Boosting Pushdown and Queue Machines by Preprocessing

Martin Kutrib, Andreas Malcher(✉), and Matthias Wendlandt

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
{kutrib,andreas.malcher,matthias.wendlandt}@informatik.uni-giessen.de

Abstract. Motivated by preprocessing devices occurring for example in the context of syntactic parsers or HTML sanitization, we study pairs of finite state transducers and deterministic machines such as pushdown automata or queue automata as language accepting devices, where the original input is translated by a finite state transducer to an input of the deterministic machine which eventually accepts or rejects the preprocessed input. As deterministic machines we study input-driven machines as well as reversible machines equipped with a pushdown store or a queue store. It turns out that the preprocessing boosts on the one hand the computational power of the machines in all four cases, but on the other hand preserves and adds some positive closure properties as well as decidable problems. Thus, the preprocessing extends the computational power moderately by retaining most of the nice properties of the original machine.

1 Introduction

The syntactical analysis of a computer program, a web page, or an XML document is typically done after the lexical analysis in which the correct formatting of the input is verified, comments are removed, the spelling of the commands is checked, and the sequence of input symbols is translated into a list of tokens. This preprocessing of the input is typically done by a finite state transducer and the output is subsequently processed by a more powerful machine such as, for example, a pushdown automaton. Further examples where the input is preprocessed and afterwards processed by other devices are HTML sanitization and embedded SQL. As a generalization of preprocessing one may have, for example, cascades of preprocessors $P_1, P_2, \ldots, P_n$, where the output of P_i is the input for the next preprocessor P_{i+1}. Cascades of finite state transducers have been used, for example, in [8] for extracting information from natural language texts.

In terms of formal languages, machines processing preprocessed input can be formulated as follows. Let T be some transducer such as, for example, a finite state transducer or a pushdown transducer (see, e.g., [1]), and M be some accepting machine such as, for example, a finite automaton or a pushdown automaton. Then, for such a pair (M, T), we are interested in the set of words w such that $T(w)$ is accepted by M. From a language theoretic perspective it is an immediate question which language classes can be accepted by such composed devices.

© Springer International Publishing AG, part of Springer Nature 2018
C. Câmpeanu (Ed.): CIAA 2018, LNCS 10977, pp. 28–40, 2018.
https://doi.org/10.1007/978-3-319-94812-6_3

Clearly, the answer depends on the power of both components. If T is a finite state transducer and M is a finite automaton, nothing more than regular languages can be described by (M, T), since a finite automaton can be constructed that simultaneously simulates T on the input given and M on the output produced by T. Similarly, if T is a finite state transducer and M is a pushdown automaton, nothing more than the context-free languages can be described. On the other hand, if T is a nondeterministic pushdown transducer and M is a pushdown automaton, then it is possible to construct for any recursively enumerable language L some pair (M, T) accepting L. For the language classes in between these both extremes there are interesting results given in [7]. For example, it is shown that a pair (M, T) of a deterministic pushdown transducer T combined with a deterministic pushdown automaton M can accept the non-context-free language $\{ wcw \mid w \in \{a, b\}^* \}$. Moreover, cascades of deterministic pushdown transducers are studied and a strict hierarchy with respect to the number of transducers is obtained. If we confine ourselves with the combination of finite state transducers and pushdown automata, then the combination of a nondeterministic finite state transducer with a deterministic or nondeterministic pushdown automaton as well as of a deterministic finite state transducer with a nondeterministic pushdown automaton gives nothing more than the context-free languages. Finally, the combination of a deterministic finite state transducer and a deterministic pushdown automaton gives the deterministic context-free languages. Thus, we can summarize that, roughly speaking, the preprocessing by finite state transducers leads to the classes inside the context-free languages, whereas the preprocessing by pushdown transducers leads to language classes beyond the context-free languages. So far, we have not put any restriction on the automata except the property of working deterministically or nondeterministically. It is therefore an obvious approach to consider restricted pushdown automata and to investigate whether the restrictions can be compensated by the preprocessing.

In this paper, we will basically consider two restricted versions of deterministic pushdown automata, namely, input-driven pushdown automata and reversible pushdown automata. Both variants are in addition real-time deterministic pushdown automata, whose corresponding language class is known to be a proper subset of the deterministic context-free languages. Input-driven pushdown automata are ordinary pushdown automata where the actions on the pushdown store are dictated by the input symbols. This variant of pushdown automata has been introduced in 1980 by Mehlhorn in [16] and further investigations have been done in 1985 by von Braunmühl and Verbeek in [5]. The early results comprise the equivalence of nondeterministic and deterministic models and the proof that the membership problem is solvable in logarithmic space. The model has been revisited in 2004 in [2] where, for example, descriptional complexity aspects for the determinization are investigated as well as closure properties and decidability questions which turned out to be similar to those of finite automata. More results on the model may be found in the survey [17].

