

# Input-Driven Pushdown Automata with Translucent Input Letters

Martin Kutrib      Andreas Malcher  
Matthias Wendlandt

Institut für Informatik, Universität Giessen  
Arndtstr. 2, 35392 Giessen, Germany

{kutrib, andreas.malcher, matthias.wendlandt}@informatik.uni-giessen.de

Input-driven pushdown automata with translucent input letters are investigated. Here, the use of translucent input letters means that the input is processed in several sweeps and that, depending on the current state of the automaton, some input symbols are visible and can be processed, whereas some other symbols are invisible, and may be processed in another sweep. Additionally, the returning mode as well as the non-returning mode are considered, where in the former mode a new sweep must start after processing a visible input symbol. Input-driven pushdown automata differ from traditional pushdown automata by the fact that the actions on the pushdown store (push, pop, nothing) are dictated by the input symbols. We obtain the result that the input-driven nondeterministic model is computationally stronger than the deterministic model both in the returning mode and in the non-returning mode, whereas it is known that the deterministic and the nondeterministic model are equivalent for input-driven pushdown automata without translucency. It also turns out that the non-returning model is computationally stronger than the returning model both in the deterministic and nondeterministic case. Furthermore, we investigate the closure properties of the language families introduced under the Boolean operations. We obtain a complete picture in the deterministic case, whereas in the nondeterministic case the language families are shown to be not closed under complementation. Finally, we look at decidability questions and obtain the non-semidecidability of the questions of universality, inclusion, equivalence, and regularity in the nondeterministic case.

## 1 Introduction

The usual way of processing input on language recognition devices is by reading input strings from left to right, one symbol at a time, and finally providing an accept or reject decision when arriving at the end of the input. This “standard” input mode has been extended in the literature in several directions. For example, two-way motion of the input head, stationary moves of the input head, rotating the input head, or restarting modes have been considered for a wide variety of machines (the reader is referred to, e.g., [2, 6, 7, 23] for an overview of these and other models). In all these cases, the extensions have consequences on the computational and descriptive power of the machines. Thus, the way of processing the input can be considered as a computational resource that then can be used to tune computational and descriptive power.

Of particular interest in recent literature have been *discontinuous* ways of input processing, where one of them is the “jumping” paradigm which means that jumping to any position inside the input string is allowed at any move. This paradigm has been investigated for finite automata in [11], where it is shown that this discontinuous input processing may increase the computational power since some non-context-free languages can be accepted. On the other hand, the discontinuous input mode may also limit the computational power since some regular languages cannot be accepted. A restricted variant, called “right one-way jumping” automata, has been considered in [1, 4]. Another way to discontinuously

process the input is to use *translucent letters*. This concept has been introduced by Nagy and Otto in [15] for deterministic and nondeterministic finite automata and the basic idea for translucent devices is to provide a translucency function that defines in which states which letters of the input are translucent. At each move, the device skips (by looking through) the translucent portion of the input, from the current input head position up to the first non-translucent letter (thus realizing a jump). After processing the non-translucent symbol, in the *returning mode* the input head returns to the left end of the input while, in the *non-returning mode*, the input head continues to process the input according to its updated current state and the corresponding translucent symbols. In both modes, the input head returns to the left end when the right end of the input is reached.

Deterministic and nondeterministic finite automata with translucent letters have deeply been investigated in the literature (see, e.g., [13, 16, 22]). However, many questions are still open. Some recently studied variants are finite automata with translucent words [19], finite automata with translucent letters and two-way motion of the input head [8], and returning and non-returning deterministic pushdown automata with translucent letters [9, 10]. It should be noted that returning pushdown automata with translucent letters have been studied by Nagy and Otto in [14, 17] in terms of certain cooperating distributed systems of restarting automata with additional pushdown store. However, in their model  $\lambda$ -transitions are not allowed and acceptance is defined by empty pushdown. In [21] the paradigm of *input-driven* languages was imposed to certain cooperating distributed systems of restarting automata with additional pushdown store with regard to characterizing certain trace languages. In input-driven pushdown automata [3, 12] the next action on the pushdown store (push, pop, nothing) is solely governed by the input symbol and to this end the input alphabet is split into three subsets. Input-driven pushdown automata possess nice features (see, e.g., the survey given in [20]) such as the equivalence of nondeterministic and deterministic models, the positive decidability of the inclusion problem and the positive closure under union, intersection, complementation, concatenation, and iteration. It should be noted that the positive decidability result as well as the positive closures only hold if both given automata have a compatible splitting of their input alphabets. In this paper, we study input-driven deterministic and nondeterministic pushdown automata with translucent letters both in the returning and non-returning mode and, therefore, extend the results from [21]. We also study the closure properties of the four corresponding language families under the Boolean operations as well as the status of their decidability questions.

The paper is organized as follows. After giving the necessary definitions and two illustrating examples in the next section, we study in Section 3 the impact of nondeterminism in our models and it turns out that the nondeterministic model is computationally stronger in the returning mode as well as in the non-returning mode. In Section 4 we compare returning and non-returning models. We yield proper inclusions of the returning language families in the non-returning language families in the deterministic as well as in the nondeterministic case. Moreover, the combination deterministic and non-returning versus the combination nondeterministic and returning leads to incomparability results. The closure under the Boolean operations is studied in Section 5 and we obtain the closure under complementation, but non-closure under union and intersection in the returning and non-returning deterministic case. In the nondeterministic case, we obtain non-closure under complementation for both variants and non-closure under intersection (even with regular languages) in the returning case. Finally, we consider the usually studied decidability questions in Section 6. In the returning case we have the decidability of the emptiness problem as well as of the finiteness problem both in the nondeterministic and deterministic case. In addition, universality is decidable in the deterministic case. On the other hand, we get the non-semidecidability of the questions of universality, equivalence, inclusion, and regularity in case of input-driven nondeterministic pushdown automata with translucent letters working in the returning as well as in the non-returning mode. This extends some results partially known for nondeterministic finite

automata with translucent input letters working in the returning mode (see [16]).

