ as follows:

- $Q_{\mathfrak{B}} := Q_{\mathfrak{A}} \times A_{\mathcal{L}}$,
- $I_{\mathfrak{B}} := I_{\mathfrak{A}} \times A_{\mathcal{L}}$,
- $F_{\mathfrak{B}} := F_{\mathfrak{A}} \times A_{\mathcal{L}}$, and
- $\Delta_{\mathfrak{B}} := \{((p,a),a,(q,b)) \mid (p,a,q) \in \Delta_{\mathfrak{A}}\} \cup \{((p,b),a,(p,b)) \mid a \in F \setminus \{b\}\}$.

In other words, we simulate $\mathfrak{A}$ in the first component of the states of $\mathfrak{B}$. In the second component we guess the letter which $\mathfrak{A}$ reads on its next step. With the help of this information we are able to read some other forgettable letters. Hence, we obtain $L(\mathfrak{B}) = \mathrm{redsup}_{\mathcal{L}}(L)$. □

Now, we can state the following connection between partially lossy computations $\circ_{\mathcal{L}}$ and reduced $\mathcal{L}$-superwords:

Lemma 6.6 *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $v, w, t \in A_{\mathcal{L}}^*$. Then we have $v \circ_{\mathcal{L}} \overline{t} = w$ if, and only if, there is $s \in \mathrm{redsup}_{\mathcal{L}}(t)$ with $v = sw$.*

Proof We prove this by induction on the length of t. First, assume $t = \varepsilon$. Then we have $v = v \circ_{\mathcal{L}} \overline{t} = w$, $\varepsilon \in \mathrm{redsup}_{\mathcal{L}}(t)$, and $v = \varepsilon w = w$.

Next, let $t = at'$ for some $a \in A_{\mathcal{L}}$ and $t' \in A_{\mathcal{L}}^*$. Assume $v \circ_{\mathcal{L}} \overline{t} = w$. Then we have $w = v \circ_{\mathcal{L}} \overline{t} = (v \circ_{\mathcal{L}} \overline{a}) \circ_{\mathcal{L}} \overline{t'}$. By definition of $\circ_{\mathcal{L}}$ there are $v_1 \in (F \setminus \{a\})^*$ and $v_2 \in A_{\mathcal{L}}^*$ with $v = v_1 a v_2$, $v \circ_{\mathcal{L}} \overline{a} = v_2$, and $v_2 \circ_{\mathcal{L}} \overline{t'} = w$. By induction hypothesis there is $s' \in \mathrm{redsup}_{\mathcal{L}}(t')$ with $v_2 = s'w$. Set $s := v_1 a s'$. Then we see $s \in \mathrm{redsup}_{\mathcal{L}}(at') = \mathrm{redsup}_{\mathcal{L}}(t)$ and $v = v_1 a v_2 = v_1 a s' w = sw$.

Conversely, let $s \in \mathrm{redsup}_{\mathcal{L}}(t)$ with $v = sw$. Then by definition there is $s_1 \in (F \setminus \{a\})^*$ and $s_2 \in A_{\mathcal{L}}^*$ with $s = s_1 a s_2$ and $s_2 \in \mathrm{redsup}_{\mathcal{L}}(t')$. By $v = sw$ there is $v_2 \in A_{\mathcal{L}}^*$ with $v = sw = s_1 a s_2 w = s_1 a v_2$, i.e., $v_2 = s_2 w$. By induction hypothesis we have $v_2 \circ_{\mathcal{L}} \overline{t'} = w$. We also have $v \circ_{\mathcal{L}} \overline{a} = v_2$ implying

$$v \circ_{\mathcal{L}} \overline{t} = (v \circ_{\mathcal{L}} \overline{a}) \circ_{\mathcal{L}} \overline{t'} = v_2 \circ_{\mathcal{L}} \overline{t'} = w\,. \qquad \square$$

With the help of Lemma 6.6 we can finally prove the following reductions from reachability in partially lossy queues to reachability in reliable queues:

Proposition 6.7 *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $L, T \subseteq A_{\mathcal{L}}^*$. Then the following statements hold:*

1. $(L \circ_{\mathcal{L}} T) \setminus \{\bot\} = (L \circ T) \setminus \{\bot\}$