The second model we are going to investigate in more detail in connection with a preprocessing transducer are reversible pushdown automata which have been introduced in [10] and are basically pushdown automata which are forward and backward deterministic. This means that every configuration has a unique successor configuration and a unique predecessor configuration. Reversible computations are information preserving computations and are mainly motivated by the physical observation that a loss of information results in heat dissipation [15]. For reversible pushdown automata it is known that they accept a language class that lies properly in between the regular languages and the real-time deterministic context-free languages, they share with deterministic pushdown automata the closure under complementation and inverse homomorphism, whereas the closure under union and intersection with regular languages gets lost, and they still have an undecidable inclusion problem.

It turns out that in both cases the preprocessing by weak deterministic finite state transducers leads to language classes that properly contain the original language class, but on the other hand is properly contained in the deterministic context-free languages as well. Thus, the preprocessing boosts the power of the original automata moderately. In addition, some closure properties as well as positive decidability results are preserved as well.

If we replace the data structure of a pushdown store by a queue, we obtain for the above-discussed cases input-driven queue automata [11] and reversible queue automata [13]. Again, we may ask what happens when the input is preprocessed by finite state transducers. Interestingly, we can apply similar methods as are done for pushdown automata and we obtain again language classes that properly contain the original language class, but on the other hand are properly contained in the general language classes. Thus, the preprocessing boosts also in this case the power of the original automata moderately and preserves some closure properties as well as positive decidability results.

2 Definitions and Preliminaries

Let Σ^* denote the set of all words over the finite alphabet Σ. The *empty word* is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The *reversal* of a word w is denoted by w^R. For the *length* of w we write $|w|$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*.

In this paper, the preprocessing of the input will be done by *one-way finite state transducers* which are basically finite automata with the ability to emit symbols. We consider here essentially *deterministic* finite state transducers (DFST) which are formally defined as a system $T = \langle Q, \Sigma, \Delta, q_0, \delta \rangle$, where Q is the set of *internal states*, Σ is the set of *input symbols*, Δ is the set of *output symbols*, q_0 is the initial state, and δ is the *transition function* mapping from $Q \times \Sigma$ to $Q \times \Delta^*$. By $T(w) \in \Delta^*$ we denote the output computed by T on input $w \in \Sigma^*$.

Since we are interested in weak preprocessing devices, we will consider *length-preserving* deterministic finite state transducers, also known as Mealy machines, where the transition function is restricted to be a mapping from $Q \times \Sigma$ to $Q \times \Delta$. Moreover, we will put the additional restriction on the transition function to be *injective* and obtain injective DFSTs (injective Mealy machines).

Let M be an automaton such as, for example, a finite automaton, pushdown automaton, or queue automaton, and T be a transducer such as, for example, a finite state transducer or a pushdown transducer. Furthermore, the output alphabet of T is the input alphabet of M. Then, the language accepted by the pair (M, T) is $L(M, T) = \{ w \in \Sigma^* \mid T(w) \in L(M) \}$.

3 Boosting Input-Driven Machines

A conventional deterministic pushdown automaton (DPDA) is an input-driven pushdown automaton (IDPDA), if the next input symbol defines the next action on the pushdown store. To this end, the input alphabet Σ is divided into three disjunct sets Σ_N, Σ_D, and Σ_R, where a symbol from Σ_N implies a state change only without changing the pushdown store, a symbol from Σ_D implies a state change and the pushing of a symbol, and a symbol from Σ_R implies a state change and the popping of a symbol. This partition of the input alphabet is also called *signature*.