## 2 Definitions and Preliminaries

We denote by  $\Sigma^*$  the set of all words on the finite alphabet  $\Sigma$ , including the *empty word*  $\lambda$ , and let  $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ . For any word  $w \in \Sigma^*$ , we let  $|w|$  denote its *length*,  $w^R$  its *reversal*, and  $|w|_a$  the *number of occurrences* of the symbol  $a \in \Sigma$  in  $w$ . We use  $\subseteq$  for *inclusions*, and  $\subset$  for *proper inclusion*. Given a set  $S$ , we denote by  $2^S$  its *power set*, and by  $|S|$  its *cardinality*. We write  $S_x$  to denote the set  $S \cup \{x\}$ , for a given element  $x \notin S$ . A *language* on  $\Sigma$  is any subset  $L \subseteq \Sigma^*$ . The *complement* of  $L$  is the language  $\bar{L} = \Sigma^* \setminus L$ , its reversal is  $L^R = \{w^R \mid w \in L\}$ . The *shuffle*  $L_1 \sqcup L_2$  of two languages  $L_1, L_2 \subseteq \Sigma^*$  is defined as  $L_1 \sqcup L_2 = \{x_1 y_1 x_2 y_2 \cdots x_n y_n \mid x_1 x_2 \cdots x_n \in L_1, y_1 y_2 \cdots y_n \in L_2 \text{ with } x_i, y_i \in \Sigma^* \text{ and } 1 \leq i \leq n\}$ . Let  $\Sigma = \{a_1, a_2, \dots, a_n\}$  and  $\Psi : \Sigma^* \rightarrow \mathbb{N}_0^n$  be a mapping such that  $\Psi(w) = (|w|_{a_1}, |w|_{a_2}, \dots, |w|_{a_n})$ . Let  $L \subseteq \Sigma^*$  be a language. Then,  $\Psi(L) = \{\Psi(w) \mid w \in L\}$  is the *Parikh image* of  $L$ . We say that two languages  $L_1, L_2 \subseteq \Sigma^*$  are *letter-equivalent* if  $\Psi(L_1) = \Psi(L_2)$ . Two language families  $\mathcal{L}_1$  and  $\mathcal{L}_2$  are said to be *incomparable* whenever  $\mathcal{L}_1$  is not a subset of  $\mathcal{L}_2$  and vice versa.

A classical pushdown automaton is called input-driven if the current 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, the input alphabet  $\Sigma$  is partitioned into the sets  $\Sigma_D$ ,  $\Sigma_R$ , and  $\Sigma_N$ , that control the actions push ( $D$ ), pop ( $R$ ), and state change only ( $N$ ).

Input-driven pushdown automata with translucent input letters are extensions of input-driven pushdown automata that do not necessarily have to read the current input symbol. Instead, depending on the current state of such devices, some of the input letters may be translucent (invisible). Accordingly, an input-driven pushdown automaton with translucent input letters either reads and processes (by deleting, if not the endmarker) the first visible input letter.

Formally, an *input-driven pushdown automaton with translucent input letters* (*NIDPDWtI*) is a system  $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \perp, \tau, \delta_D, \delta_R, \delta_N \rangle$ , where  $Q$  is the finite set of *internal states*,  $\Sigma$  is the finite set of *input symbols* partitioned into the sets  $\Sigma_D$ ,  $\Sigma_R$ , and  $\Sigma_N$ , with  $\Sigma \cap Q = \emptyset$ ,  $\Gamma$  is the finite set of *pushdown symbols*,  $q_0 \in Q$  is the *initial state*,  $\triangleleft \notin \Sigma$  is the endmarker,  $\perp \notin \Gamma$  is the *bottom-of-pushdown symbol*,  $\tau : Q \rightarrow 2^\Sigma$  is the translucency mapping, and  $\delta_D$  is the partial transition function mapping  $Q \times \Sigma_D \times (\Gamma \cup \{\perp\})$  to  $2^{Q \times \Gamma} \cup \{\text{accept}\}$ ,  $\delta_R$  is the partial transition function mapping  $Q \times \Sigma_R \times (\Gamma \cup \{\perp\})$  to  $2^Q \cup \{\text{accept}\}$ , and  $\delta_N$  is the partial transition function mapping  $Q \times (\Sigma_N \cup \{\triangleleft\}) \times (\Gamma \cup \{\perp\})$  to  $2^Q \cup \{\text{accept}\}$ .

The translucency mapping  $\tau$  bears the following meaning: for any state  $q \in Q$ , the letters from the set  $\tau(q)$  are *translucent (invisible)* for  $q$ , that is, whenever in  $q$ , the automaton  $M$  does not see these letters (or equivalently,  $M$  sees through such letters).

A *configuration* of  $M$  is a pair  $(qw\triangleleft, \gamma)$  or  $\text{accept}$ , where  $q \in Q$  is the current state,  $w \in \Sigma^*$  is the part of the input left to be processed, and  $\gamma \in \Gamma^* \perp$  denotes the current pushdown content, the leftmost symbol being the top of the pushdown store. The *initial configuration* for an input  $w$  is set to  $(q_0 w \triangleleft, \perp)$ .

Being in some configuration  $(qw\triangleleft, z\gamma)$  with  $z \in \Gamma \cup \{\perp\}$  and  $z\gamma \in \Gamma^* \perp$ , first  $M$  determines the next symbol to scan. Precisely, if  $w\triangleleft = xy\triangleleft$  with  $x \in \tau(q)^*$ ,  $y \in \Sigma^*$ , and  $a \notin \tau(q)$  is the first symbol of  $y\triangleleft$ , then  $M$  processes  $a$ . One step from a configuration to its successor configuration is denoted by  $\vdash$ .

Let  $q, q' \in Q$ ,  $x \in \tau(q)^*$ ,  $a \notin \tau(q)$ ,  $y \in \Sigma^*$ , and  $z \in \Gamma$ ,  $\gamma \in \Gamma^* \perp$ . We set

1.  $(qxay\triangleleft, z\gamma) \vdash (q'xy\triangleleft, z'\gamma)$ , if  $a \in \Sigma_D$  and  $(q', z') \in \delta_D(q, a, z)$ ,
2.  $(qxay\triangleleft, \perp) \vdash (q'xy\triangleleft, z'\perp)$ , if  $a \in \Sigma_D$  and  $(q', z') \in \delta_D(q, a, \perp)$ ,

3.  $(qxy\triangleleft, z\gamma) \vdash (q'xy\triangleleft, \gamma)$ , if  $a \in \Sigma_R$  and  $q' \in \delta_R(q, a, z)$ ,
4.  $(qxy\triangleleft, \perp) \vdash (q'xy\triangleleft, \perp)$ , if  $a \in \Sigma_R$  and  $q' \in \delta_R(q, a, \perp)$ ,
5.  $(qxy\triangleleft, z\gamma) \vdash (q'xy\triangleleft, z\gamma)$ , if  $a \in \Sigma_N$  and  $q' \in \delta_N(q, a, z)$ ,
6.  $(qxy\triangleleft, \perp) \vdash (q'xy\triangleleft, \perp)$ , if  $a \in \Sigma_N$  and  $q' \in \delta_N(q, a, \perp)$ ,
7.  $(qx\triangleleft, z\gamma) \vdash (q'x\triangleleft, z\gamma)$ , if  $q' \in \delta_N(q, \triangleleft, z)$ ,
8.  $(qx\triangleleft, \perp) \vdash (q'x\triangleleft, \perp)$ , if  $q' \in \delta_N(q, \triangleleft, \perp)$ .

