

## WHEN INPUT-DRIVEN PUSHDOWN AUTOMATA MEET REVERSIBILITY

MARTIN KUTRIB<sup>1</sup>, ANDREAS MALCHER<sup>1</sup> AND MATTHIAS WENDLANDT<sup>1</sup>

**Abstract.** We investigate subfamilies of context-free languages that share two important properties. The languages are accepted by input-driven pushdown automata as well as by a reversible pushdown automata. So, the languages are input driven and reversible at the same time. This intersection can be defined on the underlying language families or on the underlying machine classes. It turns out that the latter class is properly included in the former. The relationships between the language families obtained in this way and to reversible context-free languages as well as to input-driven languages are studied. In general, a hierarchical inclusion structure within the real-time deterministic context-free languages is obtained. Finally, the closure properties of these families under the standard operations are investigated and it turns out that all language families introduced are anti-AFLs.

**Mathematics Subject Classification.** 68Q45, 68Q68.

### 1. INTRODUCTION

One extension of finite automata in order to enlarge the underlying language class as well as to preserve many positive closure properties and decidable questions is represented by input-driven pushdown automata [26, 32]. Such automata share many desirable properties with finite automata, but still are powerful enough to describe important non-regular behavior. Basically, for such devices the operations on the pushdown store are determined by the input symbols. The first results show that the membership problem for input-driven PDA can be solved in logarithmic space, and that nondeterministic and deterministic models are equivalent. The investigation of input-driven PDA has been renewed in [1, 2], where such devices are called visibly PDA or nested word automata. Important results on the descriptive complexity of input-driven PDA are given, for example, in [9, 30, 31]. The minimization of input-driven PDA has been studied in [7], while in [8] the language family described by input-driven PDA is compared with other subclasses of deterministic context-free languages. Extensions have also been studied, for example, with respect to multiple pushdown stores in [22], more general auxiliary storages in [25], stacks in [6], or queues in [19].

Another line of research investigates (logical) reversibility of automata models. The reversibility of a computation means in essence that every configuration has at most one unique successor configuration and at most one unique predecessor configuration. The main motivation to study reversible computations is the observation that loss of information results in heat dissipation [23]. First studies of reversibility computations have been done for the massively parallel model of cellular automata since the sixties of the last century. Nowadays it is known

---

*Keywords and phrases.* Reversible computation – input-driven pushdown automata – formal languages – closure properties.

<sup>1</sup> Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany.  
{kutrib,malcher,matthias.wendlandt}@informatik.uni-giessen.de

from [27] that every, possibly irreversible, one-dimensional cellular automaton can always be simulated by a reversible one-dimensional cellular automaton in a constructive way. In [5] reversible Turing machines have been considered. Again, a fundamental result is that every Turing machine can be made reversible. These two types of devices received a lot of attention in connection with reversibility. Valuable surveys with further references to literature are, for example, [11] for cellular automata and [28], where one may find a summary of results on reversible Turing machines, reversible cellular automata, and other reversible models such as logic gates, logic circuits, or logic elements with memory (see also [4, 12, 14, 15, 18] for further investigations). Furthermore, reversibility has been studied for several computational devices such as space-bounded Turing machines [24], two-way multi-head finite automata [3, 29], one-way multi-head finite automata [17], queue automata [20], and in particular, pushdown automata [16]. In contrast to Turing machines it turned out that the family of languages accepted by reversible pushdown automata or reversible finite automata are proper subsets of the general families.

Here we focus on the essence of both lines of research described. So, we are interested in languages that share both properties, namely, to be reversible and input driven. The intersection of both worlds can be defined on the underlying language families or on the underlying machine classes. It turns out that this makes a difference in the sense that the latter class is properly included in the former. Moreover, the former class is closed under complementation while the latter is not. So, it is natural to consider the complementary class as well. Together with the union and intersection of incomparable families we obtain a hierarchical set structure within the family of real-time deterministic context-free languages as depicted in Figure 1. The innermost class which is defined as the family of languages so that the language itself as well as its complement are accepted by real-time deterministic pushdown automata that are input driven and reversible at the same time, is characterized by the regular languages. So, *any* non-regular language which is accepted by a pushdown automaton having both properties under investigation is a witness for the non-closure of that family under complementation.

Following the definition in [1], the input-driven pushdown automata considered so far have the property that tacitly a new bottom-of-pushdown symbol is placed into the stack when the stack gets empty by a pop operation. This definition is different from the general definition of pushdown automata saying that a computation halts whenever the last pushdown symbol has been removed from the stack. Thus, we introduce in Section 4 input-driven automata with this restriction. Again, the intersection of both worlds can be defined on the underlying language families or on the underlying machine classes. Thus, we investigate the relations between these classes and between the classes induced by unrestricted input-driven pushdown automata. As result we obtain that the different restricted and unrestricted variants induce proper inclusions between the corresponding language families and the restricted variants imply a hierarchical set structure (see Fig. 2) which is similar to the hierarchies shown for the unrestricted variants in Section 3.

Finally, closure properties of all discussed language families under the standard operations are investigated. 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. The closure properties are summarized in Table 1. It turns out that all newly introduced language families are anti-AFLs, that is, they are not closed under the operations union, concatenation, Kleene star, length-preserving homomorphism, inverse homomorphism, and intersection with regular languages.

## 2. PRELIMINARIES

We write  $\Sigma^*$  for the set of all *words* over the finite alphabet  $\Sigma$ . The *empty word* is denoted by  $\lambda$ , the *reversal* of a word  $w$  by  $w^R$ , and for the *length* of  $w$  we write  $|w|$ . The number of occurrences of a symbol  $a \in \Sigma$  in some word  $w$  is denoted by  $|w|_a$ . We use  $\subseteq$  for *inclusions* and  $\subset$  for *strict inclusions*.

In order to define the particularities of input-driven and reversible pushdown automata, we first consider the general devices. General deterministic pushdown automata that are not allowed to perform  $\lambda$ -steps are weaker than deterministic pushdown automata that may move on  $\lambda$  input [10]. However, in [16] it has been shown that every reversible pushdown automaton can be simulated by a *real-time* reversible pushdown automaton, that is,

without  $\lambda$ -steps. This real-time reversible machine can effectively be constructed from the given one. Moreover, input-driven pushdown automata are real-time devices by definition. For this reason and to simplify matters we do not allow  $\lambda$ -steps from the outset.

A *real-time deterministic pushdown automaton* (DPDA) is defined as a system  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$ , where  $Q$  is the finite set of *internal states*,  $\Sigma$  is the finite set of *input symbols*,  $\Gamma$  is the finite set of *pushdown symbols*,  $F \subseteq Q$  is the set of *accepting states*,  $q_0 \in Q$  is the *initial state*,  $\perp \in \Gamma$  is a distinguished pushdown symbol, called the *bottom-of-stack symbol*, which initially appears on the stack, and the (possibly partial) *transition function*  $\delta$  maps  $Q \times \Sigma \times \Gamma$  to  $Q \times \Gamma^*$ .

A *configuration* of a pushdown automaton is a quadruple  $(u, q, v, \gamma)$ , where  $q$  is the current state,  $u \in \Sigma^*$  is the part of the input to the left of the input head,  $v \in \Sigma^*$  is the part of the input to the right of the input head, and  $\gamma \in \Gamma^*$  is the current content of the pushdown store, the leftmost symbol of  $\gamma$  being the top symbol. So, the input head is positioned at the border between the last symbol of  $u$  and the first symbol of  $v$ . On input  $w$  the initial configuration is defined to be  $(\lambda, q_0, w, \perp)$ . For  $p, q \in Q$ ,  $a \in \Sigma$ ,  $u, v \in \Sigma^*$ ,  $\beta, \gamma \in \Gamma^*$ , and  $Z \in \Gamma$ , let  $(u, p, av, Z\gamma)$  be a configuration. Then its *successor configuration* is  $(ua, q, v, \beta\gamma)$ , where  $\delta(p, a, Z) = (q, \beta)$ . We write  $(u, p, av, Z\gamma) \vdash (ua, q, v, \beta\gamma)$  in this case. The reflexive transitive closure of  $\vdash$  is denoted by  $\vdash^*$ .

To simplify matters, we require that in any configuration the bottom-of-pushdown symbol appears at most once at the bottom of the pushdown store, that is, it can never appear at some other position in the pushdown store. Formally, we require that if  $\delta(p, a, Z) = (q, \beta)$  then either  $Z \neq \perp$  and  $\beta$  does not contain  $\perp$ , or  $Z = \perp$  and  $\beta$  is either  $\beta'\perp$ , where  $\beta'$  does not contain  $\perp$ , or  $\beta$  is empty. The *language accepted* by  $M$  with accepting states is

$$L(M) = \{ w \in \Sigma^* \mid (\lambda, q_0, w, \perp) \vdash^* (w, q, \lambda, \gamma), \text{ for some } q \in F \text{ and } \gamma \in \Gamma^* \}.$$

In general, the family of all languages accepted by some device of type X is denoted by  $\mathcal{L}(X)$ .

#### *Input-Driven Pushdown Automata*

A real-time 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  $\Sigma$  to be partitioned into the sets  $\Sigma_N$ ,  $\Sigma_D$ , and  $\Sigma_R$ , that control the actions state change only (N), push (D), and pop (R).

So, a DPDA  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  is said to be a *deterministic input-driven pushdown automaton* (IDPDA) if  $\Sigma$  is partitioned into the sets  $\Sigma_D$ ,  $\Sigma_R$ , and  $\Sigma_N$ , and the transition function  $\delta$  is accordingly split into  $\delta_D$ ,  $\delta_R$ , and  $\delta_N$ . Here it is understood that, for  $p \in Q$ ,  $a \in \Sigma$ ,  $u, v \in \Sigma^*$ ,  $Z, Z' \in \Gamma$ ,  $Z'' \in \Gamma \setminus \{\perp\}$ , and  $\gamma \in \Gamma^*$ , the only possibilities to obtain successor configurations are as follows:

- (1)  $(u, p, av, Z\gamma) \vdash (ua, q, v, Z'Z\gamma)$ , if and only if  $a \in \Sigma_D$  and  $\delta_D(p, a, Z) = (q, Z'Z)$ ,
- (2)  $(u, p, av, Z''\gamma) \vdash (ua, q, v, \gamma)$ , if and only if  $a \in \Sigma_R$  and  $\delta_R(p, a, Z'') = (q, \lambda)$ ,
- (3)  $(u, p, av, \perp) \vdash (ua, q, v, \perp)$ , if and only if  $a \in \Sigma_R$  and  $\delta_R(p, a, \perp) = (q, \lambda)$ ,
- (4)  $(u, p, av, Z\gamma) \vdash (ua, q, v, Z\gamma)$ , if and only if  $a \in \Sigma_N$  and  $\delta_N(p, a, Z) = (q, Z)$ .

In particular, this means that at every step at most one symbol can be pushed on the stack, that the topmost stack symbol cannot be altered, and that tacitly a new bottom-of-pushdown symbol is placed into the stack when the stack gets empty by a pop operation.

The partition of an input alphabet into the sets  $\Sigma_D$ ,  $\Sigma_R$ , and  $\Sigma_N$  is called a *signature*.