Input-driven pushdown automata have properties which are similar to those of finite automata. For example, it is shown in [5] that the language classes accepted by nondeterministic and deterministic models coincide. Considering the usually studied closure properties it has been shown in [2] that IDPDAs (like finite automata) are closed under the Boolean operations, concatenation, iteration, and reversal. It should be noted that the results for union, intersection, and concatenation only hold in general if the underlying automata have *compatible* signatures, that is, if they possess an identical pushdown behavior on their input symbols. In contrast to finite automata, it is known that IDPDAs are not closed under homomorphism and inverse homomorphism. With regard to decidability questions, inclusion is decidable in case of compatible signatures in contrast to the undecidability of inclusion for arbitrary DPDAs. Together with known positively decidable questions for arbitrary DPDAs, we obtain that the questions of emptiness, finiteness, equivalence, and inclusion are all decidable for IDPDAs, which is true for finite automata as well. Obviously, every language accepted by an IDPDA is a real-time deterministic context-free language. On the other hand, it is easy to see that the language class accepted by IDPDAs is also a proper subset of the real-time deterministic context-free languages. For example, the languages $L_1 = \{ a^n \$ a^n \mid n \geq 1 \}$ and $L_2 = \{ a^n b^{2n} \mid n \geq 1 \}$ are not accepted by any input-driven pushdown automaton. On the other hand, the marginally changed languages $L'_1 = \{ a^n \$ b^n \mid n \geq 1 \}$ and $L'_2 = \{ a^n (bc)^n \mid n \geq 1 \}$ are accepted by IDPDAs. For L'_1 every a induces a push-operation, every b induces a pop-operation, and a \$ leaves the pushdown store unchanged. Similarly, for L'_2 every a induces a push-operation, every b induces a pop-operation, and a c leaves the pushdown store unchanged. Obviously, L'_1 can be obtained from L_1 by a simple finite state transduction which translates all a's before the \$ to a's, and all a's after the \$ to b's. Similarly, L_2' can be obtained from L_2 by translating every a to a and every b alternately to b and c. In both cases we can observe that the preprocessing of the input by a deterministic finite state transducer, which is in addition injective and length-preserving, helps to enlarge the class of languages accepted.

In the following, we will study *tinput-driven* pushdown automata (TDPDA) which are pairs (M, T), where T is an injective and length-preserving DFST and M is an IDPDA. TDPDAs have been introduced in [14] and in the sequel we will summarize basic results on their computational capacity, closure properties, and decidability questions.

By choosing a DFST that realizes the identity it is clear that every IDPDA can be simulated by a TDPDA. Furthermore, as discussed above, language L_1 is an example of a deterministic context-free language that is not accepted by any IDPDA, but accepted by a TDPDA. Thus, TDPDAs can be more powerful than IDPDAs. On the other hand, every TDPDA can be simulated by a real-time DPDA. The basic idea is to compute the output of the DFST internally so that the IDPDA can directly be simulated. Since every IDPDA works in real time, the resulting DPDA works in real time as well. Finally, it is possible to show that the real-time deterministic context-free language $\{\, a^n b^{n+m} a^m \mid n, m \geq 1 \,\}$ cannot be accepted by any TDPDA. Hence, we obtain the following proper hierarchy.

Theorem 1. $\mathscr{L}(\mathit{IDPDA}) \subset \mathscr{L}(\mathit{TDPDA}) \subset \mathscr{L}(\mathit{rt\text{-}DPDA})$.

It is a nice feature of IDPDAs that their nondeterministic and deterministic variants coincide which in addition deepens the analogy to finite automata. Thus, it is an obvious question whether a similar result can be shown for TDPDAs. Since a TDPDA consists of two components, each of which may work deterministically or nondeterministically, we are concerned with four cases. We denote by TDPDA$_{x,y}$ with $x, y \in \{n, d\}$ a TDPDA whose transducer works in mode x and whose pushdown automaton works in mode y. Since determinization is possible for IDPDAs, we obtain that the classes TDPDA$_{d,n}$ and TDPDA$_{d,d}$ as well as TDPDA$_{n,n}$ and TDPDA$_{n,d}$ coincide. On the other hand, language $\{\, a^n b^{n+m} a^m \mid n, m \geq 1 \,\}$ cannot be accepted by any TDPDA$_{d,d}$, but is accepted by a TDPDA$_{n,d}$. Hence, we have the following hierarchy.

Theorem 2.

$$\mathscr{L}(\mathit{TDPDA}_{d,d}) = \mathscr{L}(\mathit{TDPDA}_{d,n}) \subset \mathscr{L}(\mathit{TDPDA}_{n,n}) = \mathscr{L}(\mathit{TDPDA}_{n,d}).$$

For the rest of the section where we will discuss closure properties and decidability questions we confine ourselves to considering TDPDA$_{d,d}$s only. When studying binary language operations such as union, intersection, or concatenation for IDPDAs and TDPDAs it is essential that the signatures of the corresponding IDPDAs are compatible. For IDPDAs the closure under union, intersection, and concatenation is shown in [2], but the compatibility of the signatures has to be provided. If this condition is not fulfilled, the closure results may get lost. Consider, for example, the languages $\{\, a^n b^n c^m \mid n, m \geq 1 \,\}$ and