In addition, whenever, the transition function yields *accept* for the current configuration, the successor configuration is *accept*.

So, on the endmarker only  $\delta_N$  is defined. Whenever the pushdown store is empty, the successor configuration is computed by the transition functions with the special bottom-of-pushdown symbol  $\perp$  which is never removed from the pushdown. As usual, we define the reflexive and transitive closure of  $\vdash$  by  $\vdash^*$ .

An input-driven pushdown automaton with translucent letters is said to be deterministic (DIDPDAtl) if  $|\delta_x(q, a, z)| \leq 1$  for  $x \in \{D, N, R\}$  and all  $q \in Q$ ,  $a \in \Sigma_x$ , and  $z \in \Gamma \cup \{\perp\}$ .

A word  $w$  is accepted by  $M$  if there is a computation on input  $w$  that ends with *accept*. The *language accepted* by  $M$  is  $L(M) = \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$ . In general, the family of all languages accepted by automata of some type  $X$  will be denoted by  $\mathcal{L}(X)$ .

Some properties of language families implied by classes of input-driven pushdown automata may depend on whether all automata involved share the same partition of the input alphabet. For easier writing, we call the partition of an input alphabet a *signature*.

In order to clarify these notions, we continue with an example.

**Example 1.** Let  $\Sigma = \{a, b, \#\}$  be an alphabet. The language  $L_{rep} \subseteq \Sigma^*$  is defined as  $L_{rep} = \{b^n(\#b^n)^k \sqcup a^n \mid n \geq 1, k \geq 0\}$ . Its complement  $\bar{L}_{rep}$  belongs to the family  $\mathcal{L}(\text{NIDPDAtl})$ . It can be represented as union  $L \cup L'$  where  $L'$  is the complement of the regular language  $\{b^+(\#b^+)^k \sqcup a^+ \mid k \geq 0\}$  with respect to  $\Sigma$ , and

$$L = \{b^{n_1}\#b^{n_2}\# \dots \#b^{n_k} \sqcup a^n \mid k \geq 0, n, n_i \geq 1 \text{ for } 1 \leq i \leq k, \text{ and there exists } 1 \leq i \leq k \text{ such that } n_i \neq n\}.$$

An NIDPDAtl accepting  $\bar{L}_{rep}$  can initially guess whether it wants to accept  $L$  or  $L'$ . Since  $L'$  is regular it is accepted by some NIDPDAtl that does not utilize its pushdown store. So, it does not care about what actually happens on the pushdown store. This means that it may have an arbitrary signature.

Now, we construct an NIDPDAtl  $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \perp, \tau, \delta_D, \delta_R, \delta_N \rangle$  accepting  $L$  as follows.

- $Q = \{q_0, q_1, p, r\}$ ,
- $\Sigma_D = \{b\}$ ,  $\Sigma_R = \{a\}$ ,  $\Sigma_N = \{\#\}$ ,
- $\Gamma = \{\bullet, B\}$ ,
- $\tau(q_0) = \tau(q_1) = \tau(p) = \{a\}$ ,  $\tau(r) = \{\#, b\}$ .

In a first phase,  $M$  reads the input symbols  $b$  and  $\#$  from left to right with symbols  $a$  translucent. At the beginning and whenever a  $\#$  is read,  $M$  guesses whether the length of the next  $b$ -block has to be matched with the number of  $a$ 's in the input. If not,  $M$  scans the  $b$ -block and pushes  $\bullet$ 's in state  $q_1$ . If yes,  $M$  enters state  $p$  and scans the  $b$ -block as well but now pushing  $B$ 's. If  $M$  never guesses yes, the computation is rejected on the endmarker. Otherwise, on reading the next  $\#$  or the endmarker,  $M$  enters state  $r$ . In this situation, the number of  $B$ 's on top of the pushdown corresponds to the length of the

$b$ -block guessed to be matched. Finally, for state  $r$  the symbols  $b$  and  $\#$  are translucent. Now  $M$  reads the  $a$ 's from left to right. For each  $a$  read, one  $B$  is popped. If and only if there are more  $a$ 's than  $B$ 's or vice versa then  $M$  accepts. So, we set:

- |   |   |
|---|---|
| (1) $\delta_D(q_0, b, \perp) = \{(q_1, \bullet), (p, B)\},$ | (7) $\delta_N(p, \triangleleft, B) = \{r\},$          |
| (2) $\delta_D(q_1, b, \bullet) = \{(q_1, \bullet)\},$       | (8) $\delta_R(r, a, B) = \{r\},$                      |
| (3) $\delta_N(q_1, \#, \bullet) = \{q_1, p\},$              | (9) $\delta_R(r, a, \bullet) = \text{accept},$        |
| (4) $\delta_D(p, b, \bullet) = \{(p, B)\},$                 | (10) $\delta_R(r, a, \perp) = \text{accept},$         |
| (5) $\delta_D(p, b, B) = \{(p, B)\},$                       | (11) $\delta_R(r, \triangleleft, B) = \text{accept}.$ |
| (6) $\delta_N(p, \#, B) = \{r\},$                           |   |

■

Deterministic pushdown automata with translucent letters have been defined in [9] also for the *non-returning* mode. In this mode, after a visible letter is processed, the input head does not return to the left end of the input, but it continues reading from the position of the visible letter just processed. Whenever the endmarker is reached and the transition on the endmarker yields a new state, the computation is continued in this state, with the input head placed at the left end of the remaining input. Such deterministic pushdown automata with translucent letters working in the non-returning mode generalize non-returning finite state automata with translucent letters [13]. Here, we also consider *input-driven pushdown automata with translucent letters working in the non-returning mode* (nrNIDPDawtl, nrDIDPDawtl).

Let  $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \perp, \tau, \delta_D, \delta_R, \delta_N \rangle$ , be an nrNIDPDawtl. Now, a configuration of  $M$  is a pair  $(uqw\triangleleft, \gamma)$  or  $\text{accept}$ , where  $q \in Q$  is the current state,  $uw \in \Sigma^*$  is the remaining part of the input with  $w$  being to the right and  $u$  to the left of the input head, and  $\gamma \in \Gamma^* \perp$  is the current pushdown content. The successor configuration yielded by  $\vdash$  is now specified as follows. Let  $q, q' \in Q$ ,  $x \in \tau(q)^*$ ,  $a \notin \tau(q)$ ,  $u, y \in \Sigma^*$ , and  $z \in \Gamma$ ,  $\gamma \in \Gamma^* \perp$ . Then:

1.  $(uqxay\triangleleft, z\gamma) \vdash (uxq'y\triangleleft, z'z\gamma)$ , if  $a \in \Sigma_D$  and  $(q', z') \in \delta_D(q, a, z)$ ,
2.  $(uqxay\triangleleft, \perp) \vdash (uxq'y\triangleleft, z'\perp)$ , if  $a \in \Sigma_D$  and  $(q', z') \in \delta_D(q, a, \perp)$ ,
3.  $(uqxay\triangleleft, z\gamma) \vdash (uxq'y\triangleleft, \gamma)$ , if  $a \in \Sigma_R$  and  $q' \in \delta_R(q, a, z)$ ,
4.  $(uqxay\triangleleft, \perp) \vdash (uxq'y\triangleleft, \perp)$ , if  $a \in \Sigma_R$  and  $q' \in \delta_R(q, a, \perp)$ ,
5.  $(uqxay\triangleleft, z\gamma) \vdash (uxq'y\triangleleft, z\gamma)$ , if  $a \in \Sigma_N$  and  $q' \in \delta_N(q, a, z)$ ,
6.  $(uqxay\triangleleft, \perp) \vdash (uxq'y\triangleleft, \perp)$ , if  $a \in \Sigma_N$  and  $q' \in \delta_N(q, a, \perp)$ ,
7.  $(uqx\triangleleft, z\gamma) \vdash (q'ux\triangleleft, z\gamma)$ , if  $q' \in \delta_N(q, \triangleleft, z)$ ,
8.  $(uqx\triangleleft, \perp) \vdash (q'ux\triangleleft, \perp)$ , if  $q' \in \delta_N(q, \triangleleft, \perp)$ .

In addition, whenever, the transition function yields  $\text{accept}$  for the current configuration, the successor configuration is  $\text{accept}$ .

The accepted language  $L(M)$  can be easily defined. Sometimes, we will be saying that an nrNIDPDawtl performs *sweeps*, where a sweep is a sequence of transitions that starts with the input head at the left end of the (remaining) input and ends after the next (if any) return move on the endmarker. Let us give an intuition on how an nrNIDPDawtl works by the following example.

**Example 2.** We consider two languages over the alphabet  $\{a, b\}$  together with its overlined variant  $\{\bar{a}, \bar{b}\}$  and its doubly-overlined variant  $\{\bar{\bar{a}}, \bar{\bar{b}}\}$ . In addition, we consider two mappings  $h_1$  and  $h_2$  such that  $h_1(a) = \bar{a}$ ,  $h_1(b) = \bar{b}$ ,  $h_2(a) = \bar{\bar{a}}$ , and  $h_2(b) = \bar{\bar{b}}$ . Then, the languages  $L_1$  and  $L_2$  are defined as follows.

$$\begin{aligned} L_1 &= \{wvh_2(w^R) \mid w \in \{a, b\}^+, v \in \{\bar{a}, \bar{b}\}^+\}, \\ L_2 &= \{wh_1(w^R)v \mid w \in \{a, b\}^+, v \in \{\bar{\bar{a}}, \bar{\bar{b}}\}^+\}. \end{aligned}$$

We will show that the union  $L = L_1 \cup L_2$  is accepted by a nondeterministic input-driven pushdown automaton with translucent input letters in the non-returning mode. We define an nrNIDPDawtl  $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \perp, \tau, \delta_D, \delta_R, \delta_N \rangle$  accepting  $L$  as follows.

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ ,
- $\Sigma_D = \{a, b\}$ ,  $\Sigma_R = \{\bar{a}, \bar{b}, \bar{\bar{a}}, \bar{\bar{b}}\}$ ,  $\Sigma_N = \emptyset$ ,
- $\Gamma = \{A, B\}$ ,
- $\tau(q_0) = \tau(q_2) = \tau(q_3) = \tau(q_4) = \tau(q_5) = \emptyset$ ,  $\tau(q_1) = \{\bar{a}, \bar{b}\}$ .

In a first phase,  $M$  reads input symbols  $a$  or  $b$  and pushes corresponding symbol  $A$  or  $B$  onto the pushdown store. At any time step,  $M$  decides nondeterministically whether it remains in this phase by remaining in state  $q_0$  or whether it starts to test whether the input belongs to the first set of the union  $L_1$  by entering state  $q_1$  or to the second set  $L_2$  by entering state  $q_3$ .

To test whether the input belongs to  $L_1$ ,  $M$  enters state  $q_1$  and all symbols  $\bar{a}$  or  $\bar{b}$  become translucent. Then,  $M$  starts to match all remaining doubly-overlined input symbols against the pushdown store. To ensure the correct format of the remaining input,  $M$  changes its state whenever the first doubly-overlined symbol is read. If  $M$  reaches the endmarker and the pushdown store is empty, the accepting state is entered. This is realized in rules (3)–(7).

To test whether the input belongs to  $L_2$ ,  $M$  enters state  $q_3$  and no symbols are translucent. Then,  $M$  starts to match all following single-overlined input symbols against the pushdown store. Once all such symbols are read and the pushdown store is empty,  $M$  proceeds to read the doubly-overlined symbols while popping from the (already empty) pushdown store. To ensure the correct format of the input,  $M$  changes its state whenever the first overlined and the first doubly-overlined symbol is read. If  $M$  encounters the endmarker and the pushdown store is empty,  $M$  enters the accepting state. This is realized in rules (8)–(16).

The transition functions are defined as follows for  $Z \in \{\perp, A, B\}$ .

- |  |  |
|--|--|
| (1) $\delta_D(q_0, a, Z) = \{(q_0, A), (q_1, A), (q_3, A)\}$ , | (9) $\delta_R(q_3, \bar{b}, B) = \{q_4\}$ ,                  |
| (2) $\delta_D(q_0, b, Z) = \{(q_0, B), (q_1, B), (q_3, B)\}$ , | (10) $\delta_R(q_4, \bar{a}, A) = \{q_4\}$ ,                 |
| (3) $\delta_R(q_1, \bar{\bar{a}}, A) = \{q_2\}$ ,              | (11) $\delta_R(q_4, \bar{b}, B) = \{q_4\}$ ,                 |
| (4) $\delta_R(q_1, \bar{\bar{b}}, B) = \{q_2\}$ ,              | (12) $\delta_R(q_4, \bar{\bar{a}}, \perp) = \{q_5\}$ ,       |
| (5) $\delta_R(q_2, \bar{\bar{a}}, A) = \{q_2\}$ ,              | (13) $\delta_R(q_4, \bar{\bar{b}}, \perp) = \{q_5\}$ ,       |
| (6) $\delta_R(q_2, \bar{\bar{b}}, B) = \{q_2\}$ ,              | (14) $\delta_R(q_5, \bar{\bar{a}}, \perp) = \{q_5\}$ ,       |
| (7) $\delta_N(q_2, \triangleleft, \perp) = \text{accept}$ ,    | (15) $\delta_R(q_5, \bar{\bar{b}}, \perp) = \{q_5\}$ ,       |
| (8) $\delta_R(q_3, \bar{a}, A) = \{q_4\}$ ,                    | (16) $\delta_N(q_5, \triangleleft, \perp) = \text{accept}$ . |

Clearly, if there is any error in the format of the input or the part of input compared to the pushdown store does not match, the transition function of  $M$  is not defined and, therefore, the input is rejected.