2. $(L \circ_{\mathcal{L}} \overline{T}) \setminus \{\bot\} = (L \circ \overline{\mathit{redsup}_{\mathcal{L}}(T)}) \setminus \{\bot\}$
3. $\textsc{Reach}_{\mathcal{L}}(L, T) = {\downarrow_{\mathcal{L}}}\, \textsc{Reach}(L, T)$
4. $\textsc{Reach}_{\mathcal{L}}(L, \overline{T}) = {\downarrow_{\mathcal{L}}}\, \textsc{Reach}(L, \overline{\mathit{redsup}_{\mathcal{L}}(T)})$
5. $\textsc{BackReach}_{\mathcal{L}}(L, T) = {\uparrow_{\mathcal{L}}}\, \textsc{BackReach}({\uparrow_{\mathcal{L}}}\, L, T)$
6. $\textsc{BackReach}_{\mathcal{L}}(L, \overline{T}) = {\uparrow_{\mathcal{L}}}\, \textsc{BackReach}({\uparrow_{\mathcal{L}}}\, L, \overline{\mathit{redsup}_{\mathcal{L}}(T)})$

Proof First, we prove (1):

$$(L \circ_{\mathcal{L}} T) \setminus \{\bot\} = LT = (L \circ T) \setminus \{\bot\}.$$

To prove (2), let $x \in (L \circ_{\mathcal{L}} \overline{T}) \setminus \{\bot\}$. Then there are $w \in L$ and $t \in T$ with $x = w \circ_{\mathcal{L}} \bar{t}$. By Lemma 6.6 there is $s \in \overline{\mathrm{redsup}_{\mathcal{L}}(t)} \subseteq \overline{\mathrm{redsup}_{\mathcal{L}}(T)}$ with $w = sx$. Then we have $x = w \circ \bar{s}$ and, hence, $x \in (L \circ \overline{\mathrm{redsup}_{\mathcal{L}}(T)}) \setminus \{\bot\}$.

Now, let $x \in (L \circ \overline{\mathrm{redsup}_{\mathcal{L}}(T)}) \setminus \{\bot\}$. Then there are $w \in L$ and $s \in \mathrm{redsup}_{\mathcal{L}}(T)$ with $x = w \circ \bar{s}$ and, therefore, $sx = w$. There is $t \in T$ with $s \in \mathrm{redsup}_{\mathcal{L}}(t)$. By Lemma 6.6 we have $x = w \circ_{\mathcal{L}} \bar{t}$ and, therefore, $x \in (L \circ_{\mathcal{L}} \overline{T}) \setminus \{\bot\}$.

The equations (3)-(6) are direct consequences of (1) and (2) as well as Theorem 3.3. □

Finally, we can prove that our results from the previous sections also hold for arbitrary partially lossy queues:

Theorem 6.8 *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $L \subseteq A^*_{\mathcal{L}}$ be regular, and $T \subseteq \Sigma^*_{\mathcal{L}}$ be regular and closed under $\equiv_{\mathcal{L}}$ (where $s \equiv_{\mathcal{L}} t$ if $v \circ_{\mathcal{L}} s = v \circ_{\mathcal{L}} t$ for each $v \in A^*_{\mathcal{L}}$). Then $\textsc{Reach}_{\mathcal{L}}(L, T)$ and $\textsc{BackReach}_{\mathcal{L}}(L, T)$ are effectively regular.*

Theorem 6.9 *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $L \subseteq A^*_{\mathcal{L}}$ be regular, and $T \subseteq \Sigma^*_{\mathcal{L}}$ be regular and read-write independent. Then $\textsc{Reach}_{\mathcal{L}}(L, T^*)$ and $\textsc{BackReach}_{\mathcal{L}}(L, T^*)$ are effectively regular.*

Theorem 6.10 *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $L \subseteq A^*_{\mathcal{L}}$ be regular, and $T \subseteq \Sigma^*_{\mathcal{L}}$ be regular. Then $\textsc{Reach}_{\mathcal{L}}(L, T^*)$ and $\textsc{BackReach}_{\mathcal{L}}(L, T^*)$ are effectively regular if*