$\{ a^n b^m c^m \mid n, m \geq 1 \}$ which each can be accepted by an IDPDA. However, both signatures are not compatible and the intersection of both languages gives the non-context-free language $\{ a^n b^n c^n \mid n \geq 1 \}$. In case of binary language operations for TDPDAs we additionally have to require that both transducers realize the same transduction. Then, similar to the proofs for IDPDAs, the closure under the Boolean operations can be shown. Additionally, a detailed construction shows the closure under inverse homomorphism for TDPDAs which is in contrast to IDPDAs.

Theorem 3. *Let (M, T) and (M', T) be two TDPDAs with compatible signatures. Then, TDPDAs accepting the intersection $L(M, T) \cap L(M', T)$, the union $L(M, T) \cup L(M', T)$, the complement $\overline{L(M, T)}$, and the inverse homomorphic image $h^{-1}(L(M, T))$ for some homomorphism h can effectively be constructed.*

On the other hand, one can prove the non-closure under iteration, reversal, length-preserving homomorphism, and concatenation. The latter non-closure result interestingly holds even if both signatures are compatible and both transducers are identical: we consider language $L = \{ a^n b^n \mid n \geq 1 \} \cup \{ b^n a^n \mid n \geq 1 \}$ which is accepted by some TDPDA since a DFST can translate L to language $\{ a^n b^n \mid n \geq 1 \} \cup \{ c^n d^n \mid n \geq 1 \}$ which is clearly accepted by some IDPDA. However, if TDPDAs were closed under concatenation, then $L \cdot L \cap a^+ b^+ a^+ = \{ a^n b^{n+m} a^m \mid n, m \geq 1 \}$ could be accepted by some TDPDA which is a contradiction. The remaining non-closure results can basically be shown by utilizing again the fact that $\{ a^n b^{n+m} a^m \mid n, m \geq 1 \}$ is not accepted by any TDPDA. The closure properties are summarized in Table 1.

Table 1. Closure properties of the language families discussed. Symbols $\cup_c$, $\cap_c$, and $\cdot_c$ denote union, intersection, and concatenation with compatible signatures. Such operations are not defined for DFAs and DPDAs and are marked with '—'.

	$\overline{}$	$\cup$	$\cap$	$\cup_c$	$\cap_c$	$\cdot$	$\cdot_c$	$*$	$h_{l.p.}$	h^{-1}	REV
REG	Yes	Yes	Yes	—	—	Yes	—	Yes	Yes	Yes	Yes
$\mathscr{L}$(IDPDA)	Yes	No	No	Yes	Yes	No	Yes	Yes	No	No	Yes
$\mathscr{L}$(TDPDA)	Yes	No	No	Yes	Yes	No	No	No	No	Yes	No
$\mathscr{L}$(rt-DPDA)	Yes	No	No	—	—	No	—	No	No	Yes	No

Since the questions of emptiness, finiteness, infiniteness, universality, and regularity are decidable for DPDAs, all questions are decidable for TDPDAs as well. However, the question of inclusion is undecidable for DPDAs, but decidable for IDPDAs. Thus, the question arises whether or not inclusion is decidable for two TDPDAs (M, T) and (M', T). Since the inclusion of the languages $L(M, T) \subseteq L(M', T)$ is equivalent to $L(M, T) \cap \overline{L(M, T)} = \emptyset$ and we know that TDPDAs are closed under complementation and intersection and the emptiness problem for TDPDAs is decidable, we obtain that the inclusion problem is decidable under the condition that the signatures of the given TDPDAs are compatible. Moreover, the latter assumption is in fact necessary, since it is shown in [14] that the inclusion problem becomes undecidable in case of incompatible signatures even for IDPDAs with the additional restriction on their pushdown store to be a counter only.

Let us now replace the data structure of a pushdown store by a queue store. In this way, we obtain *input-driven queue automata* (IDQA) introduced in [11] and *tinput-driven queue automata* studied in [12]. Here, every IDQA has a signature $(\Sigma_N, \Sigma_D, \Sigma_R)$, where a symbol from Σ_N leaves the queue store unchanged, a symbol from Σ_D enters a symbol to the queue store, and a symbol from Σ_D removes a symbol from the queue store. Interestingly, for TDQAs we can apply similar ideas and methods as for TDPDAs which lead to similar results as for the pushdown variants.