We would like to note that  $M$  uses its translucency to “overlook” the overlined part of an input belonging to  $L_1$ , since otherwise the pushdown store may be inadvertently emptied and could not be matched with the doubly-overlined input  $w^R$ . Moreover,  $M$  uses the non-returning mode to ensure that the input is correctly formatted, that is, symbols from  $\{a, b\}^+$  are followed by symbols from  $\{\bar{a}, \bar{b}\}^+$  which are in turn followed by symbols from  $\{\bar{\bar{a}}, \bar{\bar{b}}\}^+$ . ■

As is known for finite automata and regular pushdown automata [9], also for input-driven pushdown automata with translucent letters it holds that any automaton  $M$  working in the returning mode can be simulated by some automaton  $M'$  working in the non-returning mode, where both share the same

signature and the same translucency mapping (for the states of  $M$ ). The construction of  $M'$  for a given  $M$ , roughly speaking, works as follows: at each step,  $M'$  simulates the step of  $M$ , followed by a new step which brings the input head to the leftmost input symbol. To this end, let  $Q'$  be a primed copy of  $Q$ . The transition function  $\delta_i$ , for  $i \in \{D, R, N\}$ , is modified to  $\delta'_i$  so that the state  $q' \in Q'$  is entered if and only if  $M$  enters the state  $q \in Q$ . The translucency mapping  $\tau$  is extended to  $\tau'$  by adding  $\tau'(q') = \Sigma$ , for all  $q' \in Q'$ . This clearly implies that, whenever in any state from  $Q'$ ,  $M'$  sees the endmarker.

Finally,  $\delta'_N$  is extended by  $\delta'_N(q', \triangleleft, z) = \{q\}$ , for all  $q' \in Q'$  and all  $z \in \Gamma \cup \{\perp\}$ , thus bringing the input head to the leftmost position. One may easily verify that  $M'$  accepts if and only if  $M$  accepts.

### 3 Determinism versus Nondeterminism

It is well-known that for input-driven pushdown automata, deterministic devices are as powerful as their nondeterministic counterparts. This contrasts with the general case of pushdown automata, where the deterministic variant is strictly weaker than the nondeterministic one. In the following, we examine the situation for deterministic and nondeterministic input-driven pushdown automata with translucent input letters. It turns out that the family of languages accepted by deterministic input-driven pushdown automata with translucent input letters is a proper subset of the family of languages accepted by their nondeterministic counterparts, both in the returning and in the non-returning case.

**Theorem 3.** *The family  $\mathcal{L}(\text{DIDPDAwtl})$  is properly included in the family  $\mathcal{L}(\text{NIDPDAwtl})$  and the family  $\mathcal{L}(\text{nrDIDPDAwtl})$  is properly included in the family  $\mathcal{L}(\text{nrNIDPDAwtl})$ .*

*Proof.* We use the union  $L = L_1 \cup L_2$  as witness language for the properness of the inclusions, where  $L_1 = \{b^n \# b^m \sqcup a^n \mid m, n \geq 1\}$  and  $L_2 = \{b^m \# b^n \sqcup a^n \mid m, n \geq 1\}$ .

Similarly as in Example 1, two DIDPDAwtl's can be constructed that accept  $L_1$  respectively  $L_2$ , and an NIDPDAwtl can be constructed that accepts  $L = L_1 \cup L_2$ .

It remains to be shown that  $L$  is not accepted by any nrDIDPDAwtl. Assume in contrast to the assertion that  $L$  is accepted by some nrDIDPDAwtl  $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \perp, \tau, \delta_D, \delta_R, \delta_N \rangle$ . We consider accepting computations on inputs of the form  $a^n b^k \# b^\ell$  where  $n, k, \ell$  are large enough.

A basic observation is that  $M$  cannot access the  $b$ -block following the  $\#$  unless the  $\#$  is read or there are no more  $b$ 's in front of the  $\#$ .

First we claim that  $M$  cannot read the symbol  $\#$  and return to the left of the input before one or both  $b$ -blocks are read entirely.

Assume in contrast to the claim that  $M$  is in some configuration  $(a^{m_1} q_1 b^{k_1} \# b^{\ell_1} \triangleleft, \gamma_1)$  such that  $b \in \tau(q_1)$ , and from the successor configuration  $(a^{m_1} b^{k_1} q_2 b^{\ell_1} \triangleleft, \gamma_2)$  it reads some further  $b$ 's and then jumps to the endmarker reaching a configuration  $(q_3 a^{m_1} b^{k_1} b^{\ell_2} \triangleleft, \gamma_3)$ . If neither  $b^{k_1}$  nor  $b^{\ell_2}$  is empty then also the inputs  $a^n b^{k+1} \# b^{\ell-1}$  and  $a^n b^{k-1} \# b^{\ell+1}$  would be accepted but at least one of them does not belong to  $L$ . The contradiction shows the claim.

Next, we have to consider four cases.

**Case 1:** The first case is that neither  $a \in \Sigma_D$  and  $b \in \Sigma_R$  nor  $a \in \Sigma_R$  and  $b \in \Sigma_D$ . Essentially, this means that  $M$  does not use its pushdown.

A single sweep of  $M$  is analyzed. If  $M$  reads more than  $2|Q|$  consecutive symbols  $a$  (respectively  $b$ ), it must eventually enter a loop of, say  $c$ , states. Since the pushdown is not used, we can increase the length of the  $a$ -block (respectively  $b$ -block) by  $c$  without changing the overall result of the computation, a contradiction. Hence, in this case, in any sweep  $M$  cannot enter a loop while reading  $a$ 's or  $b$ 's. Therefore,  $M$  must perform multiple sweeps without loops. For the states in which the sweeps start there

are at most  $|Q|$  possibilities. Due to the deterministic behavior, eventually  $M$  will run through loops of sweeps with respect to the starting state of the loop. Let  $x_0$  be the number of  $a$ 's read before this sweep loop and let  $x_1$  be the number of  $a$ 's consumed in one sweep loop. Similarly, let  $y_0$  and  $y_1$  be the numbers of  $b$ 's consumed. Now we consider the input  $a^{x_0+x_1}b^{y_0+y_1}\#b^\ell$  where  $\ell > 2|Q|$ . Then, after the first sweep loop, the  $a$ -block and the first  $b$ -block are entirely read. If  $x_0 + x_1 \neq y_0 + y_1$ ,  $M$  must verify whether  $x_0 + x_1 = \ell$ . To do so,  $M$  needs to read the second block of  $b$ 's – but without using the pushdown store and without any leftover  $a$ 's. Given that  $\ell > 2|Q|$ ,  $M$  must enter a state loop while reading the second  $b$ -block, say of length  $c'$ . Hence, if  $M$  accepts the input for some  $\ell$ , it accepts the input with  $\ell + c'$ , which contradicts the definition of  $L$ . If  $x_0 + x_1 = y_0 + y_1$ , almost the same argument holds for the input  $a^{x_0+x_1}b^{y_0+y_1-1}\#b^\ell$ .