#### *Reversible Pushdown Automata*

Now we turn to reversible pushdown automata which have been introduced and studied in [16]. Reversibility is meant with respect to the possibility of stepping the computation back and forth. To this end, the pushdown automata have also to be backward deterministic. That is, any configuration must have at most one predecessor which, in addition, is computable by a DPDA. For reverse computation steps the head of the input tape is always

moved to the *left*. Therefore, the automaton rereads the input symbol which has been read in a preceding forward step. So, for reversible pushdown automata there must exist a reverse transition function.

A reverse transition function  $\delta^- : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$  maps a configuration to its *predecessor configuration*. For  $p, q \in Q$ ,  $a \in \Sigma$ ,  $u, v \in \Sigma^*$ ,  $Z \in \Gamma$ , and  $\beta, \gamma \in \Gamma^*$ , let  $(ua, q, v, Z\gamma)$  be a configuration. Then its *predecessor configuration* is  $(u, p, av, \beta\gamma)$ , where  $\delta^-(q, a, Z) = (p, \beta)$ . We write  $(ua, q, v, Z\gamma) \vdash^- (u, p, av, \beta\gamma)$  in this case.

A DPDA  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  is said to be *reversible* (REV-PDA), if there exists a reverse transition function  $\delta^-$  inducing a relation  $\vdash^-$  from one configuration to the next, so that

$$(u, p, v, \gamma) \vdash^- (u', p', v', \gamma') \text{ if and only if } (u', p', v', \gamma') \vdash (u, p, v, \gamma).$$

The following mandatory properties of reversible pushdown automata have been derived in [16]. In one reverse step the height of the pushdown store can be decreased by at most one. Therefore, in a forward step the height of the pushdown store may be increased by at most one, as well. Furthermore, when a forward step pops a symbol, this operation simply reveals the next-to-top symbol. Therefore, one has to take care that the original top-of-stack symbol remains unaltered in a forward step in which the height of the pushdown is increased. Thus, we have: If  $\delta(p, a, Z) = (q, \beta)$  and  $|\beta| > 1$ , then  $\beta = YZ$  for some symbol  $Y \in \Gamma$ . So, for a reversible DPDA there are only the following possibilities:

$$\begin{array}{ll} \text{push:} & \delta(p, a, Z) = (q, Z'Z) \implies \delta^-(q, a, Z') = (p, \lambda) \\ \text{change top:} & \delta(p, a, Z) = (q, Z') \implies \delta^-(q, a, Z') = (p, Z) \\ \text{pop:} & \delta(p, a, Z) = (q, \lambda) \implies \text{for all } X \in \Gamma: \delta^-(q, a, X) = (p, ZX) \end{array}$$

Finally, pushdown automata that have both properties at the same time, that is, which are reversible *and* input driven, are denoted by REV-IDPDA. Since here we are interested in the essence of input-driven and reversible context-free languages, it is natural to consider their intersection. However, this intersection can be defined on the underlying language families or on the underlying machines classes. So, we set  $L_{\text{ri}} = \mathcal{L}(\text{REV-PDA}) \cap \mathcal{L}(\text{IDPDA})$  which is the intersection of the both languages families, and  $M_{\text{ri}} = \mathcal{L}(\text{REV-IDPDA})$  which are the languages accepted by the devices from the intersection of both machine classes. The question whether these two definitions of reversible input-driven context-free languages yield the same families or not is considered in the next section.

In order to clarify our notions we continue with a simple example.

**Example 2.1.** The context-free language  $L = \{a^n \$ b^n \mid n \geq 0\}$  is accepted by the REV-IDPDA  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  with  $Q = \{q_0, q_1, q_2, q_+\}$ ,  $F = \{q_+\}$ ,  $\Gamma = \{A, \bar{A}, \perp\}$ ,  $\Sigma_D = \{a\}$ ,  $\Sigma_R = \{b\}$ , and  $\Sigma_N = \{\$\}$ . The transition functions  $\delta$  and its reverse  $\delta^-$  are as follows.

REV-IDPDA forward		REV-IDPDA backward	
(1)	$\delta(q_0, a, \perp) = (q_1, A\perp)$	(1)	$\delta^-(q_1, a, A) = (q_0, \lambda)$
(2)	$\delta(q_0, a, A) = (q_1, \bar{A}A)$		
(3)	$\delta(q_0, a, \bar{A}) = (q_1, \bar{A}\bar{A})$		
(4)	$\delta(q_0, \$, \perp) = (q_+, \perp)$	(2)	$\delta^-(q_+, \$, \perp) = (q_0, \perp)$
(5)	$\delta(q_1, a, \perp) = (q_1, A\perp)$	(3)	$\delta^-(q_1, a, A) = (q_1, \lambda)$
(6)	$\delta(q_1, a, A) = (q_1, \bar{A}A)$		
(7)	$\delta(q_1, a, \bar{A}) = (q_1, \bar{A}\bar{A})$		
(8)	$\delta(q_1, \$, A) = (q_2, \bar{A})$	(4)	$\delta^-(q_2, \$, \bar{A}) = (q_1, \bar{A})$
(9)	$\delta(q_1, \$, \bar{A}) = (q_2, A)$	(5)	$\delta^-(q_2, \$, A) = (q_1, A)$
(10)	$\delta(q_2, b, A) = (q_2, \lambda)$	(6)	$\delta^-(q_2, b, \perp) = (q_2, A\perp)$
		(7)	$\delta^-(q_2, b, A) = (q_2, \bar{A}A)$
		(8)	$\delta^-(q_2, b, \bar{A}) = (q_2, \bar{A}\bar{A})$
(11)	$\delta(q_2, b, \bar{A}) = (q_+, \lambda)$	(9)	$\delta^-(q_+, b, \perp) = (q_2, \bar{A}\perp)$
		(10)	$\delta^-(q_+, b, A) = (q_2, \bar{A}A)$
		(11)	$\delta^-(q_+, b, \bar{A}) = (q_2, \bar{A}\bar{A})$

The basic idea of the construction is clear, the prefix is pushed on the stack and compared with the suffix. However, in order to be input driven *and* reversible, the first symbol pushed has to be marked ( $\bar{A}$ ). Moreover, to obtain a reversible automaton some transition rules have to be provided that cannot be used in any reachable configuration. For example, the forward transitions (2) and (3) are used only for unreachable configurations. Let the REV-IDPDA perform the transition  $(\lambda, q_0, a, \perp) \vdash (a, q_1, \lambda, \bar{A}\perp)$ . Then the reverse transition rule  $\delta^-(q_1, a, \bar{A}) = (q_0, \lambda)$  has to be defined. However, applying this rule to the unreachable configuration  $(a, q_1, \lambda, \bar{A}\bar{A})$  gives the predecessor configuration  $(\lambda, q_0, a, \bar{A})$ . This implies that the (forward) transition rule  $\delta(q_0, a, \bar{A}) = (q_1, \bar{A}\bar{A})$  has to be defined as well.

### 3. ZOOMING INTO THE DETERMINISTIC CONTEXT-FREE LANGUAGES

In this section, we start to explore the relationships between the language families considered. Since they all are defined by classes of real-time deterministic pushdown automata, we obtain a set structure within the family  $\mathcal{L}(\text{DPDA})$  (see Fig. 1). Starting with the superclass defined by the intersection of the context-free and Church–Rosser languages (see [13]), we obtain the proper inclusion of the deterministic context-free languages (DCFL). The family of Church–Rosser languages as well as the family of context-free languages contain DCFL and are closed under reversal. Since DCFL is not closed under reversal the proper inclusion follows. Furthermore, real-time deterministic pushdown automata are well known to be weaker than deterministic pushdown automata that may move on  $\lambda$  input [10].

When diving into the family of real-time deterministic pushdown automata one first sees the families  $\mathcal{L}(\text{IDPDA})$  and  $\mathcal{L}(\text{REV-PDA})$ . Both are known to be properly included in  $\mathcal{L}(\text{DPDA})$ . For example, it is known that the following deterministic context-free languages are not accepted by any input-driven pushdown automaton

$$\begin{aligned} L_d &= \{ a^n b^{2n} \mid n \geq 0 \}, & L'_d &= \{ a^n \$ b^{2n} \mid n \geq 0 \}, \\ L_r &= \{ w \$ w^R \mid w \in \{a, b\}^* \}, & \text{and } L_s &= \{ c^n \$ c^n \mid n \geq 0 \} \end{aligned}$$

while  $L_c = \{ a^n b^n \mid n \geq 0 \}$  as well as its marked variant  $L'_c = \{ a^n \$ b^n \mid n \geq 0 \}$  are accepted by some input-driven pushdown automaton [1]. Since Example 2.1 shows that  $L'_c$  belongs to  $M_{ri}$ , we know that  $M_{ri}$  is a proper superset of the regular languages. Further it is known that the language  $L_c$  is not accepted by a reversible pushdown automaton while  $L'_c$  and  $L_r$  are [16]. We derive the following corollary.

**Corollary 3.1.** *The families  $\mathcal{L}(\text{REV-PDA})$  and  $\mathcal{L}(\text{IDPDA})$  are incomparable.*

So, the families of languages accepted by input-driven pushdown automata and by reversible pushdown automata are properly contained in the family of real-time deterministic context-free languages. But how about their union? Is there a real-time deterministic context-free language which is neither input driven nor reversible? The next theorem answers this question in the affirmative.

**Theorem 3.2.** *The union  $\mathcal{L}(\text{IDPDA}) \cup \mathcal{L}(\text{REV-PDA})$  is properly contained in the family of real-time deterministic context-free languages.*

*Proof.* Consider the language  $L = \{ a^n b^n \mid n \geq 0 \} \cup \{ c^m \$ c^m \mid m \geq 0 \}$ , that is clearly accepted by some real-time deterministic pushdown automaton.

The family of languages accepted by input-driven pushdown automata is closed under intersection with regular sets [1]. The intersection  $L \cap \{c, \$\}^*$  is exactly  $L_s$  for which it is known that it is not input driven. The family of languages accepted by reversible pushdown automata is closed under intersection with reversible regular sets, for example, with  $\{a, b\}^*$ . The intersection  $L \cap \{a, b\}^*$  gives the language  $L_c$  that cannot be accepted by any reversible pushdown automaton.  $\square$

Next we turn to the essence of both worlds and first consider the family of languages that are both, input-driven and reversible. As mentioned before, the intersection can be defined on the underlying machine classes

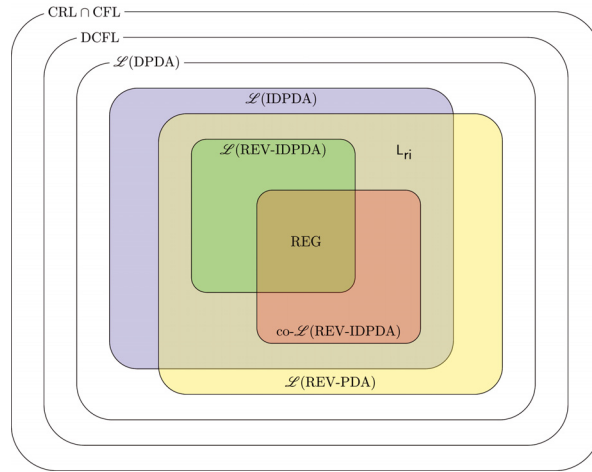


FIGURE 1. Hierarchical structure of language classes. The class  $CRL \cap CFL$  denotes the intersection of the context-free and Church–Rosser languages – a proper superclass of the deterministic context-free languages (DCFL) which, in turn, form a proper superclass of the real-time deterministic context-free languages ( $\mathcal{L}(DPDA)$ ). (Color online).

or on the underlying language families. So, we study now the relationships of  $M_{ri} = \mathcal{L}(REV-IDPDA)$  and  $L_{ri} = \mathcal{L}(REV-PDA) \cap \mathcal{L}(IDPDA)$ . It turns out that these definitions yield different families, where the latter one is a weaker restriction.