Similar to our discussion for TDPDAs, both languages $L_1 = \{ a^n\$a^n \mid n \geq 1 \}$ and $L_2 = \{ a^n b^{2n} \mid n \geq 1 \}$ are not accepted by any IDQA, whereas their preprocessed variants $L_1' = \{ a^n\$b^n \mid n \geq 1 \}$ and $L_2' = \{ a^n(bc)^n \mid n \geq 1 \}$ are easily accepted by TDQAs. Hence, TDQAs may be more powerful than IDQAs. On the other hand, every TDQA can be simulated by some real-time deterministic queue automaton (rt-DQA) and it can be shown that the language $\{ a^n b^{n+m} a^m \mid n, m \geq 1 \}$ already used is not accepted by any TDQA as well. Hence, we obtain the following hierarchy.

Theorem 4. *$\mathscr{L}(IDQA) \subset \mathscr{L}(TDQA) \subset \mathscr{L}(rt\text{-}DQA)$.*

Considering deterministic and nondeterministic variants of the underlying transducer T and the underlying IDQA M of a TDQA (M, T) leads again to four cases. For TDPDAs it is known that the underlying IDPDA can always be determinized which leads to two classes depending on whether the underlying finite state transducer is deterministic or nondeterministic. This is no longer true for TDQAs, since it can be shown that language

$$\{ a^n\$h(w_1)\$h(w_2)\$\cdots\$h(w_m) \mid m, n \geq 1, w_k \in \{a,b\}^n, 1 \leq k \leq m, \\ \text{and there exist } 1 \leq i < j \leq m \text{ so that } w_i = w_j \},$$

where h is the homomorphism that maps a to $\#a$ and b to $\#b$, is accepted by some TDQA$_{d,n}$, but not by any TDQA$_{d,d}$. On the other hand, each TDQA$_{n,n}$ can be converted to an equivalent TDQA$_{n,d}$. Here, the basic idea is to shift the nondeterminism of the IDQA to the transducer. This means basically that the transducer additionally guesses the nondeterministic moves of the IDQA and outputs these guesses as suitable symbols which in turn can be processed by an IDQA in a deterministic way. Finally, it is possible to separate the language classes induced by TDQA$_{d,n}$s and TDQA$_{n,n}$s using the union of the languages $\{ u\$v\#_1 u \mid u, v \in \{a,b\}^* \}$ and $\{ u\$v\#_2 v \mid u, v \in \{a,b\}^* \}$. Altogether, these results lead to the following hierarchy.

Theorem 5. $\mathscr{L}(TDQA_{d,d}) \subset \mathscr{L}(TDQA_{d,n}) \subset \mathscr{L}(TDQA_{n,n}) = \mathscr{L}(TDQA_{n,d})$.

Under the condition of compatible signatures it is again possible to show the closure under the Boolean operations. However, the closure under union and intersection may get lost if the signatures are no longer compatible. For example, the languages $\{ ba^nca^nb \mid n \geq 0 \}$ and $\{ ba^nba^mca^mb \mid m, n \geq 0 \}$ are each accepted by some TDQA with different signatures, but it is shown in [6] that their union is not even accepted by any real-time DQA.

Theorem 6. *Let (M, T) and (M', T) be two TDQAs with compatible signatures. Then, TDQAs accepting the intersection $L(M, T) \cap L(M', T)$, the union $L(M, T) \cup L(M', T)$, and the complement $\overline{L(M, T)}$ can effectively be constructed.*

Interestingly, the reversal of the union of the above languages is accepted by some TDQA, which shows the non-closure under reversal. Further non-closure results are known for concatenation, iteration, and length-preserving homomorphism. These proofs are basically identical to that for TDPDAs, since the proofs refer to the fact that language $\{ a^nb^{n+m}a^m \mid n, m \geq 1 \}$ is not accepted by any TDPDA, which is true for any TDQA as well. The closure properties are summarized in Table 2. Since it is known that the questions of emptiness, finiteness, universality, inclusion, equivalence, regularity, and context-freeness are undecidable for IDQAs [11], it is clear that all questions are undecidable for TDQAs as well. However, when considering the restricted variant of k-turn deterministic queue automata (DQA$_k$), which means that in every computation at most k changes between increasing and decreasing the queue store may take place for some fixed integer k, then the questions of emptiness, finite, and universality become decidable [11]. These decidability results can be extended to hold for IDQA$_k$s and TDQA$_k$s as well. Furthermore, exploiting again the closure under the Boolean operations in case of compatible signatures and the decidability of emptiness, it can be shown that inclusion and equivalence is decidable for TDQA$_k$s with compatible signatures.

