Tinput-Driven Pushdown Automata

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

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

Abstract. In input-driven pushdown automata (IDPDA) the input alphabet is divided into three distinct classes and the actions on the pushdown store (push, pop, nothing) are solely governed by the input symbols. Here, this model is extended in such a way that the input of an IDPDA is preprocessed by a deterministic sequential transducer. These automata are called tinput-driven pushdown automata (TDPDA) and it turns out that TDPDAs are more powerful than IDPDAs but still not as powerful as real-time deterministic pushdown automata. Nevertheless, even this stronger model has still good closure and decidability properties. In detail, it is shown that TDPDAs are closed under the Boolean operations union, intersection, and complementation. Furthermore, decidability procedures for the inclusion problem as well as for the questions of whether a given automaton is a TDPDA or an IDPDA are developed. Finally, representation theorems for the context-free languages using IDPDAs and TDPDAs are established.

Keywords: Input driven pushdown automata · Sequential transducers · Real-time deterministic context-free languages · Closure properties · Decidability questions

1 Introduction

In order to describe and to analyze "real-life" problems it is desirable to possess theoretical models which have on the one hand a large expressive power to model a large amount of features of the problems. On the other hand, the models should also be manageable in the sense that the commonly studied decidability issues such as emptiness, inclusion, or equivalence are decidable. With regard to the Chomsky hierarchy, two extremes are the regular languages, represented for example by deterministic or nondeterministic finite automata, and the recursively enumerable languages, represented for example by Turing machines. While the latter class is very powerful and allows to describe almost all practical problems one may think of, it is known owing to the Theorem of Rice that almost nothing is decidable for this class. On the other hand, almost all commonly studied problems are decidable for the former class, but the expressive power of regular languages is often not sufficient. Thus, one has to find an agreement in such a way that the expressive power of a model increases at the expense of losing some decidable properties.

J. Durand-Lose and B. Nagy (Eds.): MCU 2015, LNCS 9288, pp. 94–112, 2015.
DOI: 10.1007/978-3-319-23111-2_7

One such extension are pushdown automata (PDA) which are finite automata enlarged with the storage medium of a pushdown store. An interesting subclass of PDAs is represented by *input-driven* PDAs. The essential idea here is that for such devices the operations on the storage medium are dictated by the input symbols. The first references of input-driven PDAs may be found in [5,14], where input-driven PDAs are introduced as classical PDAs in which the input symbols define whether a push operation, a pop operation, or no operation on the pushdown store has to be performed. The main results obtained there show that the membership problem for input-driven PDAs can be solved in logarithmic space, and that the nondeterministic model can be determinized. More on the membership problem has been shown in [8] where the problem is classified to belong to the parallel complexity class NC^1.

The investigation of input-driven PDAs has been revisited in [1,2], where such devices are called visibly PDA or nested word automata. Some of the results are the classification of the language family described by input-driven PDAs to lie properly in between the regular and the deterministic context-free languages, the investigation of closure properties and decidable questions which turn out to be similar to those of regular languages, and descriptional complexity results for the trade-off occurring when nondeterminism is removed from input-driven PDAs. A recent survey with many valuable references on complexity aspects of input-driven PDAs may be found in [16]. Further aspects such as the minimization of input-driven PDAs and a comparison with other subclasses of deterministic context-free languages have been studied in [6,7] while extensions of the model with respect to multiple pushdown stores or more general auxiliary storages are introduced in [12,13]. Recently, the computational power of input-driven automata using the storage medium of a stack and a queue, respectively, have been investigated in [3,11].

The edge between deterministic context-free languages that are accepted by an IDPDA or not is very small. For example, language $\{ a^n\$b^n \mid n \geq 1 \}$ is accepted by an IDPDA where an a means a push-operation, b means a pop-operation, and a $\$$ leaves the pushdown store unchanged. On the other hand, the very similar language $\{ a^n\$a^n \mid n \geq 1 \}$ is not accepted by any IDPDA. Similarly, the language $\{ w\$w^R \mid w \in \{a,b\}^+ \}$, where w^R denotes the reversal of w, is not accepted by any IDPDA, but if w^R is written down with some marked alphabet $\{\hat{a},\hat{b}\}$, then language $\{ w\$\hat{w}^R \mid w \in \{a,b\}^+ \}$ is accepted by an IDPDA. To overcome these obstacles we consider a sequential transducer that translates some input to some output which in turn is the input for an IDPDA. In the above first example such a transducer translates every a before reading $\$$ to a and after reading $\$$ to b. In the second example a and b are translated to a, b or $\hat{a}$, $\hat{b}$, respectively, depending on whether or not $\$$ has been read. We call such a pair of a sequential transducer and an IDPDA *tinput-driven* PDA (TDPDA). To implement the idea without giving the transducers too much power for the overall computation, essentially, we will consider only deterministic injective and length-preserving transducers. The detailed definition of a TDPDA is in Sect. 2. Results on the computational capacity of TDPDAs are obtained in Sect. 3. It turns out

that TDPDAs are more powerful than IDPDAs, but less powerful than real-time deterministic pushdown automata. Thus, TDPDAs are a proper generalization of IDPDAs. Moreover, the determinization of TDPDAs is possible for IDPDAs as long as the corresponding sequential transducer is deterministic. IDPDAs have nice closure properties and decidability questions. In Sects. 4 and 5, we show similar results for TDPDAs. In detail, constructions for the closure under the union, intersection, complementation, and inverse homomorphism are given as well as a decidability procedure for inclusion. It should be noted that the constructions are possible as long as the underlying automata have compatible signatures, that is, an identical pushdown behavior on the input symbols. We show that IDPDAs and TDPDAs are *not* closed under union and intersection, and inclusion becomes *undecidable* in case of incompatible signatures. Finally, we present in Sect. 6 a construction that proves that IDPDAs and TDPDAs are sufficient to represent all context-free languages under λ-free homomorphism.

2 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 set of words of length at most $n \geq 0$ is denoted by $\Sigma^{\leq n}$. 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*.

A classical deterministic pushdown automaton is called input-driven if the next input symbol defines the next action on the pushdown store, that is, pushing a symbol onto the pushdown store, popping a symbol from the pushdown store, or changing the state without modifying the pushdown store. To this end, we assume the input alphabet Σ to be partitioned into the sets Σ_N, Σ_D, and Σ_R, that control the actions state change only (N), push (D), and pop (R). A formal definition is:

Definition 1. *A* deterministic input-driven pushdown automaton (*IDPDA*) *is a system* $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$, *where*

1. Q *is the finite set of* internal states,
2. Σ *is the finite set of* input symbols *partitioned into the sets* Σ_D, Σ_R, *and* Σ_N,
3. Γ *is the finite set of* pushdown symbols,
4. $q_0 \in Q$ *is the* initial state,
5. $F \subseteq Q$ *is the set of* accepting states,
6. $\bot \notin \Gamma$ *is the* empty pushdown symbol,
7. δ_D *is the partial transition function mapping from* $Q \times \Sigma_D \times (\Gamma \cup \{\bot\})$ *to* $Q \times \Gamma$,
8. δ_R *is the partial transition function mapping from* $Q \times \Sigma_R \times (\Gamma \cup \{\bot\})$ *to* Q,
9. δ_N *is the partial transition function mapping from* $Q \times \Sigma_N \times (\Gamma \cup \{\bot\})$ *to* Q.