(1) *$T = \overline{R_1} W \overline{R_2}$ for some regular sets $W, R_1, R_2 \subseteq A^*_{\mathcal{L}}$,*
(2) *$T = \{t\}$ for some $t \in \Sigma^*_{\mathcal{L}}$ (cf. [19, 21]), or*
(3) *$T \subseteq A^*_{\mathcal{L}} \cup \overline{A_{\mathcal{L}}}^*$.*

7 Conclusion and Open Problems

In this paper we considered the reachability problem of reliable and lossy queue automata having exactly one queue. We joined these two models to so-called partially lossy queue automata (plq automata, for short). These automata are allowed to forget a specified subset of their contents at any time. Depending on this specified set, the reachability problem of these automata is either undecidable or inefficient. Hence,

Boigelot et al. [19] and Abdulla et al. [21] tried to approximate the reachability problem with the help of so-called meta-transformations. These are regular languages of transformation sequences such that we can easily compute the set of reachable queue contents. Here, we considered two special kinds of meta-transformations:

1. the set of possible sequences of queue transformations is closed under certain (context-sensitive) commutations of the atomic transformations.
2. the plq automaton alternates between writing of words from a regular language and reading of words from another regular language. This is a generalization of the results [19, 21] where the authors considered queue automata looping through a single sequence of transformations.

In both cases we could prove that, starting with a regular language of queue contents the queue reaches a regular set of new contents.

Until now it is open, whether we can extend our second kind of meta-transformations to plq automata looping through a sequence of multiple such regular languages of write and read actions. We could also try to generalize the *flat queue automata* to automata consisting of simple paths and single loops as well as components which are closed under the aformentioned commutations or alternating between write and read action sequences. Possibly the decidability of these "semi-flat" queue automata is still decidable in NP. We may also ask, in which cases a plq automaton having multiple queues reaches a recognizable set of states reachable from a recognizable set of initial contents. For example, in [19] there is also a result considering multiple reliable queues looping through a sequence of transformations. So, we could also try to generalize our result to multiple queues.

We could also consider automata having other data structures as their memory. So, we could also consider automata with multiple pushdowns. Since these automata are as powerful as queue automata or Turing-machines, we also have to approximate the reachability problem.