Since all the superclasses of  $M_{ri}$ , that is, the classes  $L_{ri}$ ,  $\mathcal{L}(IDPDA)$ ,  $\mathcal{L}(DPDA)$ ,  $\mathcal{L}(REV-PDA)$ , and DCFL, are closed under complementation, but  $M_{ri}$  turns out to be not, it is natural to consider the family  $co-\mathcal{L}(REV-IDPDA)$  as well. First we show that the union  $\mathcal{L}(REV-IDPDA) \cup co-\mathcal{L}(REV-IDPDA)$  is properly contained in  $L_{ri}$ .

**Theorem 3.3.** *There is a language in  $L_{ri}$  that does not belong to the union of the classes  $\mathcal{L}(REV-IDPDA) \cup co-\mathcal{L}(REV-IDPDA)$ . Therefore, language family  $\mathcal{L}(REV-IDPDA) \cup co-\mathcal{L}(REV-IDPDA)$  is properly contained in the family  $L_{ri}$ .*

*Proof.* The language  $L = \{ a^n \$ b^n a^\ell b^m \mid \ell, m, n \geq 0 \}$  is a witness for the assertion. First assume  $L$  is accepted by some REV-IDPDA  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  with signature  $\Sigma_D, \Sigma_R, \Sigma_N$ . If symbol  $a$  does not belong to  $\Sigma_D$ , automaton  $M$  runs into a loop for input prefixes of the form  $a^*$  long enough and, thus, cannot accept  $L$ . Therefore, we derive  $a \in \Sigma_D$ . If  $\Sigma_R$  is empty or contains  $\$$  only, automaton  $M$  cannot utilize its pushdown store sufficiently and, thus,  $L$  must be regular. Therefore, we have  $b \in \Sigma_R$ .

We consider the computation of  $M$  on input  $a\$bab^m$ , where  $m \geq |Q|+2$ . After processing the input prefix  $a\$ba$ , the pushdown store of  $M$  contains a word whose length is two or one dependent on whether the  $\$$  belongs to  $\Sigma_D$  or not. Now  $M$  starts to read the symbols  $b$  whereby it pops one symbol in every step. So, after at most two steps the pushdown store gets empty. After at most  $|Q|$  further steps on input symbols  $b$ , automaton  $M$  runs into a loop whose length is at most  $|Q|$ . Since  $M$  is reversible, the reverse transition function drives  $M$  backwards through the loop and continues this behavior until the symbol  $a$  appears in the input. However, this configuration has an empty pushdown store and, thus, does not match the configuration from the forward computation. So,  $M$  does not accept  $L$ .

Next we assume that  $\bar{L}$  is accepted by the REV-IDPDA  $M$ . Arguing similar as above, if either  $\Sigma_D$  or  $\Sigma_R$  is empty or contains  $\$$  only, automaton  $M$  cannot utilize its pushdown store sufficiently and, thus,  $L$  must be regular. Since it is not, both sets  $\Sigma_D$  and  $\Sigma_R$  contain at least one symbol, say  $a \in \Sigma_D$  and  $b \in \Sigma_R$ . We consider the computation of  $M$  on input  $\$bab^m$ , where  $m \geq |Q|+1$ , and obtain the same contradiction as above. Finally, if  $a \in \Sigma_R$  and  $b \in \Sigma_D$ , the input  $\$ba^m$ , where  $m \geq |Q|+2$  yields the contradiction.

It remains to be shown that  $L$  belongs to the family  $L_{ri}$ . The IDPDA with signature  $\Sigma_D = \{a\}$ ,  $\Sigma_R = \{b, \$\}$ ,  $\Sigma_N = \emptyset$ , and  $F = \{q_2, q_3\}$  which is given through the following transition function accepts  $L$ . For example, the transition rules (11) and (12) are irreversible.

IDPDA					
(1)	$\delta(q_0, a, \perp)$	$=$	$(q_0, A\perp)$	(7)	$\delta(q_2, a, \perp) = (q_2, A\perp)$
(2)	$\delta(q_0, a, A)$	$=$	$(q_0, AA)$	(8)	$\delta(q_2, a, A) = (q_2, AA)$
(3)	$\delta(q_0, \$, \perp)$	$=$	$(q_2, \lambda)$	(9)	$\delta(q_2, b, \perp) = (q_3, \lambda)$
(4)	$\delta(q_0, \$, A)$	$=$	$(q_1, \lambda)$	(10)	$\delta(q_2, b, A) = (q_3, \lambda)$
(5)	$\delta(q_1, b, A)$	$=$	$(q_1, \lambda)$	(11)	$\delta(q_3, b, \perp) = (q_3, \lambda)$
(6)	$\delta(q_1, b, \perp)$	$=$	$(q_2, \lambda)$	(12)	$\delta(q_3, b, A) = (q_3, \lambda)$

An equivalent REV-PDA with  $F = \{q_2, q_3\}$  and  $\Gamma = \{\perp, A, Q_2, Q_3\}$  is given through the following transition functions. The reversibility is easily checked by inspecting the transition rules. The reversibility on the suffix is ensured by pushing the history on the stack. The corresponding transition rules, for example (4) and (7), violate the property of being input driven. Let  $X \in \Gamma$ .

REV-PDA forward			REV-PDA backward		
(1)	$\delta(q_0, a, X)$	$=$	$(q_0, AX)$	(1)	$\delta^-(q_0, a, A) = (q_0, \lambda)$
(2)	$\delta(q_0, \$, \perp)$	$=$	$(q_2, \perp)$	(2)	$\delta^-(q_2, \$, \perp) = (q_0, \perp)$
(3)	$\delta(q_0, \$, A)$	$=$	$(q_1, \lambda)$	(3)	$\delta^-(q_1, \$, X) = (q_0, AX)$
(4)	$\delta(q_1, b, A)$	$=$	$(q_1, \lambda)$	(4)	$\delta^-(q_1, b, X) = (q_1, AX)$
(5)	$\delta(q_1, b, \perp)$	$=$	$(q_2, \perp)$	(5)	$\delta^-(q_2, b, \perp) = (q_1, \perp)$
(6)	$\delta(q_2, a, X)$	$=$	$(q_2, Q_2X)$	(6)	$\delta^-(q_2, a, Q_2) = (q_2, \lambda)$
(7)	$\delta(q_2, b, X)$	$=$	$(q_3, Q_2X)$	(7)	$\delta^-(q_3, b, Q_2) = (q_2, \lambda)$
(8)	$\delta(q_3, b, X)$	$=$	$(q_3, Q_3X)$	(8)	$\delta^-(q_3, b, Q_3) = (q_3, \lambda)$

□

**Corollary 3.4.** *The family  $M_{ri}$  is properly contained in  $L_{ri}$ .*

Next we consider the innermost class, that is, the intersection of the classes  $\mathcal{L}(\text{REV-IDPDA})$  and its complement  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$ . The next theorem shows that this intersection characterizes the regular languages.

**Theorem 3.5.** *A language belongs to  $\mathcal{L}(\text{REV-IDPDA}) \cap \text{co-}\mathcal{L}(\text{REV-IDPDA})$  if and only if it is regular.*

*Proof.* Assume in contrast to the assertion that there is a non-regular language  $L$  belonging to  $\mathcal{L}(\text{REV-IDPDA}) \cap \text{co-}\mathcal{L}(\text{REV-IDPDA})$ . Since the regular languages are closed under complementation,  $\overline{L}$  is non-regular as well. Moreover, since all unary context-free languages are regular, the alphabet of  $L$  contains at least two symbols. Now, let  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  with signature  $\Sigma_D, \Sigma_R, \Sigma_N$  be a REV-IDPDA accepting  $L$ . If either  $\Sigma_D$  or  $\Sigma_R$  is empty, automaton  $M$  cannot utilize its pushdown store, thus  $L$  must be regular. So, both sets  $\Sigma_D$  and  $\Sigma_R$  contain at least one symbol, say  $b \in \Sigma_R$ . We consider the computation of  $M$  on some input  $vb^m$ , where  $v \in \Sigma_D^*$  with  $|v| \geq 1$  and  $m \geq |v| + 3|Q|$ . After processing the input prefix  $v$ , the pushdown store of  $M$  contains a word  $\gamma_1$  whose length is  $|v|$ :

$$(\lambda, q_0, vb^m, \perp) \vdash^* (v, q_1, b^m, \gamma_1 \perp).$$

Now  $M$  starts to read the symbols  $b$  whereby it pops one symbol in every step. So, after another  $|v|$  steps the pushdown store gets empty:

$$(v, q_1, b^m, \gamma_1 \perp) \vdash^* (vb^{|v|}, q_2, b^{m-|v|}, \perp).$$

After at most  $|Q|$  further steps on input symbols  $b$ , automaton  $M$  runs into a loop whose length is also at most  $|Q|$ :

$$(vb^{|v|}, q_2, b^{m-|v|}, \perp) \vdash^* (vb^{|v|+i}, q_3, b^{m-|v|-i}, \perp) \vdash^* (vb^{|v|+i+j}, q_3, b^{m-|v|-i-j}, \perp)$$

where  $0 \leq i \leq |Q|$  and  $1 \leq j \leq |Q|$ . Since  $M$  is reversible, the reverse transition function drives  $M$  backwards through the loop and continues this behavior until the last symbol of  $v$  appears in the input:

$$(vb^{|v|+i+j}, q_3, b^{m-|v|-i-j}, \perp) \vdash^{-*} (vb^{|v|+i}, q_3, b^{m-|v|-i}, \perp) \vdash^{-*} (v, q_4, b^m, \perp)$$

where  $q_4$  is some state in the loop. However, this configuration has an empty pushdown store and, thus, does not match the configuration from the forward computation. This implies that  $M$  is unable to perform the computation on the inputs of the form considered and has to halt rejecting before it would enter the loop. We conclude that *all* words having one of these inputs as prefix do belong to  $\bar{L}$ .

Next, consider a REV-IDPDA  $M' = \langle Q', \Sigma, \Gamma', F', q'_0, \perp, \delta' \rangle$  with signature  $\Sigma'_D, \Sigma'_R, \Sigma'_N$  that accepts  $\bar{L}$ . Since  $\bar{L}$  is not regular, we have as above that both sets  $\Sigma'_D$  and  $\Sigma'_R$  contain at least one symbol. Let  $u = vb^m$  be one of the words on which  $M$  halts rejecting. We consider inputs of the form  $uv'x^k$ , where  $x \in \Sigma'_R$  and  $v' \in \Sigma'^*_D$  with  $|v| \geq 1$  and  $k \geq |uv'| + 3|Q'|$ . Along the lines of the argumentation for  $M$  we obtain that  $M'$  has to halt rejecting on all words having a prefix of this form. This implies that these words are neither accepted by  $M$  nor by  $M'$ . From this contradiction we derive that  $L$  must be regular.

Finally, every regular language is accepted by some REV-IDPDA. The idea of the construction is to simulate a given deterministic finite automaton whereby the state history is pushed on the stack.  $\square$

Interestingly, the last result reveals that *any* non-regular language  $L$  which is accepted by a pushdown automaton that is input driven and reversible at the same time, is a witness for the non-closure of  $M_{ri}$  under complementation.