A *configuration* of an IDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ is a triple (q, w, s), where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the

input, and $s \in \Gamma^*$ denotes the current pushdown content, where the leftmost symbol is at the top of the pushdown store. The *initial configuration* for an input string w is set to (q_0, w, λ). During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $a \in \Sigma$, $w \in \Sigma^*$, $z, z' \in \Gamma$, and $s \in \Gamma^*$. We set

1. $(q, aw, zs) \vdash (q', w, z'zs)$, if $a \in \Sigma_D$ and $(q', z') \in \delta_D(q, a, z)$,
2. $(q, aw, \lambda) \vdash (q', w, z')$, if $a \in \Sigma_D$ and $(q', z') \in \delta_D(q, a, \bot)$,
3. $(q, aw, zs) \vdash (q', w, s)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q, a, z)$,
4. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q, a, \bot)$,
5. $(q, aw, zs) \vdash (q', w, zs)$, if $a \in \Sigma_N$ and $q' \in \delta_N(q, a, z)$,
6. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_N$ and $q' \in \delta_N(q, a, \bot)$.

So, whenever the pushdown store is empty, the successor configuration is computed by the transition functions with the special empty pushdown symbol $\bot$. As usual, we define the reflexive and transitive closure of $\vdash$ by $\vdash^*$. The language accepted by the IDPDA M is the set $L(M)$ of words for which there exists some computation beginning in the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered. Formally:

$$L(M) = \{\, w \in \Sigma^* \mid (q_0, w, \lambda) \vdash^* (q, \lambda, s) \text{ with } q \in F, s \in \Gamma^* \,\}.$$

The difference between an IDPDA and a classical deterministic pushdown automaton (DPDA) is that the latter makes no distinction on the types of the input symbols, and may perform λ-moves. However, in all cases, there must not be more than one choice of action for any possible configuration. So, the transition function is defined to be a (partial) mapping from $Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\bot\})$ to $Q \times (\Gamma \cup \{\texttt{pop}, \texttt{top}\})$, where it is understood that `pop` means removing the topmost symbol from the pushdown store, `top` means letting the content of the pushdown store unchanged, and a symbol of Γ means entering this symbol at the top of the pushdown store. In general, the family of all languages accepted by an automaton of some type X will be denoted by $\mathscr{L}(X)$.

For the definition of tinput-driven pushdown automata we need the notion of *deterministic one-way sequential transducers* (DST) which are basically deterministic finite automata equipped with an initially empty output tape. In every transition a DST appends a string over the output alphabet to the output tape. The transduction defined by a DST is the set of all pairs (w, v), where w is the input and v is the output produced after having read w completely. Formally, a DST is a system $T = \langle Q, \Sigma, \Delta, q_0, \delta \rangle$, where Q is the finite set of internal states, Σ is the finite set of input symbols, Δ is the finite set of output symbols, $q_0 \in Q$ is the initial state, and δ is the partial transition function mapping $Q \times \Sigma$ to $Q \times \Delta^*$. By $T(w) \in \Delta^*$ we denote the output produced by T on input $w \in \Sigma^*$. In the following, we will consider only injective and length-preserving DSTs which are also known as injective Mealy machines. The general definition is given with an eye towards possible extensions of the following model.

Let M be an IDPDA and T be an injective and length-preserving DST. Furthermore, the output alphabet of T is the input alphabet of M. Then, the pair (M, T) is

called a *tinput-driven pushdown automaton* (TDPDA) and the language accepted by (M, T) is defined as $L(M, T) = \{ w \in \Sigma^* \mid T(w) \in L(M) \}$.

In order to clarify this notion we continue with an example.

Example 2. Language $L_1 = \{ a^n \$ a^n \mid n \geq 1 \}$ is accepted by a TDPDA. Before reading symbol $\$$ the transducer maps an a to an a, and after reading $\$$ it maps an a to a b. Thus, L_1 is translated to $\{ a^n \$ b^n \mid n \geq 1 \}$ which is accepted by some IDPDA.

Similarly, $L_2 = \{ w \$ w^R \mid w \in \{a, b\}^* \}$ can be accepted by some TDPDA. Here, the transducer maps any a, b to a, b before reading $\$$ and to $\hat{a}, \hat{b}$ after reading $\$$. This gives the language $\{ w \$ \hat{w}^R \mid w \in \{a, b\}^* \}$ which clearly belongs to $\mathscr{L}$(IDPDA).

Finally, consider $L_3 = \{ a^n b^{2n} \mid n \geq 1 \}$. Here, the transducer maps an a to a and every b alternately to b and c. This gives language $\{ a^n (bc)^n \mid n \geq 1 \}$ which is accepted by some IDPDA: every a implies a push-operation, every b implies a pop, and every c leaves the pushdown store unchanged. ■

3 Computational Capacity

It is known that the language class accepted by IDPDAs is a proper subset of the deterministic context-free languages [1]. In a TDPDA, the input of the IDPDA is preprocessed by a sequential transducer. We have already seen that TDPDAs are strictly more powerful than IDPDAs. Now the question arises whether a TDPDA can accept languages which are not real-time deterministic context-free. The following theorem answers the question negatively.

Theorem 3. *The family $\mathscr{L}$(TDPDA) is effectively included in the family of real-time deterministic context-free languages.*

Proof. Given a TDPDA (M, T) where $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ is an IDPDA and $T = \langle Q', A, \Sigma, q_0', \delta \rangle$ is an injective length-preserving DST, we will construct a deterministic pushdown automaton $M' = \langle S', A, \Gamma, s_0', F', \bot, \delta' \rangle$ such that $L(M') = L(M, T)$.

The basic idea is that M' first computes the output of the DST T internally and then simulates the IDPDA M. To this end, M' needs to keep track of the states of M and T. Thus, we define $S' = Q \times Q'$ and $s_0' = (q_0, q_0')$. The automaton M' accepts, if the input is read completely and M would be in an accepting state. Hence, $F' = F \times Q'$. The transition function is defined as follows for $p, p' \in Q$, $q, q' \in Q'$, $a \in A$, $a' \in \Sigma$, and $z, z' \in \Gamma$.

$$\delta'((p, q), a, z) = \begin{cases} ((p', q'), \lambda) & \text{if } \delta(q, a) = (q', a') \text{ and } \delta_R(p, a', z) = p', \\ ((p', q'), z) & \text{if } \delta(q, a) = (q', a') \text{ and } \delta_N(p, a', z) = p', \\ ((p', q'), zz') & \text{if } \delta(q, a) = (q', a') \text{ and } \delta_D(p, a', z) = (p', z'). \end{cases}$$

By construction, a word w is accepted by (M, T) if and only if w is accepted by M'. Inspecting δ' shows that M' is indeed a deterministic PDA working in real time. □

The previous theorem gives that the family of languages accepted by tinput-driven automata is a subset of the deterministic context-free languages accepted in real time. The next result shows that this inclusion is proper.

Lemma 4. *The family $\mathscr{L}(TDPDA)$ is a proper subset of the real-time deterministic context-free languages.*

Proof. The language $L = \{ a^n b^{n+m} a^m \mid n, m \geq 0 \}$ is clearly accepted by a deterministic PDA. We will show that L is not accepted by any TDPDA.

In contrast to the assertion, assume that L is accepted by a TDPDA (M, T) with $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ and $T = \langle Q', A, \Sigma, s_0, \delta \rangle$, where T has n states, that is, $|Q'| = n$. Let $w = w'_0 w'_1 w'_2 \in L$ be a word with $w'_0, w'_2 \in \{a\}^*$ and $w'_1 \in \{b\}^*$. Then the output of T on input w is denoted by $w_0 w_1 w_2$ where $|w_i| = |w'_i|$ for $0 \leq i \leq 2$.