Table 2. Closure properties of the language families discussed. Symbols $\cup_c$ and $\cap_c$ denote union and intersection with compatible signatures. Such operations are not defined for DFAs and DQAs and are marked with '—'.

	$\overline{}$	$\cup$	$\cap$	$\cup_c$	$\cap_c$	$\cdot$	$*$	$h_{l.p.}$	REV
REG	Yes	Yes	Yes	—	—	Yes	Yes	Yes	Yes
$\mathscr{L}$(IDQA)	Yes	No	No	Yes	Yes	No	No	No	No
$\mathscr{L}$(TDQA)	Yes	No	No	Yes	Yes	No	No	No	No
$\mathscr{L}$(rt-DQA)	Yes	No	No	—	—	No	No	No	No

Theorem 7. *Let $k \geq 0$ be a constant and (M, T) as well as (M', T) be k-turn TDQA with compatible signatures. Then, emptiness, finiteness, and universality of $L(M, T)$ is decidable. Furthermore, the inclusion and the equivalence of $L(M, T)$ and $L(M', T)$ is decidable as well.*

Finally, we remark that the decidability of inclusion is no longer true for IDQA$_k$s in case of incompatible signatures which holds for TDQA$_k$s as well, whereas it is an open problem whether or not the equivalence problem is decidable for IDQA$_k$s or TDQA$_k$s in case of incompatible signatures. If we come back to general TDQAs, then it is known that inclusion and equivalence is undecidable even if compatible signatures are considered.

We can summarize so far that the preprocessing of the input in case of input-driven automata with pushdown or queue store boosts their computational power, but at the same time preserves the nice features of input-driven automata such as the closure under Boolean operations as well as the decidability of inclusion and equivalence in case of compatible signatures.

4 Boosting Reversible Machines

Reversible pushdown automata (REV-PDA) are conventional DPDAs that in addition to their transition function δ possess a reverse transition function $\delta^{\leftarrow}$ such that a configuration c' is reached from configuration c by applying δ if and only if c is reached from c' by applying $\delta^{\leftarrow}$. REV-PDAs have been introduced in [10] and it is shown there, for example, that REV-PDAs can be assumed to work in real time and induce a language class that is a proper subset of the real-time deterministic context-free languages which is witnessed by the language $L = \{\, a^n b^n \mid n \geq 1 \,\}$. The gap between reversible and irreversible context-free languages is very small, since the slightly changed language $L' = \{\, a^n c b^{n-1} \mid n \geq 1 \,\}$ is accepted by a REV-PDA. Again as observed for IDPDAs, language L' can be obtained from L by preprocessing the input by an injective and length-preserving deterministic finite state transducer. Hence, we will consider in the following *transducer reversible pushdown automata* (T-REV-PDA) which are pairs (M, T), where T is an injective and length-preserving DFST and M is a REV-PDA. Similarly, if M is a reversible finite automaton or a reversible queue automaton [13] (REV-QA), we obtain *transducer reversible finite automata* (T-REV-FA) as well as *transducer reversible queue automata* (T-REV-QA). All these models have been introduced in [3,4] and we will in the sequel summarize known results with respect to their computational capacity and closure properties. Since general DQAs may perform arbitrarily many λ-steps, which is necessary to show their computational universality, we limit for REV-QAs and T-REV-QAs the maximal number of consecutive λ-steps to a fixed number. Moreover, it can be shown for both models that they can be converted to equivalent REV-QAs and T-REV-QAs, respectively, that work in real time. It should also be noted that in a first definition the DFSTs had to be reversible as well. However, it is possible to cede this condition (as well as the condition of being injective), since it can be shown that both conditions can be recovered by the pair of transducer and automaton so that the same language classes are accepted not depending on the reversibility or injectivity of the given preprocessing transducer.