Acknowledgment The author would like to thank Dietrich Kuske and the anonymous reviewers of the conference version [1] and the submitted version of this full paper for their helpful suggestions to improve this paper.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Köcher, C.: Reachability problems on partially lossy queue automata. In: RP 2019, Lecture Notes in Computer Science, vol. 11674, pp. 149–163 Springer (2019). https://doi.org/10.1007/978-3-030-30806-3_12
2. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997, Lecture Notes in Computer Science, vol. 1243, pp. 135–150. Springer (1997). https://doi.org/10.1007/3-540-63141-0_10
3. Finkel, A., Willems, B., Wolper, P.: A direct symbolic approach to model checking pushdown systems. Electronic Notes in Theoretical Computer Science **9**, 27–37 (1997). https://doi.org/10.1016/S1571-0661(05)80426-8
4. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000, Lecture Notes in Computer Science, vol. 1855, pp. 232–247. Springer (2000)
5. Mayr, E.: An algorithm for the general petri net reachability problem. SIAM J. Comput. **13**(3), 441–460 (1984). https://doi.org/10.1137/0213029
6. Leroux, J., Schmitz, S.: Demystifying reachability in vector addition systems. In: 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 56–67. IEEE (2015)
7. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., Upper Saddle River (1967)
8. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983). https://doi.org/10.1145/322374.322380
9. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. **127**(2), 91–101 (1996). https://doi.org/10.1006/inco.1996.0053
10. Haines, L.H.: On free monoids partially ordered by embedding. Journal of Combinatorial Theory **6**(1), 94–98 (1969). https://doi.org/10.1016/S0021-9800(69)80111-0
11. Abdulla, P.A., Jonsson, B.: Undecidable verification problems for programs with unreliable channels. Inf. Comput. **130**(1), 71–90 (1996). https://doi.org/10.1006/inco.1996.0083
12. Mayr, R.: Undecidable problems in unreliable computations. Theor. Comput. Sci. **297**(1), 337–354 (2003). https://doi.org/10.1016/S0304-3975(02)00646-1
13. Schnoebelen, P.: Verifying lossy channel systems has nonprimitive recursive complexity. Inf. Process. Lett. **83**(5), 251–261 (2002). https://doi.org/10.1016/S0020-0190(01)00337-4
14. Chambart, P., Schnoebelen, P.: The Ordinal Recursive Complexity of Lossy Channel Systems. In: LICS 2008, pp. 205–216. IEEE Computer Society Press (2008). https://doi.org/10.1109/LICS.2008.47
15. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.-K.: Algorithmic Analysis of Programs with Well Quasi-Ordered Domains. Inf. Comput. **160**(1), 109–127 (2000). https://doi.org/10.1006/inco.1999.2843
16. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. **256**(1), 63–92 (2001). https://doi.org/10.1016/S0304-3975(00)00102-X
17. Haase, C., Schmitz, S., Schnoebelen, P.: The power of priority channel systems. Logical Methods in Computer Science **10**(4:4), 1–39 (2014). https://doi.org/10.2168/LMCS-10(4:4)2014
18. Boigelot, B., Godefroid, P.: Symbolic verification of communication protocols with infinite state spaces using QDDs. Formal Methods in System Design **14**(3), 237–255 (1999). https://doi.org/10.1023/A:1008719024240
19. Boigelot, B., Godefroid, P., Willems, B., Wolper, P.: The power of QDDs. In: Static analysis, Lecture Notes in Computer Science, vol. 1302, pp. 172–186. Springer (1997). https://doi.org/10.1016/S0304-3975(99)00033-X
20. Bouajjani, A., Habermehl, P.: Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. Theor. Comput. Sci. **221**(1), 211–250 (1999). https://doi.org/10.1016/S0304-3975(99)00033-X
21. Abdulla, P.A., Collomb-Annichini, A., Bouajjani, A., Jonsson, B.: Using forward reachability analysis for verification of lossy channel systems. Formal Methods in System Design **25**(1), 39–65 (2004). https://doi.org/10.1023/B:FORM.0000033962.51898.1a
22. Leroux, J., Sutre, G.: Flat counter automata almost everywhere! In: Peled, D.A., Tsay, Y.-K. (eds.) Automated technology for verification and analysis, pp. 489–503. Springer, Berlin (2005). https://doi.org/10.1007/11562948_36

23. Bozga, M., Iosif, R., Konečný, F.: Safety problems are NP-complete for flat integer programs with octagonal loops. In: McMillan, K.L., Rival, X. (eds.) Verification, model checking, and abstract interpretation, pp. 242–261. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54013-4_14
24. Finkel, A., Praveen, M.: Verification of flat FIFO systems. Log. Methods Comput. Sci. **16**(4), 4:1-4:29 (2020). https://doi.org/10.23638/LMCS-16(4:4)2020
25. Schnoebelen, P.: On flat lossy channel machines. arXiv:2007.05269 (2020)
26. Köcher, C., Kuske, D., Prianychnykova, O.: The inclusion structure of partially lossy Queue Monoids and their Trace Submonoids. RAIRO - Theoretical Informatics and Applications **52**(1), 55–86 (2018). https://doi.org/10.1051/ita/2018003
27. Köcher, C.: Rational, Recognizable, and Aperiodic Sets in the Partially Lossy Queue Monoid. In: STACS 2018, LIPIcs, vol. 96, pp. 45:1–45:14. Dagstuhl Publishing (2018). https://doi.org/10.4230/LIPIcs.STACS.2018.45
28. Huschenbett, M., Kuske, D., Zetzsche, G.: The monoid of queue actions. Semigroup forum **95**(3), 475–508 (2017). https://doi.org/10.1007/s00233-016-9835-4
29. Hopcroft, J.E., Motwani, R., Ullman, J.D. Introduction to automata theorey, languages, and computation, Third. Pearson, London (2006)
30. Render, E., Kambites, M.: Rational subsets of polycyclic monoids and valence automata. Inf. Comput. **207**(11), 1329–1339 (2009). https://doi.org/10.1016/j.ic.2009.02.012
31. Berstel, J.: Transductions and context-free languages. Teubner Studienbücher, Berlin (1979). https://link.springer.com/book/10.1007%2F978-3-663-09367-1

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.