When T processes w'_0, it has to enter a cycle after at most n steps. The cycle cannot be left before the first b appears in the input. Similar arguments hold for w'_1 and w'_2. Since T is length-preserving, each w_i, $0 \leq i \leq 2$, has the form

$$y_{i,0} y_{i,1} \cdots y_{i,l_i} (x_{i,0} x_{i,1} \cdots x_{i,m_i})^{t_i} x_{i,0} \cdots x_{i,n_i}, \text{ with } l_i, m_i \leq n, n_i < m_i, t_i \geq 0.$$

Now we turn to the computation of M on $w_0 w_1 w_2$, where the length of each of the three subwords is at least n, and analyze the possible pushdown heights while processing the subwords w_i. Since the lengths of the initial parts $y_{i,0} y_{i,1} \cdots y_{i,l_i}$ are at most n, the pushdown height after processing it increases or decreases by at most n symbols. In total, during one input cycle $x_{i,0} x_{i,1} \cdots x_{i,m_i}$ automaton M may increase the height of the pushdown store, leave it as it is, or decrease it.

Subword $\mathbf{w_0}$: Assume that, in total, during a cycle of w_0 the pushdown height is not increased. Then the total height of the pushdown store is at most n after processing w_0. Moreover, there are two different prefixes w'_0 and $\hat{w}'_0$ so that M has the same pushdown content and is in the same state after processing $w_0 = T(w'_0)$ and $\hat{w}_0 = T(\hat{w}'_0)$. Now we can always choose some $w'_1 \in \{b\}^*$ and $w'_2 \in \{a\}^*$ so that $w'_0 w'_1 w'_2$ belongs to L and, thus, is accepted. Since then $\hat{w}'_0 w'_1 w'_2 \notin L$ is accepted as well, we obtain a contradiction and conclude that the total pushdown height is increased during a cycle of w_0.

Subword $\mathbf{w_1}$: Next, we assume that during a cycle of w_1 the pushdown height is decreased. Then we can choose some w'_1 so that $|T(w'_1)| > n \cdot |T(w'_0)|$. So, the height of the pushdown store is at most n after processing $T(w'_0 w'_1)$. Arguing similarly as above, there must be two words w'_1 and $\hat{w}'_1$ so that M has the same pushdown content and is in the same state after processing $w_0 w_1 = T(w'_0 w'_1)$ and $w_0 \hat{w}_1 = T(w'_0 \hat{w}'_1)$. There is a unique $w'_2 \in \{a\}^*$ so that $w'_0 w'_1 w'_2$ belongs to L and, thus, is accepted. Since $w'_0 \hat{w}'_1 w'_2 \notin L$ is accepted as well, we obtain a contradiction and conclude that the total pushdown height is not decreased during a cycle of w_1.

Now, assume that a cycle of w_1 leaves the total pushdown height unchanged. Then, by providing more b's in the input, the cycle can be passed through arbitrarily often. In particular, there must be two words w'_1 and $\hat{w}'_1$ so that M has

the same pushdown content and is in the same state after processing the two prefixes $w_0w_1 = T(w_0'w_1')$ and $w_0\hat{w}_1 = T(w_0'\hat{w}_1')$. Now the contradiction follows as before. We conclude that the total pushdown height is increased during a cycle of w_1.

Subword w_2: If the pushdown height is not decreased during a cycle of w_2, then its total height is never reduced by more then a constant number of symbols while processing the subword w_2 entirely. Since we know already that the same is true for the subwords w_0 and w_1, the IDPDA M can be simulated by a finite automaton that stores the finite number of accessible symbols at the top of the pushdown store in its state. Since L is not a regular language this is a contradiction. We conclude that the total pushdown height is decreased during a cycle of w_2.

Now we choose two long and different words w_0' and $\hat{w}_0'$ so that M is in the same state and has the same $2n$ symbols on top of the pushdown store after processing $T(w_0')$ and $T(\hat{w}_0')$. The prefix w_0' is completed by w_1' and w_2', where $w_0'w_1'w_2' \in L$ and $|w_2'| < |w_1'|/n$. So, w_2' is such short in comparison to w_1' that the pushdown content pushed while processing $T(w_0')$ is untouched by the computation on $T(w_2')$. It follows that $\hat{w}_0'w_1'w_2' \notin L$ is accepted as well. So, we have a contradiction and obtain that L is not accepted by any TDPDA. □

Determinization

In the previous part we considered a tinput-driven pushdown automaton as a pair of a deterministic sequential transducer and a deterministic input-driven pushdown automaton. The related model of input-driven automata was also investigated in the nondeterministic case [1]. It is shown there that every nondeterministic input-driven pushdown automaton can be transformed into an equivalent deterministic one.

Now, the question arises whether the nondeterministic version of a tinput-driven pushdown automaton can be determinized as well. There are four different working modes for a tinput-driven pushdown automaton. The sequential transducer can be deterministic or nondeterministic and also the input-driven pushdown automaton may be deterministic or nondeterministic. We use the notation $\mathrm{TDPDA}_{x,y}$ with $x, y \in \{d, n\}$ where x stands for the working mode of the transducer and y for the mode of the input-driven pushdown automaton. For example, $\mathrm{TDPDA}_{n,d}$ is a tinput-driven pushdown automaton with a nondeterministic sequential transducer and a deterministic input-driven pushdown automaton.

Theorem 5. *The family of languages accepted by $\mathrm{TDPDA}_{d,d}$'s is properly included in the family of languages accepted by $\mathrm{TDPDA}_{n,d}$'s.*

Proof. By definition we know that every $\mathrm{TDPDA}_{d,d}$ is in particular a $\mathrm{TDPDA}_{n,d}$.

It remains to be shown that there is a language accepted by a $\mathrm{TDPDA}_{n,d}$, but not by any $\mathrm{TDPDA}_{d,d}$. We will use the language $L = \{\, a^n b^{n+m} a^m \mid n, m \geq 0 \,\}$ from Lemma 4 and prove that L is accepted by a $\mathrm{TDPDA}_{n,d}$ M. This can be

done as follows. The nondeterministic sequential transducer writes for every a of the first a-sequence an a as output. Then it writes for every b a b as output until it nondeterministically decides that it has already written as many b's as a's. It continues and writes now for every b an a as output until the first a of the second sequence of a's is reached. Then, for every a a b is output. Subsequently, an IDPDA tests whether its input is of the form $a^n b^n a^m b^m$ for some $m, n \geq 0$. If this is the case, then the input is accepted. Otherwise, it is rejected.

On the other hand, it has been shown in Lemma 4 that L is not accepted by any $\mathrm{TDPDA}_{d,d}$. □

Thus, we can conclude that it is not possible to determinize $\mathrm{TDPDA}_{n,d}$'s as well as $\mathrm{TDPDA}_{n,n}$'s. It remains for us to consider the determinization of $\mathrm{TDPDA}_{d,n}$'s.

Theorem 6. *The family of languages accepted by $\mathrm{TDPDA}_{d,n}$'s and $\mathrm{TDPDA}_{d,d}$'s coincide.*

Proof. It has been shown in [1] that IDPDAs can be determinized. Applying this construction to the IDPDA belonging to a $\mathrm{TDPDA}_{d,n}$, we obtain that every $\mathrm{TDPDA}_{d,n}$ can be converted to an equivalent $\mathrm{TDPDA}_{d,d}$. □

4 Closure Properties

In this section, we investigate the closure properties of tinput-driven pushdown automata. For input-driven pushdown automata, strong closure properties have been derived in [1] *provided that* all automata involved share the same partition of the input alphabet. Here we distinguish this important special case from the general one. For easier writing, we call the partition of an input alphabet a *signature*, and say that two signatures $\Sigma = \Sigma_D \cup \Sigma_R \cup \Sigma_N$ and $\Sigma' = \Sigma'_D \cup \Sigma'_R \cup \Sigma'_N$ are *compatible* if and only if

$$\bigcup_{j \in \{D,R,N\}} (\Sigma_j \setminus \Sigma'_j) \cap \Sigma' = \emptyset \quad \text{and} \quad \bigcup_{j \in \{D,R,N\}} (\Sigma'_j \setminus \Sigma_j) \cap \Sigma = \emptyset.$$

We consider first TDPDAs having compatible signatures and identical translations. Later, we will see that IDPDAs and TDPDAs lose some positive closure properties if the signatures are no longer compatible.