Concerning the computational capacity we can state that the preprocessing boosts the computational power of all three variants. For T-REV-FAs the equality with the regular languages can be established. As discussed above, a T-REV-PDA can accept the irreversible deterministic context-free language $\{ a^n b^n \mid n \geq 1 \}$, and language $\{ a^m b^n \$ w \# w \mid m, n \geq 0, w \in \{a, b\}^* \}$ is accepted by some T-REV-QA, but is not accepted by any REV-QA. On the other hand, by computing the output of the DFST internally and simulating the automaton directly on the output, every T-REV-PDA can be simulated by a real-time DPDA as well as every T-REV-QA can be simulated by a real-time DQA. The inclusions of both corresponding language classes are proper which is witnessed by the two languages

$$\{ w\$w^R \mid w \in \{a,b\}^* \} \cup \{ w\$c^n \mid w \in \{a,b\}^* \text{ and } |w| = n \} \text{ and}$$
$$\{ w_1\$w_1\#w_2\$w_2 \mid w_1, w_2 \in \{a,b\}^* \} \cup \{ w_1\$c^{|w_1|}\#w_2\$w_2 \mid w_1, w_2 \in \{a,b\}^* \}.$$

Hence, we have the following hierarchies.

Theorem 8. – *$\mathscr{L}$(T-REV-FA) = REG,*
– *$\mathscr{L}$(T-REV-FA) $\subset$ $\mathscr{L}$(REV-PDA) $\subset$ $\mathscr{L}$(T-REV-PDA) $\subset$ $\mathscr{L}$(rt-DPDA), and*
– *$\mathscr{L}$(T-REV-FA) $\subset$ $\mathscr{L}$(REV-QA) $\subset$ $\mathscr{L}$(T-REV-QA) $\subset$ $\mathscr{L}$(rt-DQA).*

So far, we have required that the preprocessing transducer of our devices is deterministic and length-preserving. It has been discussed above that the reversibility and injectivity can be ceded without changing the language classes. This immediately raises the question what happens when both conditions are weakened. We will not discuss nondeterministic transducers here, since we want to stick with reversible models which are deterministic by definition. Interestingly, the condition to be length-preserving can be ceded as well without changing the corresponding language classes which means that the preprocessing transducer may be an arbitrary general finite state transducer. The basic idea of the construction is that such a general transducer T is converted to a length-preserving transducer T' that emits symbols which are identified with the words emitted by T. Then, the automaton M' has to work on this compressed alphabet which means that reading one symbol emitted by T' implies to simulate several steps of the original automaton M, which in turn means that M' must be able to handle compressed symbols over the pushdown alphabet and queue alphabet, respectively. However, this can be achieved for REV-PDAs as well as for REV-QAs by detailed constructions.

Theorem 9. *The family $\mathscr{L}$(T-REV-PDA) is equal to the family of languages accepted by pairs of general DFSTs and REV-PDAs. The family $\mathscr{L}$(T-REV-QA) is equal to the family of languages accepted by pairs of general DFSTs and REV-QAs.*

Finally, we discuss the closure properties of the families $\mathscr{L}$(T-REV-PDA) and $\mathscr{L}$(T-REV-QA) which are summarized in Table 3. We start with the complementation operation for which the positive closure can be shown. The traditional

Table 3. Closure properties of the language classes induced by transducer reversible automata.

	$\overline{}$	$\cup$	$\cap$	$\cup$R	$\cap$R	$\cdot$	$*$	$h_{l.p.}$	h^{-1}	REV
REG	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
$\mathscr{L}$(REV-PDA)	Yes	No	No	No	No	No	No	No	Yes	No
$\mathscr{L}$(T-REV-PDA)	Yes	No	No	Yes	Yes	No	No	No	Yes	No
$\mathscr{L}$(rt-DPDA)	Yes	No	No	Yes	Yes	No	No	No	Yes	No
$\mathscr{L}$(REV-QA)	Yes	No	No	No	No	No	No	No	Yes	No
$\mathscr{L}$(T-REV-QA)	Yes	No	No	Yes	Yes	No	No	No	Yes	No
$\mathscr{L}$(rt-DQA)	Yes	No	No	Yes	Yes	No	No	No	Yes	No

approach to show closure under complementation is to interchange accepting and non-accepting states. However, as in the construction for DPDAs [9] one has to ensure that the complete input is read and that no infinite loop on empty input is ever entered. The latter problem does neither occur for T-REV-PDAs nor for T-REV-QAs since both models can be assumed to work in real time. The former problem can be overcome by introducing a new sink state to which all undefined transitions are redirected and, to preserve reversibility, to log the predecessor of the sink state and the remaining input symbols on the pushdown store and queue store, respectively. For the closure under inverse homomorphism we can apply the fact shown in Theorem 9. In detail, it shown that the inverse homomorphic image can be represented by a pair of a general DFST and a REV-PDA or a REV-QA, respectively. Finally, we can use the DFST of a T-REV-PDA (T-REV-QA) for the reversible simulation of a regular language. Then, a standard construction using the Cartesian product shows the closure under union and intersection with regular languages. We note that REV-PDAs as well as REV-QAs are not closed under both operations in general. One has additionally to ensure that the given regular languages are reversible.