**Theorem 3.6.** *The families  $\mathcal{L}(\text{REV-IDPDA})$  and  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$  are incomparable.*

*Proof.* The language  $L = \{a^n b^n \mid n \geq 0\}$  is not regular and, thus, does not belong to the intersection  $\mathcal{L}(\text{REV-IDPDA}) \cap \text{co-}\mathcal{L}(\text{REV-IDPDA})$  by Theorem 3.5. On the other hand, language  $L$  belongs to the family  $\mathcal{L}(\text{REV-IDPDA})$  by Example 2.1. So, we can conclude  $L \in \mathcal{L}(\text{REV-IDPDA}) \setminus \text{co-}\mathcal{L}(\text{REV-IDPDA})$  and  $\bar{L} \in \text{co-}\mathcal{L}(\text{REV-IDPDA}) \setminus \mathcal{L}(\text{REV-IDPDA})$ .  $\square$

#### 4. ZOOMING MORE CLOSELY INTO INPUT-DRIVEN LANGUAGES

The definition of input-driven pushdown automata given in Section 2 implies that the stack never gets empty during any computation since tacitly a new bottom-of-pushdown symbol is placed into the stack when the stack would get empty by a pop operation. This definition is very convenient, since it allows to accept languages such as  $\{a^n b^m \mid m > n \geq 0\}$  or  $\{a^n b^n a^\ell b^m \mid \ell, m, n \geq 0\}$  by IDPDA. However, in the standard definition of pushdown automata a computation halts when the bottom-of-pushdown symbol has been removed by a pop operation. In this section, we consider IDPDA and their restricted variants introduced and investigated in the previous sections with the additional restriction that a computation halts as soon as the bottom-of-pushdown symbol is removed. If the complete input has not been read up to that time or a non-accepting state is entered, the input is rejected. IDPDA with this restrictions are denoted  $\text{IDPDA}_0$  and by  $\text{REV-IDPDA}_0$  we denote  $\text{IDPDA}_0$  which are additionally reversible. The language family which contains all languages which are either accepted by some  $\text{IDPDA}_0$  or some  $\text{REV-IDPDA}_0$  is denoted by  $L_{ri,0}$ . We will also use the notion  $M_{ri,0}$  for the family  $\mathcal{L}(\text{REV-IDPDA}_0)$ . Before we investigate and compare the computational capacity of these new language families we prove the following lemma which we will need in the sequel.

**Lemma 4.1.** *Language family  $\mathcal{L}(\text{IDPDA}_0)$  is closed under intersection with regular sets.*



*Proof.* Consider an IDPDA<sub>0</sub>  $M$  and a deterministic finite automaton  $A$ . An IDPDA<sub>0</sub>  $M'$  accepting  $L(M) \cap L(A)$  is obtained by the standard cross product between the finite state controls of  $M$  and  $A$ . The resulting set of states simulates the computation of  $M$  in the first component while the second replicates the behavior of  $A$ . The moves on the stack are governed by the moves of  $M$ , since  $A$  does not use any memory storage. Finally, the accepting states of  $M'$  are exactly those pairs where both components are accepting states of  $M$  and  $A$ . Since the stack gets empty in any accepting computation of  $M$  at most in the very last step, the same is true for any (parallel) accepting computation of  $M$  and  $A$ . Thus,  $M'$  is again an IDPDA<sub>0</sub> which shows the assertion of the Lemma.  $\square$

As first result with respect to the computational capacity we obtain that all variants of IDPDA<sub>0</sub> are less powerful than their unrestricted counterparts.

**Theorem 4.2.** *The following proper inclusions hold.*

- (1)  $\mathcal{L}(\text{IDPDA}_0) \subset \mathcal{L}(\text{IDPDA})$ ,
- (2)  $L_{ri,0} \subset L_{ri}$ ,
- (3)  $\mathcal{L}(\text{REV-IDPDA}_0) \subset \mathcal{L}(\text{REV-IDPDA})$ , and
- (4)  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0) \subset \text{co-}\mathcal{L}(\text{REV-IDPDA})$ .

*Proof.* All inclusions claimed follow by structural reasons. The first two proper inclusions can be shown using the witness language  $L = \{a^n \$ b^m \mid m > n \geq 0\}$ . Clearly,  $L$  can be accepted by some IDPDA. Furthermore,  $L$  is accepted by some REV-PDA that first checks the correctness of a prefix  $a^n \$ b^n$  using the construction given in Example 2.1 and then pushes some new symbol on its pushdown store for every additional  $b$  read. Thus,  $L$  belongs to  $L_{ri}$ . Let us assume that  $L$  is accepted by some IDPDA<sub>0</sub>  $M$  with signature  $\Sigma_D$ ,  $\Sigma_R$ , and  $\Sigma_N$ . Similar to the proof of Theorem 3.3 we may assume that  $a \in \Sigma_D$  and  $b \in \Sigma_R$ . Thus, the word  $a^n \$ b^{n+3} \in L$  brings  $M$  to halt before the complete input is read, since the stack gets empty. Therefore, this word is not accepted by  $M$  which is a contradiction that shows the first two inclusions to be proper.

For the properness of the third inclusion consider  $L' = \{a^n \$ b^{n+3} \mid n \geq 0\}$  that can be shown not to belong to  $\mathcal{L}(\text{IDPDA}_0)$  in a similar way as the above proof for the language  $L$ . On the other hand,  $L'$  can be accepted by a REV-IDPDA as follows. First, the construction of Example 2.1 accepting a prefix of the form  $a^n \$ b^n$  is applied. Then, from state  $q_+$  three further input symbols are read while entering three additional states from which the last one is the only accepting state. The pushdown store remains  $\perp$  in these three steps. Clearly, this construction gives an input-driven and reversible DPDA and shows the properness of the third inclusion.

Finally, since  $L'$  belongs to  $\mathcal{L}(\text{REV-IDPDA})$ , the complement  $\overline{L'}$  belongs to  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$ . Now, assume that the complement  $\overline{L'}$  belongs to the family  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$ . Then  $L'$  belongs to  $\mathcal{L}(\text{REV-IDPDA}_0)$  which is a contradiction.  $\square$

Next, we show that it is again a difference whether the family of languages that are input-driven and reversible is defined on the underlying language families or on the underlying machine classes. It turns out that the latter language family is a proper subset of the former.

**Theorem 4.3.** *Language family  $\mathcal{L}(\text{REV-IDPDA}_0)$  is properly included in  $L_{ri,0}$ .*

*Proof.* The inclusion follows for structural reasons. The witness language that separates both classes is  $L = \{a^n \$ b^n c^\ell (ab)^m \mid \ell, m, n \geq 0\}$ . Language  $L$  belongs to  $L_{ri,0}$  since we can construct an IDPDA<sub>0</sub> as well as a REV-PDA accepting  $L$ . An IDPDA<sub>0</sub> for  $L$  first checks the correctness of the prefix  $a^n \$ b^n$  as in previous constructions and then the regular suffix  $c^*(ab)^*$ . Since every  $b \in \Sigma_R$  follows an  $a \in \Sigma_D$ , it is clear that the pushdown store never gets empty. Thus,  $L$  is accepted by an IDPDA<sub>0</sub>. A REV-PDA for  $L$  checks the correctness of the prefix  $a^n \$ b^n$  as in previous constructions and the correctness of the suffix by simulating a deterministic finite automaton whose history is written on the stack.

Next, we show that  $L$  even does not belong to  $\mathcal{L}(\text{REV-IDPDA})$ . Contrarily we assume that  $L$  is accepted by a REV-IDPDA  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  with signature  $\Sigma_D, \Sigma_R, \Sigma_N$ . As in Theorem 3.3 we can conclude

that  $a \in \Sigma_D$  and  $b \in \Sigma_R$ . Let us assume that  $c \in \Sigma_D \cup \Sigma_N$ . Then there exist two natural numbers  $n_2 > n_1$  such that the computations on input  $a\$bc^{n_1}$  and  $a\$bc^{n_2}$  end in the same accepting state  $p_0$  with the same topmost pushdown symbol  $C \in \Gamma$ . Thus, we have  $\gamma_1, \gamma_2 \in \Gamma^*$  such that  $(\lambda, q_0, a\$bc^{n_1}, \perp) \vdash^* (a\$bc^{n_1}, p_0, \lambda, C\gamma_1\perp)$  and  $(\lambda, q_0, a\$bc^{n_2}, \perp) \vdash^* (a\$bc^{n_2}, p_0, \lambda, C\gamma_2\perp)$ . If  $c \in \Sigma_R$ , we can simply replace  $C\gamma_1$  and  $C\gamma_2$  by  $\lambda$  in the rest of the proof. Now, we consider the computation on further input  $(ab)^*$ . Since  $a \in \Sigma_D$  pushes some pushdown symbol which is immediately removed by  $b \in \Sigma_R$ , we know that the topmost pushdown symbol  $C$  remains the same for any further input from  $(ab)^*$ . Thus:

$$(\lambda, p_0, (ab)^n, C\gamma_1\perp) \vdash^2 (ab, p_1, (ab)^{n-1}, C\gamma_1\perp) \vdash^2 ((ab)^2, p_2, (ab)^{n-2}, C\gamma_1\perp) \vdash^2 \dots$$

For  $n$  large enough, the computation will enter a loop, that is, there are integers  $0 \leq i < j$  such that  $p_j = p_i$ . Since  $M$  is reversible, we know that  $i = 0$  and  $(\lambda, p_0, (ab)^j, C\gamma_1\perp) \vdash^{2j} ((ab)^j, p_0, \lambda, C\gamma_1\perp)$ . Thus, we have the following computation:

$$\begin{aligned} (\lambda, q_0, a\$bc^{n_1}(ab)^j c^{n_2-n_1}, \perp) \vdash^* (a\$bc^{n_1}, p_0, (ab)^j c^{n_2-n_1}, C\gamma_1\perp) \vdash^* \\ (a\$bc^{n_1}(ab)^j, p_0, c^{n_2-n_1}, C\gamma_1\perp) \vdash^* (a\$bc^{n_1}(ab)^j c^{n_2-n_1}, p_0, \lambda, C\gamma_2\perp). \end{aligned}$$

Since  $p_0$  is an accepting state,  $j \geq 1$ , and  $n_2 - n_1 \geq 1$ , we obtain that an input is accepted that does not belong to  $L$ . This is a contradiction and shows that  $L$  cannot be accepted by any REV-IDPDA.  $\square$

The relations between the classes  $\mathcal{L}(\text{REV-IDPDA}_0)$  and  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$  are similar to those shown for  $\mathcal{L}(\text{REV-IDPDA})$  and  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$  in Theorems 3.5 and 3.6.

**Theorem 4.4.** *The families  $\mathcal{L}(\text{REV-IDPDA}_0)$  and  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$  are incomparable, but their intersection  $\mathcal{L}(\text{REV-IDPDA}_0) \cap \text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$  equals the set of regular languages.*

*Proof.* The incomparability follows by the same argumentation as in the proof of Theorem 3.6 by using language  $L'_c = \{a^n \$b^n \mid n \geq 0\}$ . The inclusions

$$\mathcal{L}(\text{REV-IDPDA}_0) \cap \text{co-}\mathcal{L}(\text{REV-IDPDA}_0) \subseteq \mathcal{L}(\text{REV-IDPDA}) \cap \text{co-}\mathcal{L}(\text{REV-IDPDA}) \subseteq \text{REG}$$

follow for structural reasons and the result of Theorem 3.5. On the other hand, every regular language is accepted by some  $\text{REV-IDPDA}_0$  by simulating a given deterministic finite automaton whereby the state history is pushed on the stack. Thus,  $\text{REG} \subseteq \mathcal{L}(\text{REV-IDPDA}_0) \cap \text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$  which shows the second claim.  $\square$

Finally, we obtain the following incomparability results.