Lemma 7. *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 complement $\overline{L(M,T)}$, and the union $L(M,T) \cup L(M',T)$ can effectively be constructed.*

Proof. Let us first consider the closure under intersection. Since (M,T) and (M',T) have compatible signatures and both TDPDAs apply the sequential transducer T, the closure under intersection follows from the standard construction using the Cartesian product. In detail, we consider the two IDPDAs $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ and $M' = \langle Q', \Sigma', \Gamma', q'_0, F', \bot, \delta'_D, \delta'_R, \delta'_N \rangle$

and define $M'' = \langle Q \times Q', \Sigma \cup \Sigma', \Gamma \times \Gamma', (q_0, q_0'), F \times F', (\bot, \bot), \delta_D'', \delta_R'', \delta_N'' \rangle$ assuming that Σ and Σ' are compatible. The transition functions are defined as follows. Let $q, \hat{q} \in Q$, $q', \hat{q}' \in Q'$, $Z \in \Gamma \cup \{\bot\}$, $\hat{Z} \in \Gamma$, $Z' \in \Gamma' \cup \{\bot\}$, and $\hat{Z}' \in \Gamma'$. For $a \in \Sigma_D \cap \Sigma_D'$, we define $\delta_D''((q, q'), a, (Z, Z')) = ((\hat{q}, \hat{q}'), (\hat{Z}, \hat{Z}'))$ with $\delta_D(q, a, Z) = (\hat{q}, \hat{Z})$ and $\delta_D'(q', a, Z') = (\hat{q}', \hat{Z}')$. For $a \in \Sigma_R \cap \Sigma_R'$, we define $\delta_R''((q, q'), a, (Z, Z')) = ((\hat{q}, \hat{q}'))$ with $\delta_R(q, a, Z) = \hat{q}$ and $\delta_R'(q', a, Z') = \hat{q}'$. For $a \in \Sigma_N \cap \Sigma_N'$, we define $\delta_N''((q, q'), a, (Z, Z')) = ((\hat{q}, \hat{q}'))$ with $\delta_N(q, a, Z) = \hat{q}$ and $\delta_N'(q', a, Z') = \hat{q}'$. For all remaining input symbols $a \in \Sigma \cup \Sigma'$, M'' enters a non-accepting sink state which can never be left once entered. Clearly, (M'', T) is a TDPDA accepting $L(M, T) \cap L(M', T)$.

Next, we consider the closure under complementation. The classical construction for a DPDA is to interchange accepting and non-accepting states. Before doing that two problems have to be overcome. First, the given DPDA may not read its input completely, since some moves are undefined or an infinite λ-loop is entered. Second, it may happen that the given DPDA performs λ-moves leading from an accepting state to a non-accepting state and vice versa. For a TDPDA it is clear from the definition that no λ-moves are performed. Thus, it is sufficient to add a non-accepting state which is entered for so far undefined configurations. This new state cannot be left. It drives the IDPDA component of the TDPDA over the rest of the input obeying the pushdown operations. Finally, accepting and non-accepting states are interchanged.

The effective closure under union follows from the effective closure under intersection and complementation. □

The next result shows that TDPDAs are closed under inverse homomorphism which is in contrast to IDPDAs.

Lemma 8. *Let (M, T) be a TDPDA and h be a homomorphism. Then a TDPDA accepting $h^{-1}(L(M, T))$ can effectively be constructed.*

Proof. For the construction we will need the following mapping which assigns an integer value to each sequence of input symbols. Let $\Sigma = \Sigma_D \cup \Sigma_R \cup \Sigma_N$ and $\varphi : \Sigma^* \to \mathbb{Z}$ be a mapping such that $\varphi(\lambda) = 0$ and $\varphi(x_1 x_2 \cdots x_n) = \sum_{i=1}^{n} v(x_i)$ setting, for $x \in \Sigma$, $v(x) = 1$ if $x \in \Sigma_D$, $v(x) = -1$ if $x \in \Sigma_R$, and $v(x) = 0$ otherwise.

Now, we will consider an IDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$, an injective and length-preserving DST $T = \langle P, \Delta, \Sigma, \delta, p_0 \rangle$, and an arbitrary homomorphism $h : \Lambda^* \to \Delta^*$. We have to construct a TDPDA (M', T') which accepts $h^{-1}(T(L(M)) = \{ w \in \Lambda^* \mid h(w) \in T(L(M)) \}$. The idea of the construction is first to define T' in such a way that T' simulates T and h in its state set and outputs $T(h(w))$ on given input $w \in \Lambda^*$. Since h may map one symbol to a sequence of symbols, but T' has to be length-preserving, the output alphabet of T' will consist of compressed symbols. In a second step we will construct an IDPDA M' working on an alphabet of compressed symbols and accepting all inputs which are originally and uncompressed accepted by M. Since M' works on compressed input symbols, it will have to work on compressed pushdown symbols as well.

Let $m = \max\{\, |h(a)| \mid a \in \Lambda \,\}$ be the maximum length of the image of h. Each (compressed) output symbol will comprise at most m symbols from Δ and two other components ensuring the injectivity of the translation. We now define the DST $T' = \langle P', \Lambda, \Sigma', \delta', p_0' \rangle$ as follows. We set $\Lambda_N = \{\, a_N \mid a \in \Lambda \,\}$ and

$$\begin{aligned} P' &= P \times \Sigma^{\leq m-1} \times \{-m+1, -m+2, \ldots, m-1\}, \\ p_0' &= (p_0, \lambda, 0), \\ \Sigma' &= \Lambda_N \cup \bigcup_{X \in \{D,N,R\}} \left(\Sigma^{\leq m} \times \Lambda \times \Delta^{\leq m}\right)_X . \end{aligned}$$

For the transition function δ' we differentiate two cases:

Case 1: If $a \in \Lambda$ such that $h(a) = \lambda$, then we add

$$\delta'((p, d_1 d_2 \cdots d_r, \ell), a) = ((p, d_1 d_2 \cdots d_r, \ell), a_N),$$

for all $p \in P$, $d_1 d_2 \cdots d_r \in \Sigma^{\leq m-1}$, and $-m < \ell < m$. In this case, we just output a symbol a_N which will be ignored by the IDPDA.

Case 2: We have $a \in \Lambda$ such that $h(a) = b_1 b_2 \cdots b_n$ with $n \geq 1$. For $p \in P$, we compute by $\delta(p, b_1 b_2 \cdots b_n) = (p', c_1 c_2 \cdots c_n)$ the state reached and the output produced in T from p on input $b_1 b_2 \cdots b_n$.