Theorem 10. *Let (M, T) be a T-REV-PDA (resp. T-REV-QA) and R a regular language. Then, a T-REV-PDA (resp. T-REV-QA) accepting the intersection $L(M, T) \cap R$, the union $L(M, T) \cup R$, the complement $\overline{L(M, T)}$, and the inverse homomorphic image $h^{-1}(L(M, T))$ for some homomorphism h can effectively be constructed.*

The families $\mathscr{L}$(T-REV-PDA) and $\mathscr{L}$(T-REV-QA) are not closed under intersection in general. Here, basically the proofs known for DPDAs and rt-DQAs apply. Due to the closure under complementation, both families cannot be closed under union as well. To show the non-closure under concatenation, iteration, reversal, and length-preserving homomorphism one can utilize some languages already used in the corresponding proofs for REV-PDAs, DPDAs, REV-QAs, and rt-DQAs. In all cases, the assumption that $\mathscr{L}$(T-REV-PDA) or $\mathscr{L}$(T-REV-QA) is closed under one operation leads to a language that is no longer acceptable by a DPDA or a rt-DQA, respectively.

Theorem 11. *The families $\mathscr{L}(T\text{-}REV\text{-}PDA)$ and $\mathscr{L}(T\text{-}REV\text{-}QA)$ are neither closed under union, intersection, concatenation, iteration, reversal, nor under length-preserving homomorphism.*

We can summarize that the preprocessing of the input also in case of reversible automata with pushdown or queue store boosts their computational power, preserves the positive closure results of the reversible automata, and adds the closure under union and intersection with regular languages.

References

1. Aho, A.V., Ullman, J.D.: The Theory of Parsing, Translation, and Compiling. Vol. I: Parsing. Prentice-Hall Inc., Englewood Cliffs (1972)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) Symposium on Theory of Computing (STOC 2004), pp. 202–211. ACM (2004)
3. Axelsen, H.B., Kutrib, M., Malcher, A., Wendlandt, M.: Boosting reversible pushdown machines by preprocessing. In: Devitt, S., Lanese, I. (eds.) RC 2016. LNCS, vol. 9720, pp. 89–104. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40578-0_6
4. Axelsen, H.B., Kutrib, M., Malcher, A., Wendlandt, M.: Boosting reversible pushdown and queue machines by preprocessing (2018, Submitted to a journal)
5. von Braunmühl, B., Verbeek, R.: Input-driven languages are recognized in $\log n$ space. In: Karpinski, M., van Leeuwen, J. (eds.) Topics in the Theory of Computation, Mathematics Studies, vol. 102, pp. 1–19. North-Holland, Amsterdam (1985)
6. Cherubini, A., Citrini, C., Crespi-Reghizzi, S., Mandrioli, D.: QRT FIFO automata, breadth-first grammars and their relations. Theor. Comput. Sci. **85**, 171–203 (1991)
7. Citrini, C., Crespi-Reghizzi, S., Mandrioli, D.: On deterministic multi-pass analysis. SIAM J. Comput. **15**(3), 668–693 (1986)
8. Friburger, N., Maurel, D.: Finite-state transducer cascades to extract named entities in texts. Theor. Comput. Sci. **313**(1), 93–104 (2004)
9. Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Reading (1978)
10. Kutrib, M., Malcher, A.: Reversible pushdown automata. J. Comput. Syst. Sci. **78**(6), 1814–1827 (2012)
11. Kutrib, M., Malcher, A., Mereghetti, C., Palano, B., Wendlandt, M.: Deterministic input-driven queue automata: finite turns, decidability, and closure properties. Theor. Comput. Sci. **578**, 58–71 (2015)
12. Kutrib, M., Malcher, A., Wendlandt, M.: Input-driven queue automata with internal transductions. In: Dediu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) LATA 2016. LNCS, vol. 9618, pp. 156–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30000-9_12
13. Kutrib, M., Malcher, A., Wendlandt, M.: Reversible queue automata. Fundam. Inform. **148**(3–4), 341–368 (2016)
14. Kutrib, M., Malcher, A., Wendlandt, M.: Tinput-driven pushdown, counter, and stack automata. Fundam. Inform. **155**(1–2), 59–88 (2017)
15. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961)

16. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_89
17. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News **45**(2), 47–67 (2014)