**Theorem 4.5.** *Language family  $L_{r_i,0}$  is incomparable with each of the families  $\mathcal{L}(\text{REV-IDPDA})$ ,  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$ , and  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$ . Moreover, family  $\mathcal{L}(\text{IDPDA}_0)$  is incomparable with each of the families  $\mathcal{L}(\text{REV-IDPDA})$ ,  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$ ,  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$ , and  $L_{r_i}$ .*

*Proof.* Language  $L_1 = \{a^n \$b^m \mid m > n \geq 0\}$  belongs to  $\mathcal{L}(\text{REV-IDPDA})$ , but not to  $L_{r_i,0}$ , since  $L_1$  does not even belong to  $\mathcal{L}(\text{IDPDA}_0)$  by the proof given for Theorem 4.2. Language  $L_2 = \{a^n \$b^m \mid 0 \leq m \leq n\}$  belongs to  $\mathcal{L}(\text{REV-IDPDA}_0)$ . Thus,  $L_3 = \overline{L_2}$  belongs to  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$  as well as to  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$ . On the other hand,  $L_3$  does not belong to  $L_{r_i,0}$ , since  $L_3$  does not even belong to  $\mathcal{L}(\text{IDPDA}_0)$ . Assume by way of contradiction that  $L_3$  belongs to  $\mathcal{L}(\text{IDPDA}_0)$ . Since  $\mathcal{L}(\text{IDPDA}_0)$  is closed under intersection with regular languages by Lemma 4.1, we obtain that  $L_3 \cap a^* \$b^* = L_1$  belongs to  $\mathcal{L}(\text{IDPDA}_0)$  which is a contradiction. Now, consider language  $L'_c = \{a^n \$b^n \mid n \geq 0\}$  which belongs to  $L_{r_i,0}$ , but neither belongs to  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$  nor to  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$  by the proof given for Theorem 3.6. Language  $L_4 = \{a^n \$b^n c^\ell (ab)^m \mid \ell, m, n \geq 0\}$  belongs to  $L_{r_i,0}$ , but not to  $\mathcal{L}(\text{REV-IDPDA})$  by the proof given for Theorem 4.3.

Language  $L_c = \{a^n b^n \mid n \geq 0\}$  belongs to  $\text{IDPDA}_0$ , but not to  $\mathcal{L}(\text{REV-IDPDA})$  which is shown in [16]. Thus,  $L_c$  does not belong to the classes  $\mathcal{L}(\text{REV-IDPDA})$ ,  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$ ,  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$ , and  $L_{r_i}$ . On the other hand,  $L_1$  belongs to  $\mathcal{L}(\text{REV-IDPDA})$  and  $L_{r_i}$ , but not to  $\mathcal{L}(\text{IDPDA}_0)$ . Finally,  $L_3$  belongs to  $\text{co-}\mathcal{L}(\text{REV-IDPDA}_0)$  as well as  $\text{co-}\mathcal{L}(\text{REV-IDPDA})$ , but not to  $\mathcal{L}(\text{IDPDA}_0)$ . This shows all incomparabilities claimed.  $\square$

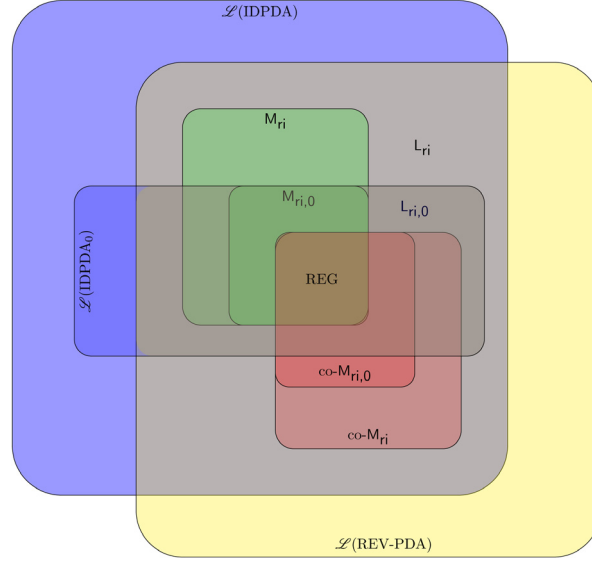


FIGURE 2. The hierarchical structure of the language classes discussed in Section 4. (Color online).

## 5. CLOSURE PROPERTIES

In this section, we investigate the closure properties of the language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$ . The closure properties of REV-PDA have been studied in [16]. 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, that is, the same signature. Here we distinguish this important special case from the general one. We 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  $\bigcup_{j \in \{D,R,N\}} (\Sigma_j \setminus \Sigma'_j) \cap \Sigma' = \emptyset$  and  $\bigcup_{j \in \{D,R,N\}} (\Sigma'_j \setminus \Sigma_j) \cap \Sigma = \emptyset$ .

In the following, we obtain several non-closure results for the language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$ . To this end, the languages

$$L_1 = \{ a^n \$_1 b^m \$_2 c^\ell \mid \ell, m, n \geq 0 \text{ and } \ell = m + n \},$$

$$L_2 = \{ a^n \$_1 b^m \$_2 c^\ell \mid \ell, m, n \geq 0 \text{ and } \ell = m \}, \text{ and } L_{12} = L_1 \cup L_2$$

are utilized.

**Lemma 5.1.** *Language  $L_{12}$  is not accepted by any REV-PDA.*

*Proof.* Assume in contrast to the assertion that language  $L_{12}$  is accepted by some REV-PDA  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$ . Since all REV-PDA can be made real time [16], we may safely assume that  $M$  is a real-time machine. During the computation of  $M$  on input prefixes  $a^+$  no combination of state and content of the pushdown store may appear twice. If

$$(\lambda, q_0, a^n \$_1 b^m \$_2 c^{n+m}, \perp) \vdash^* (a^{n_1}, q_1, a^{n-n_1} \$_1 b^m \$_2 c^{n+m}, \sigma_1) \vdash^+ (a^{n_1+n_2}, q_1, a^{n-n_1-n_2} \$_1 b^m \$_2 c^{n+m}, \sigma_1)$$

is the beginning of an accepting computation, then so is

$$(\lambda, q_0, a^{n-n_2} \$_1 b^m \$_2 c^{n+m}, \perp) \vdash^* (a^{n_1}, q_1, a^{n-n_1-n_2} \$_1 b^m \$_2 c^{n+m}, \sigma_1),$$

but  $a^{n-n_2} \$_1 b^m \$_2 c^{n+m}$  does not belong to  $L_{12}$ . This implies that each height of the pushdown store may appear only finitely often, thus the height increases arbitrarily. So,  $M$  runs into a loop while processing  $a$ 's, that is,

the combination of a state and, for any fixed number  $k$ , some  $k$  topmost pushdown symbols  $\alpha$  appear again and again. To render the loop more precisely, let  $(a^{n-x}, q, a^x \$1 b^m \$2 c^{n+m}, \alpha\gamma)$  be a configuration of the loop. Then  $(a^{n-x+y}, q, a^{x-y} \$1 b^m \$2 c^{n+m}, \alpha\beta)$  is a successor configuration with the same combination of state and topmost pushdown symbols. We may choose  $\alpha$  so that no symbol of  $\gamma$  is touched during the computation starting in configuration  $(a^{n-x}, q, a^x \$1 b^m \$2 c^{n+m}, \alpha\gamma)$ , that is,  $\alpha\beta = \alpha\gamma'\gamma$ . Thus, the computation continues as  $(a^{n-x+y}, q, a^{x-y} \$1 b^m \$2 c^{n+m}, \alpha\gamma'\gamma)$  and further to  $(a^{n-x+2y}, q, a^{x-2y} \$1 b^m \$2 c^{n+m}, \alpha\gamma'\gamma'\gamma)$ .

Similarly,  $M$  enters a loop while processing infixes  $b^+$ . Assume that during the computation of  $M$  on input infixes  $b^+$  some combination of state and content of the pushdown store appears twice. If

$$(\lambda, q_0, a^n \$1 b^m \$2 c^{n+m}, \perp) \vdash^* (a^n \$1 b^{m_1}, q'_1, b^{m-m_1} \$2 c^{n+m}, \sigma'_1) \vdash^+ (a^n \$1 b^{m_1+m_2}, q'_1, b^{m-m_1-m_2} \$2 c^{n+m}, \sigma'_1)$$

is the beginning of an accepting computation, then so is

$$(\lambda, q_0, a^n \$1 b^{m-m_2} \$2 c^{n+m}, \perp) \vdash^* (a^n \$1 b^{m_1}, q'_1, b^{m-m_1-m_2} \$2 c^{n+m}, \sigma'_1),$$

but  $a^n \$1 b^{m-m_2} \$2 c^{n+m}$  does not belong to  $L_{12}$ . This implies that each height of the pushdown store may appear only finitely often and, thus, that the height increases or decreases arbitrarily. Assume that the height decreases arbitrarily. Then, for infinitely many  $n$  and  $m$  large enough  $M$  enters on input prefix  $a^n \$1 b^{m-m_1}$  with  $0 \leq m_1 \leq m$  a configuration in which the height of the pushdown store is bounded by a fixed number only. Hence, some height of the pushdown store up to this fixed bound appears infinitely often which is a contradiction. Thus, the height increases arbitrarily.

Next, we turn to the input suffixes. While  $M$  processes the input suffixes  $c^+$ , again, no combination of state and content of the pushdown store may appear twice. If

$$(\lambda, q_0, a^n \$1 b^m \$2 c^{n+m}, \perp) \vdash^* (a^n \$1 b^m \$2 c^{m_1}, q''_1, c^{n+m-m_1}, \sigma''_1) \vdash^+ (a^n \$1 b^m \$2 c^{m_1+m_2}, q''_1, c^{n+m-m_1-m_2}, \sigma''_1)$$

results in an accepting computation, then so does

$$(\lambda, q_0, a^n \$1 b^m \$2 c^{n+m-m_2}, \perp) \vdash^* (a^n \$1 b^m \$2 c^{m_1}, q''_1, c^{n+m-m_1-m_2}, \sigma''_1),$$

but  $a^n \$1 b^m \$2 c^{n+m-m_2}$  does not belong to  $L_{12}$ , if  $m_2 \neq n$ . On the other hand, if  $m_2 = n$ , we run through the loop a second time: if

$$\begin{aligned} (\lambda, q_0, a^n \$1 b^m \$2 c^{n+m}, \perp) \vdash^* & (a^n \$1 b^m \$2 c^{m_1}, q''_1, c^{n+m-m_1}, \sigma''_1) \vdash^+ \\ & (a^n \$1 b^m \$2 c^{m_1+m_2}, q''_1, c^{n+m-m_1-m_2}, \sigma''_1) \vdash^+ \\ & (a^n \$1 b^m \$2 c^{m_1+2m_2}, q''_1, c^{n+m-m_1-2m_2}, \sigma''_1) \end{aligned}$$

results in an accepting computation, then so does

$$(\lambda, q_0, a^n \$1 b^m \$2 c^{n+m-2m_2}, \perp) \vdash^* (a^n \$1 b^m \$2 c^{m_1}, q''_1, c^{n+m-m_1-2m_2}, \sigma''_1),$$

but  $a^n \$1 b^m \$2 c^{n+m-2m_2} = a^n \$1 b^m \$2 c^{m-n}$  does not belong to  $L_{12}$ , since  $m_2 = n$ .