To compute the correct index D, R, or N for the output alphabet, we have to check whether the (compressed) symbols $c_1, c_2, \ldots, c_n$ eventually imply a pop-, top-, or push-action in M. To this end, we calculate the value $V = \varphi(c_1 c_2 \cdots c_n)$. If this value is exactly $-m$ or m, we know that a pop- or push-action, respectively, has to take place. If $-m < V < m$, then the pushdown remains unchanged, but V is stored in the state set. If $V < -m$ or $V > m$, then a pop- or push-action, respectively, has to take place, but not all symbols to be output can be compressed into one symbol. Thus, the remaining symbols and their value are stored in the state set. Formally, let $(p, d_1 d_2 \cdots d_r, \ell)$ be a state in P' with $p \in P$, $d_1 d_2 \cdots d_r \in \Sigma^{\leq m-1}$, and $-m < \ell < m$. To compute in T' the next state s and the output o on input a, that is, $\delta'((p, d_1 d_2 \cdots d_r, \ell), a) = (s, o)$, we distinguish five subcases for $K = \ell + \varphi(c_1 c_2 \cdots c_n)$ as follows:

1. If $K = -m$, then $s = (p', \lambda, 0)$ and $o = (d_1 d_2 \cdots d_r c_1 c_2 \cdots c_n, a, h(a))_R$.
2. If $K = m$, then $s = (p', \lambda, 0)$ and $o = (d_1 d_2 \cdots d_r c_1 c_2 \cdots c_n, a, h(a))_D$.
3. If $-m < K < m$, then we define $s = (p', \lambda, \ell + \varphi(c_1 c_2 \cdots c_n))$ and $o = (d_1 d_2 \cdots d_r c_1 c_2 \cdots c_n, a, h(a))_N$.
4. If $K < -m$, then we determine the maximal integer $1 \leq i \leq n$ such that $\ell + \varphi(c_1 c_2 \cdots c_i) = -m$ and we set $s = (p', c_{i+1} c_{i+2} \cdots c_n, \varphi(c_{i+1} c_{i+2} \cdots c_n))$ and $o = (d_1 d_2 \cdots d_r c_1 c_2 \cdots c_i, a, h(a))_R$.
5. If $K > m$, then we determine the maximal integer $1 \leq i \leq n$ such that $\ell + \varphi(c_1 c_2 \cdots c_i) = m$ and we set $s = (p', c_{i+1} c_{i+2} \cdots c_n, \varphi(c_{i+1} c_{i+2} \cdots c_n))$ and $o = (d_1 d_2 \cdots d_r c_1 c_2 \cdots c_i, a, h(a))_D$.

We observe that T' is injective due to the second and third component of the output and length-preserving. On input $w \in \Lambda^*$, T' outputs a compressed

version of $T(h(w))$ where up to m symbols are compressed and the index D, R, or N determines the actions on the compressed pushdown of the following IDPDA M' which is defined to work with a compressed pushdown alphabet comprising exactly m symbols. Additionally, the two topmost (compressed) pushdown symbols are simulated in the state set and not in the pushdown store. To realize this, we need an additional dummy pushdown symbol $\perp_D$. Formally, we define $M' = \langle Q', \Sigma', \Gamma', (q_0, \lambda), F', \perp, \delta'_D, \delta'_R, \delta'_N \rangle$ with $\Gamma' = \Gamma^m \cup \{\perp_D\}$, state set $Q' = Q \times (\Gamma^{\leq m-1} \cup \Gamma^{\leq m-1} \times \Gamma^m)$, and $F' = F \times (\Gamma^{\leq m-1} \cup \Gamma^{\leq m-1} \times \Gamma^m)$.

Case 1: We have $a \in \Lambda_N$.

A: Let (q, Z) with $q \in Q$ and $Z \in \Gamma^{\leq m-1}$ be a state in Q'. Then, we set $\delta'_N((q, Z), a, Z') = (q, Z)$ for all $Z' \in \Gamma' \cup \{\perp\}$.

B: Let (q, Z, Z') with $q \in Q$, $Z \in \Gamma^{\leq m-1}$, and $Z' \in \Gamma^m$ be a state in Q'. Then, we set $\delta'_N((q, Z, Z'), a, Z'') = (q, Z, Z')$ for all $Z'' \in \Gamma' \cup \{\perp\}$.

Case 2: We have $(a_1 a_2 \cdots a_n, b, d) \in (\Sigma^{\leq m} \times \Lambda \times \Delta^{\leq m})_X$ with $X \in \{D, N, R\}$.

A: Let $(q, c_1 c_2 \cdots c_r)$ with $q \in Q$ and $c_1 c_2 \cdots c_r \in \Gamma^{\leq m-1}$ be a state in Q'. Consider the computation $(q, a_1 a_2 \cdots a_n, c_1 c_2 \cdots c_r) \vdash^n (q', \lambda, Y_1 Y_2 \cdots Y_k)$ in the IDPDA M with $q' \in Q$ and $Y_i \in \Gamma$ for $1 \leq i \leq k$.

1. If $X = N$, then we know that $k \leq m - 1$ due to the definition of T' and we set $\delta'_N((q, c_1 c_2 \cdots c_r), (a_1 a_2 \cdots a_n, b, d), \perp) = (q', Y_1 Y_2 \cdots Y_k)$.
2. If $X = D$, then we know that $m \leq k \leq m + r$ and we set

$$\delta'_D((q, c_1 c_2 \cdots c_r), (a_1 a_2 \cdots a_n, b, d), \perp) = ((q', Y_1 Y_2 \cdots Y_{k-m}), Y_{k-m+1} \cdots Y_k), \perp_D).$$

3. The case $X = R$ does not occur, since the pushdown height is less than m.

B: Let $(q, c_1 c_2 \cdots c_r, Z_1 Z_2 \cdots Z_m)$ with $q \in Q$, $c_1 c_2 \cdots c_r \in \Gamma^{\leq m-1}$, and $Z_1 Z_2 \cdots Z_m \in \Gamma^m$ be a state in Q'. Let us consider the following computation in M: $(q, a_1 a_2 \cdots a_n, c_1 c_2 \cdots c_r Z_1 Z_2 \cdots Z_m) \vdash^n (q', \lambda, Y_1 Y_2 \cdots Y_k)$ with $q' \in Q$ and $Y_i \in \Gamma$ for $1 \leq i \leq k$.

1. If $X = N$, then we know that $m \leq k \leq m + r$ and we set, for $Z \in \Gamma'$,

$$\delta'_N((q, c_1 c_2 \cdots c_r, Z_1 Z_2 \cdots Z_m), (a_1 a_2 \cdots a_n, b, d), Z) = (q', Y_1 Y_2 \cdots Y_{k-m}, Y_{k-m+1} \cdots Y_k).$$

2. If $X = D$, then we know that $2m \leq k \leq 2m + r$ and we set, for $Z \in \Gamma'$,

$$\delta'_D((q, c_1 c_2 \cdots c_r, Z_1 Z_2 \cdots Z_m), (a_1 a_2 \cdots a_n, b, d), Z) = ((q', Y_1 Y_2 \cdots Y_{k-2m}, Y_{k-2m+1} \cdots Y_{k-m}), Y_{k-m+1} \cdots Y_k).$$

3. If $X = R$, then we know that $k \leq m - 1$ and we set, for $Z \in \Gamma' \setminus \{\perp_D\}$,

$$\delta'_R((q, c_1 c_2 \cdots c_r, Z_1 Z_2 \cdots Z_m), (a_1 a_2 \cdots a_n, b, d), Z) = (q', Y_1 Y_2 \cdots Y_k, Z).$$

For $Z = \perp_D$, we set $\delta'_R((q, c_1 c_2 \cdots c_r, Z_1 Z_2 \cdots Z_m), (a_1 a_2 \cdots a_n, b, d), \perp_D) = (q', Y_1 Y_2 \cdots Y_k)$.

For $w \in \Lambda^*$, M' accepts a compressed version of $T(h(w))$ if and only if $T(h(w))$ is accepted by M. Thus, (M', T') accepts the inverse homomorphic image of $L(M, T)$. □

Next, we turn to non-closure results for TDPDAs even if the signatures are compatible and the transducers are identical.