**Case 2:** The second case is that  $a \in \Sigma_D$  and  $b \in \Sigma_R$ . Moreover,  $M$  does not enter loops with respect to the current state and the topmost pushdown symbol.

In this case, in one sweep,  $M$  reads at most  $|Q|(|\Gamma| + 1)$  symbols of the  $a$ -block and the first  $b$ -block (as long as these blocks are non-empty). We argue similarly as in Case 1. Due to the deterministic behavior, eventually  $M$  will run through loops of sweeps with respect to the starting state of the loop. Let  $x_0$  be the number of  $a$ 's read before this sweep loop and let  $x_1$  be the number of  $a$ 's consumed in one sweep loop. Similarly, let  $y_0$  and  $y_1$  be the numbers of  $b$ 's consumed.

If  $x_0 + x_1 > y_0 + y_1$ , we consider the input  $a^{x_0+x_1}b^{x_0+x_1+y_0+y_1}\#b^\ell$  where  $\ell > 2|Q|$ . Then, after the first sweep loop, all  $a$ 's are read, and the remaining input is  $b^{x_0+x_1}\#b^\ell$ . Moreover, there are  $x_0 + x_1 - y_0 - y_1$  symbols left in the pushdown store. If  $x_0 + x_1 > y_0 + y_1$ , next  $M$  will empty its pushdown store after reading  $x_0 + x_1 - y_0 - y_1$  many  $b$ 's. Afterward,  $M$  can only rely on its finite set of states. Hence, after reading at most  $|Q|$  additional symbols  $b$ ,  $M$  enters a state loop of some length  $c''$ . Consequently, if  $M$  accepts the input for some  $\ell$  it must also accept the input with  $\ell + c''$ , which contradicts the definition of  $L$ .

The same reasoning applies for the input  $a^{x_0+x_1}b^{y_0+y_1}\#b^\ell$ , where  $\ell > 2|Q|$ , if  $x_0 + x_1 < y_0 + y_1$ , and for the input  $a^{x_0+x_1}b^{y_0+y_1-1}\#b^\ell$ , where  $\ell > 2|Q|$ , if  $x_0 + x_1 = y_0 + y_1$ .

**Case 3:** The third case is that  $a \in \Sigma_D$  and  $b \in \Sigma_R$  and  $M$  enters a loop with respect to the current state and the topmost pushdown symbol while reading an  $a$ -block or a  $b$ -block.

First, consider the  $a$ -block. After reading the  $a$ -block,  $M$  has  $n$  symbols stored in the pushdown. In a sweep where the  $\#$  symbol is read,  $M$  must either read the first  $b$ -block or the second  $b$ -block entirely. Assume it reads the first  $b$ -block and consider the input  $a^n b^k \# b^\ell$  with  $k > n$ . Then, after reading this block, the pushdown store is empty, and the rest of the input must be processed using only the finite control. As a result, the length of the second  $b$ -block can be increased which is again a contradiction. The same argument applies analogously if the second  $b$ -block is read first.

Second, consider one of the  $b$ -blocks. Since  $b \in \Sigma_R$ ,  $M$  runs through a state loop with empty pushdown while processing the block. So the length of the block can be increased by the length of the loop without changing the overall result of the computation, a contradiction.

**Case 4:** The fourth case is that  $a \in \Sigma_R$  and  $b \in \Sigma_D$  and  $M$  enters a loop with respect to the current state and the topmost pushdown symbol while reading an  $a$ -block or a  $b$ -block. The argumentation in this case is symmetric to Case 3. Here the roles of  $a$  and  $b$  (that is, their memberships in  $\Sigma_D$  and  $\Sigma_R$ ) are switched.

Finally, from the contradictions in all possible cases we conclude that  $L$  is not accepted by any  $\text{nrDIPDA}_{\text{wtl}}$ .  $\square$

The additional power of nondeterministic input-driven pushdown automata with translucent input letters versus their deterministic counterpart is due to the fact that the translucency of input letters al-



lows different computation paths to treat the same input symbol differently, thereby enabling different operations on the pushdown store. In a nondeterministic setting, this flexibility permits branching into multiple computational paths, each potentially using the pushdown store in a distinct way. However, a deterministic machine cannot simulate these differing operations simultaneously, which limits its expressive power.

## 4 Returning versus Non-Returning

It is known that deterministic pushdown automata with translucent letters working in the non-returning mode can accept even a non-semilinear language [9]. So a natural question is whether the same still holds for the structurally weaker device which has to obey the input-driven mode. The next theorem answers this question in the affirmative. To this end, we tweak the witness language from [9]. We define the non-semilinear language  $L_{nsl}$  as

$$L_{nsl} = \{ a \$ \# a^3 \$ \#^2 a^5 \$ \#^3 \dots \$ \#^{k-1} a^{2k-1} \$ \#^k a^{2k+1} \mid k \geq 0 \}.$$

**Theorem 4.** *The language  $L_{nsl}$  is accepted by some  $nrDIDPDA_{wtl}$ .*

The language  $L_{nsl}$  is accepted by some  $nrDIDPDA_{wtl}$  and thus by some  $nrNIDPDA_{wtl}$ . It is known that all languages accepted by  $DPDA_{wtl}$  are semilinear [9]. So, the next question is whether this is still true when we trade nondeterminism for the input-driven property. The following result has been shown in [15] for finite automata with translucent letters and can be adapted for our purposes.

**Proposition 5.** *From any given  $NIDPDA_{wtl}$   $M$ , an  $NPDA$   $M'$  can effectively be constructed, such that  $L(M') \subseteq L(M)$  and  $L(M')$  is letter-equivalent to  $L(M)$ .*

Proposition 5 together with the well-known result that all context-free languages are semilinear [24] implies that all languages in  $\mathcal{L}(NIDPDA_{wtl}) \supset \mathcal{L}(DIDPDA_{wtl})$  are semilinear. So, by Theorem 4 we have the following corollary.

**Corollary 6.** *The family  $\mathcal{L}(DIDPDA_{wtl})$  is properly included in  $\mathcal{L}(nrDIDPDA_{wtl})$ , and the family  $\mathcal{L}(NIDPDA_{wtl})$  is properly included in  $\mathcal{L}(nrNIDPDA_{wtl})$ .*

The remaining two language families under consideration to be compared are  $\mathcal{L}(nrDIDPDA_{wtl})$  and  $\mathcal{L}(NIDPDA_{wtl})$ .