This implies that each height of the pushdown store may appear only finitely often. Moreover, in any accepting computation of inputs from  $L_1$  the pushdown store has to be decreased until some symbol of  $\gamma$  appears. Otherwise, we could increase the number of  $a$ 's by  $y$  to drive  $M$  through an additional loop while processing the input prefix. The resulting computation would also be accepting but the input does not belong to  $L_{12}$ . Together we conclude that  $M$  runs into a loop that decreases the height of the pushdown store while processing the  $c$ 's, and that on inputs from  $L_1$  there are only finitely many combinations of state and content of the pushdown store which are accepting.

Now, consider two different pairs  $(n_1, m_1) \neq (n_2, m_2)$  having the property  $\ell = n_1 + m_1 = n_2 + m_2$ ,  $n_1, n_2, m_1, m_2 > 0$ , and  $M$  accepts  $a^{n_1} \$1 b^{m_1} \$2 c^\ell$  and  $a^{n_2} \$1 b^{m_2} \$2 c^\ell$  in the same combinations of state and

content of the pushdown store, say in state  $q_a$  with  $\gamma_a$  in the pushdown store. We have the forward computations

$$\begin{aligned} (\lambda, q_0, a^{n_1}\$1b^{m_1}\$2c^\ell, \perp) \vdash^{n_1+m_1+2} (a^{n_1}\$1b^{m_1}\$2, q_1, c^\ell, \gamma_1) \vdash^\ell (a^{n_1}\$1b^{m_1}\$2c^\ell, q_a, \lambda, \gamma_a), \text{ and} \\ (\lambda, q_0, a^{n_2}\$1b^{m_2}\$2c^\ell, \perp) \vdash^{n_2+m_2+2} (a^{n_2}\$1b^{m_2}\$2, q_2, c^\ell, \gamma_2) \vdash^\ell (a^{n_2}\$1b^{m_2}\$2c^\ell, q_a, \lambda, \gamma_a). \end{aligned}$$

Since  $M$  is reversible, this implies that  $q_1 = q_2$  and  $\gamma_1 = \gamma_2$ . Then, we end up in the same state and pushdown content on both inputs  $a^{n_1}\$1b^{m_1}\$2$  and  $a^{n_2}\$1b^{m_2}\$2$ . Since  $a^{n_1}\$1b^{m_1}\$2c^{m_1} \in L_{12}$  and  $a^{n_2}\$1b^{m_2}\$2c^{m_2} \in L_{12}$ , we obtain that also the input  $a^{n_1}\$1b^{m_1}\$2c^{m_2}$  is in  $L_{12}$  which is a contradiction, since  $m_2 \neq m_1$  and  $n_1 + m_1 \neq m_2$ .  $\square$

Let us now establish closure and non-closure results for the language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  starting with the Boolean operations union, intersection, and complementation.

**Theorem 5.2.** *Language class  $L_{ri}$  is closed under complementation, but the language classes  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  are not. Language classes  $M_{ri}$ , and  $M_{ri,0}$  are not closed under union, but closed under intersection with compatible signatures. Language class  $L_{ri}$  is not closed under union and intersection with compatible signatures. Language class  $L_{ri,0}$  is not closed under union with compatible signatures.*

*Proof.* Language class  $L_{ri}$  is closed under complementation, since each of the language classes  $\mathcal{L}(\text{REV-PDA})$  and  $\mathcal{L}(\text{IDPDA})$  is closed under complementation. On the other hand, following the remark after Theorem 3.5 and the proof of Theorem 3.6, language  $L'_c = \{a^n\$b^n \mid n \geq 0\}$  is a witness for the non-closure of  $M_{ri}$  under complementation. Since  $L'_c$  is also accepted by some  $\text{REV-IDPDA}_0$ ,  $L'_c$  is also a witness for the non-closure of  $M_{ri,0}$  under complementation. For the non-closure of  $L_{ri,0}$  under complementation we consider the language  $\{a^n\$b^m \mid 0 \leq m \leq n\}$  that belongs to  $L_{ri,0}$ , but whose complement does not even belong to  $\mathcal{L}(\text{IDPDA}_0)$  by the proof given in Theorem 4.5.

Next, we consider the closure of the language classes  $M_{ri,0}$  and  $M_{ri}$  under intersection, which follows from the standard cross product construction. In detail, let  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  and  $M' = \langle Q', \Sigma', \Gamma', F', q'_0, \perp, \delta' \rangle$  be two  $\text{REV-IDPDA}_0$ , so that  $\Sigma$  and  $\Sigma'$  are compatible. A  $\text{REV-IDPDA}_0$

$$M'' = \langle Q \times Q', \Sigma \cup \Sigma', \Gamma \times \Gamma', F \times F', (q_0, q'_0), (\perp, \perp), \delta'' \rangle$$

accepting  $L(M) \cap L(M')$  is constructed as follows. Let  $q, \hat{q} \in Q$ ,  $q', \hat{q}' \in Q'$ ,  $Z \in \Gamma$ ,  $\hat{Z} \in \Gamma \setminus \{\perp\}$ ,  $Z' \in \Gamma'$ , and  $\hat{Z}' \in \Gamma' \setminus \{\perp\}$ .

- For  $a \in \Sigma_D \cap \Sigma'_D$ , we define  $\delta''_D((q, q'), a, (Z, Z')) = ((\hat{q}, \hat{q}'), (\hat{Z}, \hat{Z}')(Z, Z'))$  with  $\delta_D(q, a, Z) = (\hat{q}, \hat{Z}Z)$  and  $\delta'_D(q', a, Z') = (\hat{q}', \hat{Z}'Z')$ .
- For  $a \in \Sigma_R \cap \Sigma'_R$ , we define  $\delta''_R((q, q'), a, (Z, Z')) = ((\hat{q}, \hat{q}'), \lambda)$  with  $\delta_R(q, a, Z) = (\hat{q}, \lambda)$  and  $\delta'_R(q', a, Z') = (\hat{q}', \lambda)$ .
- For  $a \in \Sigma_N \cap \Sigma'_N$ , we define  $\delta''_N((q, q'), a, (Z, Z')) = ((\hat{q}, \hat{q}'), (Z, Z'))$  with  $\delta_N(q, a, Z) = (\hat{q}, Z)$  and  $\delta'_N(q', a, Z') = (\hat{q}', Z')$ .
- If  $\delta_X$  or  $\delta'_X$  with  $X \in \{D, R, N\}$  is undefined for some triple  $(q, a, Z)$  or  $(q', a, Z')$ , then  $\delta''_X$  is undefined as well. Moreover,  $\delta''_X$  with  $X \in \{D, R, N\}$  is undefined for all remaining input symbols  $a \in \Sigma \cup \Sigma'$ .

Clearly,  $M''$  accepts  $L(M) \cap L(M')$ . By construction,  $M''$  is an  $\text{IDPDA}_0$ . Furthermore, since  $M$  and  $M'$  are reversible and both transition functions are simulated at the same time in one component each, it is possible to compute the reverse transition in each component uniquely. Thus,  $M''$  is reversible as well. The construction for two given  $\text{REV-IDPDA}$  is identical.

To show the non-closure under union, we consider the witness language  $L_{12}$ . By Lemma 5.1 it is sufficient to show that  $L_1$  as well as  $L_2$  are accepted by  $\text{REV-IDPDA}_0$  with compatible signatures.

The basic idea to accept  $L_1$  is to push one symbol  $A$  for every  $a$  and  $b$  read and to pop one symbol  $A$  for every  $c$  read. The detailed construction is similar to the construction given in Example 2.1. Again, the first  $a$  read is marked by a special symbol  $\bar{A}$  on the stack. Additionally, we have to take care of the special cases that

the number of  $a$ 's or  $b$ 's might be zero. To be reversible in situations when the reverse transition function pops some symbol, we add forward transitions for all possible symbols on top of the stack. Formally, we construct a REV-IDPDA<sub>0</sub>  $M_1 = \langle Q_1, \Sigma, \Gamma_1, F_1, q_0, \perp, \delta_1 \rangle$  with state set  $Q_1 = \{q_0, q_1, \dots, q_6, q_+, q'_+\}$ , accepting states  $F_1 = \{q_+, q'_+\}$ ,  $\Gamma_1 = \{A, \bar{A}, B, \bar{B}, \perp\}$ ,  $\Sigma_D = \{a, b\}$ ,  $\Sigma_R = \{c\}$ , and  $\Sigma_N = \{\$1, \$2\}$ . The transition functions  $\delta_1$  and its reverse  $\delta_1^-$  are as follows. Let  $X$  denote all possible pushdown symbols from  $\Gamma_1$  and  $Y \in \{A, \bar{A}\}$ .

REV-IDPDA <sub>0</sub> forward		REV-IDPDA <sub>0</sub> backward	
(1)	$\delta_1(q_0, a, X) = (q_1, \bar{A}X)$	(1)	$\delta_1^-(q_1, a, \bar{A}) = (q_0, \lambda)$
(2)	$\delta_1(q_0, \$1, \perp) = (q_4, \perp)$	(2)	$\delta_1^-(q_4, \$1, \perp) = (q_0, \perp)$
(3)	$\delta_1(q_1, a, X) = (q_1, AX)$	(3)	$\delta_1^-(q_1, a, A) = (q_1, \lambda)$
(4)	$\delta_1(q_1, \$1, Y) = (q_2, Y)$	(4)	$\delta_1^-(q_2, \$1, Y) = (q_1, Y)$
(5)	$\delta_1(q_2, b, X) = (q_2, AX)$	(5)	$\delta_1^-(q_2, b, A) = (q_2, \lambda)$
(6)	$\delta_1(q_2, \$2, Y) = (q_3, Y)$	(6)	$\delta_1^-(q_3, \$2, Y) = (q_2, Y)$
(7)	$\delta_1(q_3, c, A) = (q_3, \lambda)$	(7)	$\delta_1^-(q_3, c, X) = (q_3, AX)$
(8)	$\delta_1(q_3, c, \bar{A}) = (q_+, \lambda)$	(8)	$\delta_1^-(q_+, c, X) = (q_3, \bar{A}X)$
(9)	$\delta_1(q_4, b, X) = (q_5, \bar{A}X)$	(9)	$\delta_1^-(q_5, b, \bar{A}) = (q_4, \lambda)$
(10)	$\delta_1(q_4, \$2, \perp) = (q'_+, \perp)$	(10)	$\delta_1^-(q'_+, \$2, \perp) = (q_4, \perp)$
(11)	$\delta_1(q_5, b, X) = (q_5, AX)$	(11)	$\delta_1^-(q_5, b, A) = (q_5, \lambda)$
(12)	$\delta_1(q_5, \$2, Y) = (q_6, Y)$	(12)	$\delta_1^-(q_6, \$2, Y) = (q_5, Y)$
(13)	$\delta_1(q_6, c, A) = (q_6, \lambda)$	(13)	$\delta_1^-(q_6, c, X) = (q_6, AX)$
(14)	$\delta_1(q_6, c, \bar{A}) = (q'_+, \lambda)$	(14)	$\delta_1^-(q'_+, c, X) = (q_6, \bar{A}X)$

The construction of a REV-IDPDA<sub>0</sub>  $M_2$  for  $L_2$  is similar. The basic idea to accept  $L_2$  is to push one symbol  $A$  for every  $a$  read and one symbol  $B$  for every  $b$  read, while one symbol  $B$  is popped for every  $c$ . Again, the first  $a$  or  $b$  read is marked by a special symbol  $\bar{A}$  or  $\bar{B}$  on the stack.  $M_2 = \langle Q_2, \Sigma, \Gamma_2, F_2, q_0, \perp, \delta_2 \rangle$  with  $Q_2 = \{q_0, q_1, q_2, q_3, q_4, q_+\}$ ,  $F_2 = \{q_+\}$ ,  $\Gamma_2 = \{A, \bar{A}, B, \bar{B}, \perp\}$ ,  $\Sigma_D = \{a, b\}$ ,  $\Sigma_R = \{c\}$ , and  $\Sigma_N = \{\$1, \$2\}$ . The transition functions  $\delta_2$  and its reverse  $\delta_2^-$  are as follows. Let  $X$  denote all possible pushdown symbols from  $\Gamma_2$ .