Lemma 9. *$\mathscr{L}$(TDPDA) is not closed under concatenation, Kleene star, reversal, and length-preserving homomorphism.*

Proof. Let $L = \{ a^n b^n \mid n \geq 1 \} \cup \{ b^m a^m \mid m \geq 1 \}$. Language L can easily be accepted by a TDPDA (M, T). The sequential transducer T first maps a to a and then b to b if the first symbol read is an a. Otherwise, reading a b first, the transducer maps first b to a and then a to b. In any case, the language output by the transducer is $\{ a^n b^n \mid n \geq 1 \}$ which can be accepted by an IDPDA M. Now, let us consider the concatenation of $L(M, T)$ and assume that a TDPDA for $L(M, T)^2$ can be constructed. Since a TDPDA can simulate a deterministic finite automaton in a second component of its state set, we obtain that $L(M, T)^2 \cap a^+ b^+ a^+ = \{ a^n b^{n+m} a^m \mid n, m \geq 1 \}$ belongs to $\mathscr{L}$(TDPDA). This is a contradiction to the proof of Lemma 4. Thus, $\mathscr{L}$(TDPDA) is not closed under concatenation even if the TDPDAs have compatible signatures.

The non-closure under Kleene star and length-preserving homomorphism can be shown similarly observing that $L^* \cap a^+ b^+ a^+ = \{ a^n b^{n+m} a^m \mid n, m \geq 1 \}$ and $h(L') = L$ for $L' = \{ a^n b^n c^m d^m \mid n, m \geq 1 \}$, which is accepted by some IDPDA, and homomorphism h such that $h(a) = h(d) = a$ and $h(b) = h(c) = b$.

Finally, consider language $\{ c^n \$_1 b^m a^n \mid n, m \geq 0 \} \cup \{ d^m \$_2 b^m a^n \mid n, m \geq 0 \}$ which is accepted by some IDPDA. On the other hand, its reversal is not even a real-time deterministic context-free language. □

Remark 10. We would like to remark that the language classes $\mathscr{L}$(TDPDA) and $\mathscr{L}$(IDPDA) are not closed under intersection, union, and concatenation in case of incompatible signatures. It suffices to consider the intersection of the languages $\{ a^n b^n c^m \mid n, m \geq 1 \}$ and $\{ a^n b^m c^m \mid n, m \geq 1 \}$ each of which is accepted by some IDPDA. However, the intersection leads to $\{ a^n b^n c^n \mid n \geq 1 \}$ which is not context free. Due to the closure under complementation, both classes cannot be closed under union. For non-closure under concatenation we consider the languages $\{ a^n b^n \mid n \geq 1 \}$ and $\{ b^m a^m \mid m \geq 1 \}$ each of which is accepted by an IDPDA, but their concatenation is not even accepted by any TDPDA due to Lemma 4.

The closure properties discussed in this section are summarized in the following Table 1.

5 Decidability Questions

We recall (see, for example, [10]) that a decidability problem is *semidecidable* (*decidable*) if and only if the set of all instances for which the answer is 'yes' is

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

	—	$\cup$	$\cap$	$\cup_c$	$\cap_c$	$\cdot$	$\cdot_c$	$*$	$h_{l.p.}$	h^{-1}	REV
DFA	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IDPDA	yes	no	no	yes	yes	no	yes	yes	no	no	yes
TDPDA	yes	no	no	yes	yes	no	no	no	no	yes	no
DPDA	yes	no	no	—	—	no	—	no	no	yes	no

recursively enumerable (recursive). Clearly, any decidable problem is also semidecidable, while the converse does not generally hold. An immediate consequence of the effective construction of an equivalent DPDA from a given TDPDA shown in Theorem 3 is the decidability of the decidable problems for deterministic context-free languages. Since an IDPDA is a DPDA by definition, the decidability carries over to the family $\mathscr{L}$(IDPDA) as well.

Lemma 11. *The problems of equivalence, emptiness, universality, finiteness, infiniteness, and regularity are decidable for TDPDAs and IDPDAs.*

It is known that the inclusion problem for deterministic context-free languages is undecidable. However, for TDPDAs with compatible signatures it *is* decidable.

Theorem 12. *Let (M,T) and (M',T) be two TDPDAs with compatible signatures. Then the inclusion of both TDPDAs is decidable.*

Proof. The inclusion $L(M,T) \subseteq L(M',T)$ can equivalently be expressed by $L(M,T) \cap \overline{L(M',T)} = \emptyset$. Since by Lemma 7 the family $\mathscr{L}$(TDPDA) is closed under complementation, we obtain that $\overline{L(M',T)}$ is accepted by some TDPDA (M'',T) having the same signature as M'. Since $\mathscr{L}$(TDPDA) is closed under intersection with compatible signatures by Lemma 7, we obtain a TDPDA (M''',T) which accepts $L(M,T) \cap \overline{L(M',T)}$ and whose emptiness can be tested by Lemma 11. We conclude that the inclusion $L(M,T) \subseteq L(M',T)$ is decidable. □

The role played by the compatibility of the signatures is once more emphasized by the following theorem which states that the inclusion problem becomes even non-semidecidable for incompatible signatures.

The non-semidecidability of the inclusion problem is shown by reduction of the emptiness problem of Turing machines. It is well known that emptiness for such machines is not semidecidable (see, for example, [10]).

In [9] complex Turing machine computations have been encoded in small grammars. Basically, we consider *valid computations of Turing machines*. It suffices to consider deterministic Turing machines with one single tape and one single read-write head. Without loss of generality and for technical reasons,

we assume that the Turing machines can halt only after an odd number of moves, accept by halting, make at least three moves, and cannot print a blank. A valid computation is a string built from a sequence of configurations passed through during an accepting computation.

Let Q be the state set of some Turing machine M, where q_0 is the initial state, $T \cap Q = \emptyset$ is the tape alphabet containing the blank symbol, $\Sigma \subset T$ is the input alphabet, and $F \subseteq Q$ is the set of accepting states. Then a configuration of M can be written as a word of the form T^*QT^* such that $t_1 \cdots t_i q t_{i+1} \cdots t_n$ is used to express that M is in state q, scanning tape symbol t_{i+1}, and t_1 to t_n is the support of the tape inscription. For our purpose the valid computations $\mathrm{VALC}(M)$ of M are now defined to be the set of strings of the form $\$\bar{w}_1\$w_2^R\$\bar{w}_3\$w_4^R\$ \cdots \$\bar{w}_{2n-1}\$w_{2n}^R\$$, where $\bar{T}$ and $\bar{Q}$ are disjoint copies of T and Q, $\$ \notin T \cup Q \cup \bar{T} \cup \bar{Q}$, $w_{2i} \in T^*QT^*$ and $w_{2i-1} \in \bar{T}^*\bar{Q}\bar{T}^*$ are configurations of M, $1 \leq i \leq n$, $\bar{w}_1$ is an initial configuration of the form $\bar{q}_0\bar{\Sigma}^*$, w_{2n} is an accepting configuration of the form T^*FT^*, and w_{i+1} is the successor configuration of w_i, $1 \leq i \leq 2n-1$.