**Proposition 7.** *The language families  $\mathcal{L}(nrDIDPDA_{wtl})$  and  $\mathcal{L}(NIDPDA_{wtl})$  are incomparable.*

*Proof.* Theorem 4 provides a non-semilinear language accepted by some  $nrDIDPDA_{wtl}$  but not accepted by any  $NIDPDA_{wtl}$ .

Conversely, the proof of Theorem 3 provides a language that is accepted by some  $NIDPDA_{wtl}$  but cannot be accepted by any  $nrDIDPDA_{wtl}$ .  $\square$

## 5 Closure under Boolean Operations

Often nondeterministic devices induce language families that are closed under union but are not closed under intersection. This implies immediately the non-closure under complementation. However, here the closure under union is an open problem. Therefore, we show the non-closure under complementation directly. A witness for the nondeterministic families  $\mathcal{L}(nrNIDPDA_{wtl})$  and  $\mathcal{L}(NIDPDA_{wtl})$  is the language

$$L_{rep} = \{ b^n (\# b^n)^k \sqcup a^n \mid n \geq 1, k \geq 0 \}.$$

**Theorem 8.** *The language families  $\mathcal{L}(\text{nrNIDPDAwtl})$  and  $\mathcal{L}(\text{NIDPDAwtl})$  are not closed under complementation.*

It is well known that the families of languages induced by deterministic pushdown automata and real-time deterministic pushdown automata are closed under complementation. The closure has also been derived for DPDAwtl and nrDPDAwtl [9]. Here we complement these results by showing the closure for the deterministic language families studied here.

**Proposition 9.** *The language families  $\mathcal{L}(\text{nrDIDPDAwtl})$  and  $\mathcal{L}(\text{DIDPDAwtl})$  are closed under complementation.*

Next, we show that both deterministic families are not closed under the remaining Boolean operations union and intersection.

**Proposition 10.** *The language families  $\mathcal{L}(\text{nrDIDPDAwtl})$  and  $\mathcal{L}(\text{DIDPDAwtl})$  are neither closed under union nor under intersection.*

*Proof.* We use the languages  $L_1 = \{b^n \# b^m \sqcup a^n \mid m, n \geq 1\}$  and  $L_2 = \{b^m \# b^n \sqcup a^n \mid m, n \geq 1\}$  introduced in the proof of Theorem 3. Each language can be accepted by a DIDPDAwtl with signature  $\Sigma_D = \{a\}$ ,  $\Sigma_R = \{b\}$ , and  $\Sigma_N = \{\#\}$ . The rough idea for a DIDPDAwtl  $M_1$  accepting  $L_1$  is to push in a first phase all  $a$ 's while symbols  $b$  and  $\#$  are translucent. In a second phase, the first block of  $b$ 's is matched against the pushdown store and the second block of  $b$ 's is in fact ignored since  $M_1$  may pop from the empty pushdown only.

The rough idea for a DIDPDAwtl  $M_2$  accepting  $L_2$  is to consume all  $b$ 's up to and including the symbol  $\#$  in a first phase. At the end of this phase the pushdown store is empty. In a second phase, all symbols  $b$  are translucent and the  $a$ 's are consumed and pushed. Finally, in a third phase, the remaining  $b$ 's from the input are matched against the pushdown store.

Since it is shown in Theorem 3 that the union  $L_1 \cup L_2$  is not accepted by any nrDIDPDAwtl, we obtain the non-closure under union for  $\mathcal{L}(\text{DIDPDAwtl})$  as well as for  $\mathcal{L}(\text{nrDIDPDAwtl})$ , even if the given automata have identical signatures. Since both families are closed under complementation by Proposition 9, we also obtain the non-closure under intersection for  $\mathcal{L}(\text{DIDPDAwtl})$  as well as for  $\mathcal{L}(\text{nrDIDPDAwtl})$ .  $\square$

For the nondeterministic families we already know the non-closure under complementation. We now show that the nondeterministic returning class is not closed under intersection even with regular languages. To this end, we use the result of Proposition 5 stating that from any given NIDPDAwtl  $M$  we can effectively construct an NPDA  $M'$  such that  $L(M') \subseteq L(M)$  and  $L(M')$  is letter-equivalent to  $L(M)$ . Since the context-sensitive language  $L_{abc} = \{a^n b^n c^n \mid n \geq 0\}$  does not contain any letter-equivalent context-free sub-language, we can conclude that  $L_{abc} \notin \mathcal{L}(\text{NIDPDAwtl})$ . On the other hand,  $L_{abc} = L_{a,b,c} \cap a^* b^* c^*$  with  $L_{a,b,c} = \{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}$  and  $L_{a,b,c}$  as well as the regular language  $a^* b^* c^*$  can be accepted by NIDPDAwtl having identical signatures. Hence, we obtain the following proposition.

**Proposition 11.** *The language family  $\mathcal{L}(\text{NIDPDAwtl})$  is neither closed under intersection nor under intersection with regular languages.*

It remains an open question whether or not the family  $\mathcal{L}(\text{NIDPDAwtl})$  is closed under union and whether or not the family  $\mathcal{L}(\text{nrNIDPDAwtl})$  is closed under union or intersection. Obviously, we obtain the closure under union for both families if the signatures are compatible. However, we strongly conjecture non-closure in all other cases.

## 6 Decidability Questions

In this section, we investigate decidability questions such as, for example, emptiness, finiteness, universality, inclusion, equivalence, and regularity for our introduced input-driven variants of pushdown automata with translucent letters. These decidability questions have already been investigated for deterministic and nondeterministic finite automata with translucent letters in [16, 18] where some partial results have been obtained. For returning deterministic and nondeterministic finite automata with translucent letters the questions of emptiness and finiteness are decidable. In addition, universality is decidable in the deterministic case. Inclusion is already undecidable for returning deterministic finite automata with translucent letters. This negative result carries over to all other models in the returning and/or non-deterministic case. For returning nondeterministic finite automata with translucent letters the questions of equivalence and regularity are undecidable and carry over to non-returning nondeterministic finite automata with translucent letters as well. Here, we study these questions for pushdown automata with translucent letters. We note that some of the undecidability results obtained for finite automata with translucent letters carry over to pushdown automata with translucent letters. However, we show here the non-semidecidability of the problems and, in addition, the non-semidecidability of universality for nondeterministic input-driven pushdown automata in the returning and non-returning case. We start with decidable questions and show that some questions are decidable for returning pushdown automata with translucent letters.