REV-IDPDA <sub>0</sub> forward		REV-IDPDA <sub>0</sub> backward	
(1)	$\delta_2(q_0, a, X) = (q_1, \bar{A}X)$	(1)	$\delta_2^-(q_1, a, \bar{A}) = (q_0, \lambda)$
(2)	$\delta_2(q_0, \$1, \perp) = (q_2, \perp)$	(2)	$\delta_2^-(q_2, \$1, \perp) = (q_0, \perp)$
(3)	$\delta_2(q_1, a, X) = (q_1, AX)$	(3)	$\delta_2^-(q_1, a, A) = (q_1, \lambda)$
(4)	$\delta_2(q_1, \$1, A) = (q_2, A)$	(4)	$\delta_2^-(q_2, \$1, A) = (q_1, A)$
(5)	$\delta_2(q_1, \$1, \bar{A}) = (q_2, \bar{A})$	(5)	$\delta_2^-(q_2, \$1, \bar{A}) = (q_1, \bar{A})$
(6)	$\delta_2(q_2, b, X) = (q_3, \bar{B}X)$	(6)	$\delta_2^-(q_3, b, \bar{B}) = (q_2, \lambda)$
(7)	$\delta_2(q_2, \$2, \perp) = (q_+, \perp)$	(7)	$\delta_2^-(q_+, \$2, \perp) = (q_2, \perp)$
(8)	$\delta_2(q_3, b, X) = (q_3, BX)$	(8)	$\delta_2^-(q_3, b, B) = (q_3, \lambda)$
(9)	$\delta_2(q_3, \$2, B) = (q_4, B)$	(9)	$\delta_2^-(q_4, \$2, B) = (q_3, B)$
(10)	$\delta_2(q_3, \$2, \bar{B}) = (q_4, \bar{B})$	(10)	$\delta_2^-(q_4, \$2, \bar{B}) = (q_3, \bar{B})$
(11)	$\delta_2(q_4, c, B) = (q_4, \lambda)$	(11)	$\delta_2^-(q_4, c, X) = (q_4, BX)$
(12)	$\delta_2(q_4, c, \bar{B}) = (q_+, \lambda)$	(12)	$\delta_2^-(q_+, c, X) = (q_4, \bar{B}X)$

Altogether, we obtain that  $L_1$  as well as  $L_2$  are accepted by REV-IDPDA<sub>0</sub> having the same signature. On the other hand, Lemma 5.1 shows that the union of  $L_1$  and  $L_2$  does not even belong to  $L_{ri}$ . Thus,  $M_{ri,0}$  and  $L_{ri,0}$  as well as  $M_{ri}$  and  $L_{ri}$  are not closed under union. Since  $L_{ri}$  is closed under complementation, we can conclude that  $L_{ri}$  is not closed under intersection as well.  $\square$

We continue with the closure properties under the operations concatenation, iteration, reversal, length-preserving homomorphism, and inverse homomorphism, and obtain the non-closure under every such operation for the language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$ .

**Theorem 5.3.** *Language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  are neither closed under concatenation with compatible signatures nor under iteration.*

*Proof.* Let us first note that it can be shown similar to the proof given in Lemma 5.1 that language  $L'_{12} = \{a^n \$1 \# b^m \$2 c^\ell \mid \ell, m, n \geq 0 \text{ and } (\ell = m + n \text{ or } \ell = m)\}$  is not accepted by any REV-PDA. Now, we consider the following languages  $L'_1 = \{a^n \$1 \# b^m \$2 c^\ell \mid \ell, m, n \geq 0 \text{ and } \ell = m + n\} \cup \{\# b^m \$2 c^m \mid m \geq 0\}$  and  $L'_2 = \{a^n \$1 \mid n \geq 0\}$ . We give an informal description of the constructions only, since the formal constructions are similar to those presented in the proof of Theorem 5.2. First, we describe how  $L'_1$  can be accepted by some REV-IDPDA<sub>0</sub>. The basic idea is to check whether or not the input starts with  $a^* \$1$  and to store this information in the state set. If the input starts with  $a^* \$1$ , then symbols  $A$  while reading  $a$ 's and  $b$ 's are pushed on the stack which are popped while reading  $c$ 's. The input is accepted if it is correctly formatted and the stack is empty up to  $\perp$ . If the input starts with  $\# b^* \$2$ , then symbols  $A$  while reading  $b$ 's are pushed on the stack which are popped while reading  $c$ 's. Again, the input is accepted if it is correctly formatted and the stack is empty up to  $\perp$ . A REV-IDPDA<sub>0</sub> accepting  $L'_2$  pushes symbols  $A$  while reading  $a$ 's and accepts if the input is correctly formatted. Thus,  $L'_1$  and  $L'_2$  are accepted by REV-IDPDA<sub>0</sub> with compatible signatures.

Next, we show that  $L'_2 L'_1$  is not accepted by any REV-PDA, and therefore does not belong to  $L_{ri}$ . Assume that  $L'_2 L'_1$  is accepted by some REV-PDA. Since  $\mathcal{L}(\text{REV-PDA})$  is closed under intersection with reversible regular languages [16], we obtain that  $L'_2 L'_1 \cap a^* \$1 \# b^* \$2 c^* = L'_{12}$  is accepted by some REV-PDA which is a contradiction.

To obtain the non-closure under iteration we first show that  $L'_1 \cup L'_2$  belongs to  $M_{ri,0}$ . The construction is nearly identical to the above-described construction of a REV-IDPDA<sub>0</sub> for  $L'_1$ . The only difference is that an accepting state is also entered after reading some prefix  $a^* \$1$ . Now, we assume that  $M_{ri,0}$ ,  $M_{ri}$ ,  $L_{ri,0}$ , or  $L_{ri}$  is closed under iteration. Then  $(L'_1 \cup L'_2)^* \cap a^* \$1 \# b^* \$2 c^* = L'_{12}$  belongs to  $\mathcal{L}(\text{REV-PDA})$  which is again a contradiction.  $\square$

**Theorem 5.4.** *Language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  are not closed under reversal.*

*Proof.* We consider  $L' = \{c^\ell \$2 b^m \$1 a^n \mid \ell, m, n \geq 0 \text{ and } (\ell = m \text{ or } \ell = m + n)\}$  which can be accepted by a REV-IDPDA which is similarly constructed as in the proofs of Theorems 5.2 and 5.3. Basically, symbols  $A$  are pushed on the stack while reading  $c$ 's which are popped while reading  $b$ 's and  $a$ 's. The input is accepted if it is correctly formatted and the stack is empty up to  $\perp$  after reading  $\$1$  or at the end of the input. On the other hand, the reversal  $(L')^R$  equals  $L_{12} = L_1 \cup L_2$  which does not belong to  $L_{ri}$  due to Lemma 5.1. This shows the non-closure under reversal for the classes  $L_{ri}$  and  $M_{ri}$ .

Next, consider language  $L'' = \{b^m \$a^n \mid m > n \geq 0\}$ . Clearly,  $L''$  can be accepted by some REV-IDPDA<sub>0</sub> and, therefore,  $L''$  belongs to  $M_{ri,0}$  as well as  $L_{ri,0}$ . On the other hand,  $(L'')^R = \{a^n \$b^m \mid m > n \geq 0\}$ . It is shown in the proof of Theorem 4.2 that  $(L'')^R$  is not even accepted by any IDPDA<sub>0</sub>. This shows the non-closure under reversal for the classes  $L_{ri,0}$  and  $M_{ri,0}$ .  $\square$

**Theorem 5.5.** *Language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  are not closed under intersection with regular languages, but closed under intersection with reversible regular languages.*

*Proof.* We consider the Dyck language

$$L' = \{w \in \{a, b\}^* \mid |w|_a = |w|_b \text{ and } |w'|_a \geq |w'|_b \text{ for every prefix } w' \text{ of } w\}$$

and present a REV-IDPDA<sub>0</sub> accepting  $L'$ . Let  $M = \langle Q, \Sigma, \Gamma, F, q_0, \perp, \delta \rangle$  with  $Q = \{q_0, q_1\}$ ,  $F = \{q_0\}$ ,  $\Gamma = \{A, \bar{A}, \perp\}$ ,  $\Sigma_D = \{a\}$ ,  $\Sigma_R = \{b\}$ , and  $\Sigma_N = \emptyset$ . The transition functions  $\delta$  and its reverse  $\delta^-$  are as follows. Let  $X$  denote all possible pushdown symbols from  $\Gamma$ .

REV-IDPDA <sub>0</sub> forward		REV-IDPDA <sub>0</sub> backward	
(1)	$\delta(q_0, a, X) = (q_1, AX)$	(1)	$\delta^-(q_1, a, A) = (q_0, \lambda)$
(2)	$\delta(q_1, a, X) = (q_1, AX)$	(2)	$\delta^-(q_1, a, A) = (q_1, \lambda)$
(3)	$\delta(q_1, b, A) = (q_1, \lambda)$	(3)	$\delta^-(q_1, b, X) = (q_1, AX)$
(4)	$\delta(q_1, b, A) = (q_0, \lambda)$	(4)	$\delta^-(q_0, b, X) = (q_1, AX)$

Assume that  $L_{ri}$  is closed under intersection with regular languages. Then  $L' \cap a^*b^* = L_c$  belongs to  $L_{ri}$  which is a contradiction.

The closure under intersection with reversible regular languages can be shown similar to the constructions for IDPDA<sub>0</sub> in Lemma 4.1 and for REV-PDA given in [16]. The basic idea is to use the standard cross product construction and to simulate a reversible deterministic finite automaton in a second component.  $\square$

**Theorem 5.6.** *Language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  are neither closed under length-preserving homomorphism nor under inverse homomorphism.*

*Proof.* Language  $L'_c = \{a^n\$b^n \mid n \geq 0\}$  belongs to  $M_{ri,0}$ . On the other hand, let us consider the length-preserving homomorphism  $h$  mapping  $a$  to  $c$ ,  $\$$  to  $\$,$  and  $b$  to  $c$ . Then,  $h(L'_c) = \{c^n\$c^n \mid n \geq 0\} = L_s$  which is not accepted by any IDPDA.