The valid computations can be decomposed into $\mathrm{VALC}_1(M)$ which is the set of strings of the form $\$\bar{w}_1\$w_2^R\$\bar{w}_3\$w_4^R\$ \cdots \$\bar{w}_{2n-1}\$w_{2n}^R\$$, where $\bar{w}_1$ is an initial and w_{2n} is an accepting configuration, and $\bar{w}_{2i+1}$ is the successor configuration of w_{2i}, $1 \leq i \leq n-1$, and $\mathrm{VALC}_2(M)$ which is the set of strings of the form $\$\bar{w}_1\$w_2^R\$\bar{w}_3\$w_4^R\$ \cdots \$\bar{w}_{2n-1}\$w_{2n}^R\$$, where $\bar{w}_1$ is an initial and w_{2n} is an accepting configuration, and w_{2i} is the successor configuration of $\bar{w}_{2i-1}$, $1 \leq i \leq n$. Clearly, the intersection $\mathrm{VALC}_1(M) \cap \mathrm{VALC}_2(M)$ is exactly $\mathrm{VALC}(M)$. The next lemma gives a construction of an IDPDA accepting $\mathrm{VALC}(M)$.

Lemma 13. *Let M be a Turing machine. Then IDPDAs accepting $VALC_1(M)$ and $VALC_2(M)$ can effectively be constructed from M.*

Proof. The IDPDA M_1 accepting $\mathrm{VALC}_1(M)$ uses the input symbols $\Sigma_N = \{\$\}$, $\Sigma_D = \bar{Q} \cup \bar{T}$, $\Sigma_R = Q \cup T$. Whenever it starts to read a configuration with odd number, it pushes all symbols read. In addition it remembers the last three symbols read in its finite control until the state symbol of that configuration is the middle one of these three. When the $\$$ appears in the input, M_1 changes its mode. Now it pops a symbol for every symbol read, thus, verifying that the current configuration is the reversal of the successor configuration of the previous one. Both configurations differ only locally at the state symbol. But from the information remembered in the finite control, the differences can be computed and verified. In addition M_1 checks in its finite control whether $\bar{w}_1$ is an initial configuration, and whether the last configuration is an accepting one.

The IDPDA M_2 accepting $\mathrm{VALC}_2(M)$ works similarly. It uses the input symbols $\Sigma_N = \{\$\}$, $\Sigma_D = Q \cup T$, and $\Sigma_R = \bar{Q} \cup \bar{T}$ instead. In addition it just reads $\bar{w}_1$ (popping from the empty pushdown). □

Now we are prepared to prove the undecidability of the inclusion. Since it is shown for IDPDAs, the result carries over to TDPDAs even if the associated transducers are the same.

Theorem 14. *Let (M,T) and (M',T) be two TDPDAs. Then the inclusion $L(M,T) \subseteq L(M',T)$ is not semidecidable. Let M and M' be two IDPDAs. Then the inclusion $L(M) \subseteq L(M')$ is not semidecidable.*

Proof. We have to show the assertion for IDPDAs only, since IDPDAs are particular TDPDAs. Let M be a Turing machine. From M the two IDPDAs M_1 and M_2 accepting $\mathrm{VALC}_1(M)$ and $\mathrm{VALC}_2(M)$ are constructed according to Lemma 13. Since the family $\mathscr{L}(\mathrm{IDPDA})$ is closed under complementation, an IDPDA M' accepting $\overline{L(M_2)}$ can be constructed.

In contrast to the assertion, assume that the inclusion problem is semidecidable. Then the inclusion $L(M_1) \subseteq L(M') = \overline{L(M_2)}$ is semidecidable. This is equivalent to semidecide $L(M_1) \cap L(M_2) = \emptyset$ which implies that the emptiness of $\mathrm{VALC}(M)$, and hence of $L(M)$, is semidecidable. This is a contradiction. □

We conclude this section with another decidability problem. Given a deterministic pushdown automaton M and a sequential transducer T, is (M,T) a TDPDA or not? Essentially, this question reduces to the question of whether M is an IDPDA or not. If the output alphabet of T is equal to the input alphabet of M and T is injective and length-preserving, then (M,T) is a TDPDA if and only if M is an IDPDA.

First we present an algorithm which tests whether a given DPDA is an IDPDA.

Theorem 15. *Let M be a DPDA. It is decidable whether M is an IDPDA.*

Proof. In order to decide whether a given deterministic pushdown automaton $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta\rangle$ is input driven, in general, it is not sufficient to inspect the transition function since it may contain surplus transitions for situations that never appear in any computation. These could be transitions with λ-moves or transitions that perform conflicting pushdown operations on the same input symbol.

So, essentially, it remains to be tested whether a transition is applied in some computation or whether it is surplus. To this end, we label the transitions of δ uniquely, say by the set of labels $R = \{r_1, r_2, \ldots, r_m\}$, for some $m \geq 0$. Then we consider words over the alphabet R. On input $u \in R^*$ a DPDA $\tilde{M}$ with all states final tries to imitate a computation of M by applying in every step the transition whose label is currently read. If $\tilde{M}$ accepts some input $u_1u_2\cdots u_n$, then there is a computation (not necessarily accepting) of M that uses the transitions $u_1u_2\cdots u_n$ in this order. If conversely there is a computation of M that uses the transitions $u_1u_2\cdots u_n$ in this order, then $u_1u_2\cdots u_n$ is accepted by $\tilde{M}$. So, in order to determine whether a transition with label r_i of M is useful, it suffices to decide whether $\tilde{M}$ accepts an input containing the letter r_i. This decision can be done by testing the emptiness of the deterministic context-free language $L(\tilde{M}) \cap R^*r_iR^*$.

Assume that M' is constructed from M by deleting all surplus transitions. Clearly, M' and M are equivalent. Now, it is checked that there is no transition with a λ-move, and for any input symbol we consider all transitions on this

symbol and check whether the pushdown operations are identical. If and only if this is true for all symbols, M is an IDPDA. □

To decide the general question of whether (M, T) is a TDPDA it is now sufficient to polish the transducer a little bit.

Theorem 16. *Let M be a DPDA and T be a DST. It is decidable whether (M, T) is a TDPDA.*

Proof. By applying Theorem 15 it is first checked that M is an IDPDA. Second, it has to be verified that the output alphabet of T equals the input alphabet of M. Since after the first step all surplus transitions of M are removed, its input alphabet can be determined by inspection of the remaining transitions. Surplus transitions can be removed from T along the lines of the proof of Theorem 15. It should be noted that the emptiness of the output of a DST can be tested the same way as emptiness is tested for deterministic finite automata. After having removed surplus transitions from T, its output alphabet and the question of whether T is length-preserving can be determined by inspection of the remaining transitions. To conclude the proof it suffices to decide the injectivity of T. To this end, we use the result that the functionality of nondeterministic sequential transducers is decidable (see, for example, [17]). Furthermore, it is known (see, for example, [4,18]) that nondeterministic sequential transducers are closed under inversion. To decide the injectivity of T, we construct from T its inverse transducer T^{-1} and test its functionality. Now, T is injective if and only if T^{-1} is functional. □

The previous decidability problem concerns devices. For the languages represented by the devices, the decidability status is an open problem: Let M be a deterministic pushdown automaton and T be a sequential transducer. Does $L(M, T)$ belong to $\mathscr{L}$(TDPDA)?

6 Representation Theorems

In [15] Myhill has proved that the regular languages are exactly the closure of the finite languages under union, concatenation and iteration. Such results open the possibility to characterize certain language families by, in some sense, simpler ones and some kind of operations. Besides they shed some light on the structure of the family itself that may be used as powerful reduction tool in order to simplify some proofs or constructions.

Here we turn to characterize the context-free languages by the closure of the deterministic (t)input-driven pushdown automata languages under λ-free homomorphism. Thus replacing the nondeterminism and free pushdown operations on the input symbols by λ-free homomorphisms and vice versa.

Theorem 17. *(a) Let L be a language belonging to $\mathscr{L}$(IDPDA) and h be a λ-free homomorphism. Then $h(L)$ is a context-free language.*

(b) Let L be a context-free language. Then there exist a λ-free homomorphism h and an IDPDA M so that $L = h(L(M))$.

Proof. (a) Since $\mathscr{L}$(IDPDA) is included in the context-free languages and the latter are closed under λ-free homomorphisms, the assertion follows immediately.

(b) Let the context-free language L be given as $L(M')$ for some nondeterministic pushdown automaton (NPDA) $M' = \langle Q', \Sigma', \Gamma', q_0', F', \bot, \delta' \rangle$, where the transition function maps $Q' \times \Sigma' \times (\Gamma' \cup \{\bot\})$ to the finite subsets of $Q' \times \Gamma'^*$. We may assume that M' never pushes more than one symbol and that – except for pop moves – it never modifies the symbol read at the top of the pushdown store. Clearly, any NPDA can be transformed into such a normal form. Now, the transition function δ' can be represented as finite list of transitions of the form $Q' \times \Sigma' \times (\Gamma' \cup \{\bot\}) \to Q' \times (\Gamma' \cup \{\mathtt{pop}, \mathtt{top}\})$. We fix an arbitrary list T of these transitions and number the elements $t_1, t_2, \ldots, t_m$, for some $m \geq 0$.

The IDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_N, \delta_D, \delta_R \rangle$ is defined by $Q = Q'$, $\Gamma = \Gamma'$, $q_0 = q_0'$, $F = F'$. Furthermore, the input alphabet is given through

$$\begin{aligned} \Sigma_N &= \{\, [a,t,N] \mid a \in \Sigma', t \in T \,\}, \\ \Sigma_D &= \{\, [a,t,D] \mid a \in \Sigma', t \in T \,\}, \text{ and} \\ \Sigma_R &= \{\, [a,t,R] \mid a \in \Sigma', t \in T \,\}. \end{aligned}$$

To conclude the definition of M, for $a \in \Sigma'$, $p, q \in Q$, $z, z' \in \Gamma$, the transition functions are set as

$$\begin{aligned} \delta_N(p,[a,t,N],z) &= q && \text{if } \delta(p,a,z) = (q,\mathtt{top}) \text{ is transition } t \text{ in } T, \\ \delta_D(p,[a,t,D],z) &= (q,z') && \text{if } \delta(p,a,z) = (q,z') \text{ is transition } t \text{ in } T, \text{ and} \\ \delta_R(p,[a,t,R],z) &= q && \text{if } \delta(p,a,z) = (q,\mathtt{pop}) \text{ is transition } t \text{ in } T. \end{aligned}$$

The λ-free homomorphism h maps the input triples to their first component, that is, $h([a,t,S]) = a$, for $a \in \Sigma'$, $t \in T$, and $S \in \{N, D, R\}$.

In order to show that $h(L(M)) = L = L(M')$ we encode accepting computations of M' as follows. Let $w = a_1 a_2 \cdots a_n \in \Sigma'^*$ be an input from $L(M')$. Then the set $\varphi(w)$ contains the word

$$[a_1,t_1,S_1][a_2,t_2,S_2]\cdots[a_n,t_n,S_n]$$

if and only if there is an accepting computation of M' so that, for $1 \leq i \leq n$,

$$(q_0', a_1 a_2 \cdots a_n, \lambda) \vdash^* (p, a_i a_{i+1} \cdots a_n, z\gamma) \vdash (q, a_{i+1} \cdots a_n, \gamma_1\gamma)$$

and $\delta'(p,a_i,z) = (q,op)$ is transition t_i in T and $S_i = N$, $\gamma_1 = z$ if $op = \mathtt{top}$, $S_i = D$, $\gamma_1 = z'z$ if $op = z' \in \Gamma$, $S_i = R$, $\gamma_1 = \lambda$ if $op = \mathtt{pop}$.

Next we consider the language accepted by M. The idea of the construction is that M simulates M'. To this end, it gets some information on the transition chosen by M' as well as on the type of pushdown operation. This information is provided as second and third component of the input symbols. So, being in some state p on input symbol $[a,t,S]$, automaton M tries to simulate transition t of M'.

If this transition fits to state p, input symbol a, and the type of the pushdown operation S, then it is simulated by M; otherwise the transition functions δ' are undefined and the simulation blocks rejecting. So, for all $w \in L(M')$ the set $\varphi(w)$ is accepted by M. We conclude $L(M) \supseteq \varphi(L(M'))$.

Now let $w' = [a_1, t_1, S_1][a_2, t_2, S_2] \cdots [a_n, t_n, S_n] \in L(M)$. By construction this implies that M' accepts $w = a_1 a_2 \cdots a_n$ in a computation that uses the sequence of transitions $t_1 t_2 \cdots t_n$. Therefore, $w' \in \varphi(w)$ and, thus, $L(M) \subseteq \varphi(L(M'))$.

Together we have $\varphi(L(M')) = L(M)$. Furthermore, since the homomorphism h simply removes the last two components of the input triple, we obtain $h(\varphi(L(M'))) = L(M')$ and, thus, $h(L(M)) = L(M')$. □

The proof of the previous theorem reveals immediately that the homomorphic characterization of the context-free languages is also by tinput-driven pushdown automata.

Corollary 18. *A language L is context free if and only if there is a λ-free homomorphism h and a TDPDA M so that $L = h(L(M))$.*

7 Conclusion

In this paper, we have introduced a generalization of input-driven automata in such a way that the input is preprocessed by an injective and length-preserving deterministic sequential transducer. We obtained that almost all positive closure and decidability results for IDPDAs with respect to compatible signatures could be carried over to TDPDAs. It would be interesting to know how these results vary when the properties of the underlying transducer are weakened or strengthened. Possible generalizations would be, for example, non-injective or nondeterministic sequential transducers.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) Symposium on Theory of Computing (STOC 2004), pp. 202–211. ACM (2004)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM **56**, 16 (2009)
3. Bensch, S., Holzer, M., Kutrib, M., Malcher, A.: Input-driven stack automata. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 28–42. Springer, Heidelberg (2012)
4. Bordihn, H., Holzer, M., Kutrib, M.: Economy of description for basic constructions on rational transductions. J. Autom. Lang. Comb. **9**, 175–188 (2004)
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 (1985)
6. Chervet, P., Walukiewicz, I.: Minimizing variants of visibly pushdown automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 135–146. Springer, Heidelberg (2007)

7. Crespi-Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. J. Comput. Syst. Sci. **78**, 1837–1867 (2012)
8. Dymond, P.W.: Input-driven languages are in $\log n$ depth. Inform. Process. Lett. **26**, 247–250 (1988)
9. Hartmanis, J.: Context-free languages and turing machine computations. Proc. Symposia in Applied Mathematics **19**, 42–51 (1967)
10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
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. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: Logic in Computer Science (LICS 2007), pp. 161–170. IEEE Computer Society (2007)
13. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: Ball, T., Sagiv, M. (eds.) Principles of Programming Languages (POPL 2011), pp. 283–294. ACM (2011)
14. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J.W., van Leeuwen, J. (eds.) Automata, Languages and Programming. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980)
15. Myhill, J.: Finite automata and the representation of events. Technical Report TR 57–624, WADC (1957)
16. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News **45**, 47–67 (2014)
17. Schützenberger, M.P.: Sur les relations rationnelles. In: Brakhage, H. (ed.) Automata Theory and Formal Languages. LNCS, vol. 33, pp. 209–213. Springer, Heidelberg (1975)
18. Sheng, Y.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 41–110. Springer, Heidelberg (1997)