To show the non-closure under inverse homomorphism, we consider the language  $L'' = \{a^{2n}\$b^{2n} \mid n \geq 0\}$  which clearly belongs to  $M_{ri,0}$  and a homomorphism  $h$  mapping  $a$  to  $aa$ ,  $\$$  to  $\$,$  and  $b$  to  $b$ . Then  $h^{-1}(L'') = \{a^n\$b^{2n} \mid n \geq 0\} = L'_d$  which is not accepted by any IDPDA.  $\square$

**Corollary 5.7.** *Language classes  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  are anti-AFLs.*

**Remark 5.8.** The closure properties discussed in this section are summarized in Tables 1 and 2, where also the closure properties of the language classes REG, DCFL,  $\mathcal{L}(\text{IDPDA})$ , and  $\mathcal{L}(\text{REV-PDA})$  are listed. The proofs for the latter classes may be found in [1, 2, 16]. The non-closure of the class  $\mathcal{L}(\text{IDPDA})$  under union, intersection, and concatenation in case of incompatible signatures is discussed in [21]. To obtain the non-closure of  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  under intersection we consider the languages  $\{a^n\$1b^n\$2c^m \mid n, m \geq 0\}$  and  $\{a^n\$1b^m\$2c^m \mid n, m \geq 0\}$ . It is not difficult to show following the previous constructions that each language can be accepted by some REV-IDPDA<sub>0</sub>. However, the intersection of both languages yields a non-context-free language. The non-closure under union and concatenation of  $L_{ri}$ ,  $L_{ri,0}$ ,  $M_{ri}$ , and  $M_{ri,0}$  in case of incompatible signatures follows from the non-closure results for the corresponding operations with compatible signatures.

TABLE 1. Closure properties of the language classes discussed. Symbols  $\cup_c$  and  $\cap_c$  denote union and intersection with compatible signatures. Such operations are not defined for finite automata, deterministic pushdown automata, and REV-PDA and marked with ‘—’.

	—	$\cup$	$\cup_c$	$\cap$	$\cap_c$	$\cap \text{REG}$	$\cap \text{REG}_{rev}$
REG	yes	yes	—	yes	—	yes	yes
DCFL	yes	no	—	no	—	yes	yes
$\mathcal{L}(\text{IDPDA})$	yes	no	yes	no	yes	yes	yes
$\mathcal{L}(\text{IDPDA}_0)$	no	no	yes	no	yes	yes	yes
$\mathcal{L}(\text{REV-PDA})$	yes	no	—	no	—	no	yes
$L_{ri}$	yes	no	no	no	no	no	yes
$L_{ri,0}$	no	no	no	no	?	no	yes
$M_{ri}$	no	no	no	no	yes	no	yes
$M_{ri,0}$	no	no	no	no	yes	no	yes



TABLE 2. Further closure properties of the language classes discussed. Symbol  $\cdot_c$  denotes concatenation with compatible signatures. By  $h_{l.p.}$  we denote length-preserving homomorphisms.

	$\cdot$	$\cdot_c$	$*$	$h_{l.p.}$	$h^{-1}$	$R$
REG	yes	—	yes	yes	yes	yes
DCFL	no	—	no	no	yes	no
$\mathcal{L}(\text{IDPDA})$	no	yes	yes	no	no	yes
$\mathcal{L}(\text{IDPDA}_0)$	no	yes	yes	no	no	no
$\mathcal{L}(\text{REV-PDA})$	no	—	no	no	yes	no
$L_{ri}$	no	no	no	no	no	no
$L_{ri,0}$	no	no	no	no	no	no
$M_{ri}$	no	no	no	no	no	no
$M_{ri,0}$	no	no	no	no	no	no

**Remark 5.9.** Finally, we have to discuss the closure properties of language class  $\mathcal{L}(\text{IDPDA}_0)$ . The closure under union, intersection, and concatenation with compatible signatures and under iteration can be shown the same way as for language class  $\mathcal{L}(\text{IDPDA})$ . The constructions for union and intersection are standard cross product constructions in which both  $\text{IDPDA}_0$  are simulated at the same time. For union we should note that both  $\text{IDPDA}$  are prepared in such a way that they enter a non-accepting state whenever they would remove the symbol  $\perp$ . This avoids that the overall simulation gets stuck in case of one  $\text{IDPDA}_0$  getting stuck. The non-closure under union, intersection and concatenation follows from Remark 5.8. The non-closure under complementation can be shown using language  $\{a^n \$ b^m \mid 0 \leq m \leq n\}$  and its complement that does not belong to  $\mathcal{L}(\text{IDPDA}_0)$ . The non-closure under reversal, length-preserving homomorphism, and inverse homomorphism follows by the same arguments as given in Theorems 5.4 and 5.6. The closure under intersection with regular languages is shown in Lemma 4.1.

## 6. CONCLUSIONS

In this paper, we have investigated variants of real-time deterministic pushdown automata restricted to reversible, input-driven, or reversible *and* input-driven computations. It turned out that each restriction leads to a different language class properly lying inside the real-time deterministic context-free languages. In the second part of the paper, we have also studied the closure properties of these language classes and we obtained results for all of the commonly studied closure properties such as the Boolean operations, concatenation, Kleene star, reversal, homomorphism, and inverse homomorphism. One topic of further research may be the investigation of decidability questions for these restricted real-time deterministic pushdown automata. It may happen that questions which are undecidable for real-time deterministic pushdown automata become decidable for the restricted variants. On the other hand, questions which are decidable for real-time deterministic pushdown automata may exhibit another computational complexity in the restricted cases.

## REFERENCES

- [1] R. Alur and P. Madhusudan, Visibly pushdown languages. In *Symposium on Theory of Computing, STOC ACM* (2004) 202–211.
- [2] R. Alur and P. Madhusudan, Adding nesting structure to words. *J. ACM* **56** (2009) 16.
- [3] H.B. Axelsen, Reversible multi-head finite automata characterize reversible logarithmic space. In *Language and Automata Theory and Applications (LATA 2012)*. Vol. 7183 of *Lect. Notes Comput. Sci.* Springer (2012) 95–105.
- [4] H.B. Axelsen and R. Glück, A simple and efficient universal reversible Turing machine. In *Language and Automata Theory and Applications (LATA 2011)*. Vol. 6638 of *Lect. Notes Comput. Sci.* Springer (2011) 117–128.

- [5] C.H. Bennett, Logical reversibility of computation. *IBM J. Res. Dev.* **17** (1973) 525–532.
- [6] S. Bensch, M. Holzer, M. Kutrib and A. Malcher, Input-driven stack automata. In *Theoretical Computer Science (TCS 2012)*. Vol. 7604 of *Lect. Notes Comput. Sci.* Springer (2012) 28–42.
- [7] P. Chervet and I. Walukiewicz, Minimizing variants of visibly pushdown automata. In *Mathematical Foundations of Computer Science (MFCS 2007)*. Vol. 4708 of *Lect. Notes Comput. Sci.* Springer (2007) 135–146.
- [8] S. Crespi-Reghizzi and D. Mandrioli, Operator precedence and the visibly pushdown property. *J. Comput. System Sci.* **78** (2012) 1837–1867.
- [9] Y.-S. Han and K. Salomaa, Nondeterministic state complexity of nested word automata. *Theoret. Comput. Sci.* **410** (2009) 2961–2971.
- [10] M.A. Harrison, *Introduction to Formal Language Theory*. Addison-Wesley (1978).
- [11] J. Kari, Reversible cellular automata. In *Developments in Language Theory (DLT 2005)*. Vol. 3572 of *Lect. Notes Comput. Sci.* Springer (2005) 57–68.
- [12] M. Kutrib, Aspects of reversibility for classical automata. In *Computing with New Resources*. Vol. 8808 of *Lect. Notes Comput. Sci.* Springer (2014) 83–98.
- [13] M. Kutrib and A. Malcher, When Church-Rosser becomes context free. *Int. J. Found. Comput. Sci.* **18** (2007) 1293–1302.
- [14] M. Kutrib and A. Malcher, Fast reversible language recognition using cellular automata. *Inform. Comput.* **206** (2008) 1142–1151.
- [15] M. Kutrib and A. Malcher, Real-time reversible iterative arrays. *Theoret. Comput. Sci.* **411** (2010) 812–822.
- [16] M. Kutrib and A. Malcher, Reversible pushdown automata. *J. Comput. System Sci.* **78** (2012) 1814–1827.
- [17] M. Kutrib and A. Malcher, One-way reversible multi-head finite automata. In *Reversible Computation (RC 2012)*. Vol. 7581 of *Lect. Notes Comput. Sci.* Springer (2013) 14–28.
- [18] M. Kutrib and A. Malcher, Real-time reversible one-way cellular automata. In *Cellular Automata and Discrete Complex Systems (AUTOMATA 2014)*. Vol. 8996 of *Lect. Notes Comput. Sci.* Springer (2015) 56–69.
- [19] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano and M. Wendlandt, Input-driven queue automata: Finite turns, decidability, and closure properties. *Theoret. Comput. Sci.* **578** (2015) 58–71.
- [20] M. Kutrib, A. Malcher and M. Wendlandt, Reversible queue automata. In *Non-Classical Models of Automata and Applications (NCMA 2014)*. Vol. 304 of *books@ocg.at*. Austrian Computer Society (2014) 163–178.
- [21] M. Kutrib, A. Malcher and M. Wendlandt, Tinput-driven pushdown automata. In *Machines, Computations, and Universality (MCU 2015)*. Vol. 9288 of *Lect. Notes Comput. Sci.* Springer (2015) 94–112.
- [22] S. La Torre, P. Madhusudan and G. Parlato, A robust class of context-sensitive languages. In *Logic in Computer Science (LICS 2007)*. IEEE Computer Society (2007) 161–170.
- [23] R. Landauer, Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.* **5** (1961) 183–191.
- [24] K.-J. Lange, P. McKenzie and A. Tapp, Reversible space equals deterministic space. *J. Comput. System Sci.* **60** (2000) 354–367.
- [25] P. Madhusudan and G. Parlato, The tree width of auxiliary storage. In *Principles of Programming Languages (POPL 2011)*. ACM (2011) 283–294.
- [26] K. Mehlhorn, Pebbling mountain ranges and its application of DCFL-recognition. In *International Colloquium on Automata, Languages and Programming (ICALP 1980)*. Vol. 85 of *Lect. Notes Comput. Sci.* Springer (1980) 422–435.
- [27] K. Morita, Reversible simulation of one-dimensional irreversible cellular automata. *Theoret. Comput. Sci.* **148** (1995) 157–163.
- [28] K. Morita, Reversible computing and cellular automata – a survey. *Theoret. Comput. Sci.* **395** (2008) 101–131.
- [29] K. Morita, Two-way reversible multi-head finite automata. *Fund. Inform.* **110** (2011) 241–254.
- [30] A. Okhotin and K. Salomaa, State complexity of operations on input-driven pushdown automata. In *Mathematical Foundations of Computer Science (MFCS 2011)*. Vol. 6907 of *Lect. Notes Comput. Sci.* Springer (2011) 485–496.
- [31] X. Piao and K. Salomaa, Operational state complexity of nested word automata. *Theoret. Comput. Sci.* **410** (2009) 3290–3302.
- [32] B. von Braunmühl and R. Verbeek, Input-driven languages are recognized in  $\log n$  space. In *Topics in the Theory of Computation*. Vol. 102 of *Mathematics Studies*. North-Holland (1985) 1–19.

Communicated by N. Moreira.

Received December 19, 2015. Accepted August 3, 2016.