**Theorem 12.** *For DIDPDawtl or DPDAwtl as input, the problems of testing emptiness, finiteness, and universality are decidable. For NIDPDawtl as input, the problems of testing emptiness and finiteness are decidable.*

Next, we switch to undecidability results and we will show, in particular, the non-semidecidability of some problems. To prove these results we use the technique of valid computations of Turing machines [5]. It suffices to consider deterministic Turing machines with one single read-write head and one single tape whose space is fixed by the length of the input, that is, so-called linear bounded automata (LBA). Without loss of generality and for technical reasons, we assume that the LBAs can halt only after an odd number of moves, accept by halting, and make at least three moves. 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 LBA  $M$ , where  $q_0$  is the initial state,  $T \cap Q = \emptyset$  is the tape alphabet containing the endmarkers  $\triangleright$  and  $\triangleleft$ , and  $\Sigma \subset T$  is the input alphabet. A configuration of  $M$  can be written as a string of the form  $\triangleright T^* Q T^* \triangleleft$  such that,  $\triangleright t_1 t_2 \cdots t_i q_{i+1} \cdots t_n \triangleleft$  is used to express that  $\triangleright t_1 t_2 \cdots t_n \triangleleft$  is the tape inscription,  $M$  is in state  $q$ , and is scanning tape symbol  $t_{i+1}$ .

The set of valid computations  $\text{VALC}(M)$  is now defined to be the set of words having the form  $w_0 \# w_2 \# \cdots \# w_{2m} c w_{2m+1}^R \# w_{2m-1}^R \# \cdots \# w_3^R \# w_1^R$ , where  $\#, c \notin T \cup Q$ ,  $w_i \in \triangleright T^* Q T^* \triangleleft$  are configurations of  $M$ ,  $w_0$  is an initial configuration from  $\triangleright q_0 \Sigma^* \triangleleft$ ,  $w_{2m+1}$  is a halting, that is, an accepting configuration, and  $w_{i+1}$  is the successor configuration of  $w_i$  for  $0 \leq i \leq 2m$ . The set of invalid computations  $\text{INVALC}(M)$  is defined as the complement of  $\text{VALC}(M)$  with respect to the alphabet  $T \cup Q \cup \{\#, c\}$ .

To accept the set  $\text{INVALC}(M)$  by an NIDPDawtl we make some modifications. Let  $h'$  be a mapping that maps every symbol from  $T \cup Q \cup \{\#\}$  to its primed version. Similarly, let  $h''$  be a mapping that maps every symbol from  $T \cup Q \cup \{\#\}$  to its double-primed version. We then define the set of valid computations  $\text{VALC}'(M)$  to be the set of words of the form  $w_0 \# w_2 \# \cdots \# w_{2m} c (h'(w_{2m+1}^R \#) \sqcup h''(w_{2m-1}^R \# \cdots \# w_3^R \# w_1^R))$ , where  $w_0 \# w_2 \# \cdots \# w_{2m} c w_{2m+1}^R \# w_{2m-1}^R \# \cdots \# w_3^R \# w_1^R$  belongs to  $\text{VALC}(M)$ . The set of invalid computations  $\text{INVALC}'(M)$  is then defined as the complement of  $\text{VALC}'(M)$ .

**Lemma 13.** *Let  $M$  be an LBA. Then, an NIDPDawtl accepting  $\text{INVALC}'(M)$  can effectively be constructed.*

*Proof.* We sketch the construction of an NIDPDawtl  $M'$  accepting  $\text{INVALC}'(M)$  whose signature is defined as  $\Sigma_D = T \cup Q \cup \{\#\}$ ,  $\Sigma_N = \{c\}$ , and  $\Sigma_R = T' \cup T'' \cup Q' \cup Q'' \cup \{\#', \#''\}$ . To check whether an input  $x$  belongs to  $\text{INVALC}'(M)$ ,  $M'$  guesses and tests one of the following four possibilities.

1.  $x$  has the wrong format to belong to  $\text{VALC}'(M)$ .
2.  $x$  has the correct format, but the number of configurations to the left of  $c$  is different from the number of configurations to the right of  $c$ .
3.  $x = w_0\#w_2\#\dots\#w_{2m}c(h'(w_{2m+1}^R\#) \sqcup h''(w_{2m-1}^R\#\dots\#w_3^R\#w_1^R))$ , but  $w_{2i+1}$  is not the successor configuration of  $w_{2i}$  for some  $0 \leq i \leq m$ .
4.  $x = w_0\#w_2\#\dots\#w_{2m}c(h'(w_{2m+1}^R\#) \sqcup h''(w_{2m-1}^R\#\dots\#w_3^R\#w_1^R))$ , but  $w_{2i+2}$  is not the successor configuration of  $w_{2i+1}$  for some  $0 \leq i \leq m-1$ .

The first possibility can be tested by a finite automaton and, hence, by  $M'$  disregarding the actions on the pushdown store. For the second possibility  $M'$  acts as follows: it reads the input up to the middle marker  $c$  and pushes the input as it is on the pushdown store. After reading the marker  $c$ ,  $M'$  makes all symbols from  $T'' \cup Q'' \cup \{\#''\}$  translucent and pops for every input symbol from  $T' \cup Q' \cup \{\#'\}$  a symbol from the pushdown store taking care that  $\#'$  in the input is only matched against  $\#$  on the pushdown store. If an error is encountered in this phase, the input is accepted. If  $M'$  sees the endmarker,  $M'$  reads the remaining input and pops for every input symbol from  $T'' \cup Q'' \cup \{\#''\}$  a symbol from the pushdown store taking again care that  $\#''$  in the input is only matched against  $\#$  on the pushdown store. If an error is encountered in this phase, the input is accepted as well. If the input is read completely and the pushdown store is not empty, or the pushdown store gets empty before the input is read completely,  $M'$  accepts as well and rejects in all other cases.

To test the third possibility  $M'$  reads the input up to the middle marker  $c$  and pushes the input as it is on the pushdown store. Additionally, at some point of time  $M'$  guesses the index  $i$ . Then,  $M'$  pushes configuration  $w_{2i}$  with suitably marked symbols on the pushdown store and  $M'$  remembers the last three symbols read in its finite control until the state symbol of configuration  $w_{2i}$  is the middle one of these three. After reading the middle marker  $c$  the task is to identify configuration  $w_{2i+1}$  in the input and to check that  $w_{2i+1}$  is not the successor configuration of  $w_{2i}$ . If the suitably marked configuration on the pushdown store is the topmost one after reading  $c$ ,  $M'$  makes all symbols from  $T'' \cup Q'' \cup \{\#''\}$  translucent and pops for every input symbol from  $T' \cup Q' \cup \{\#'\}$  a symbol from the pushdown store verifying that the current configuration is *not* the reversal of the successor configuration of the configuration stored in the

Automata Family	$\emptyset$	FIN	$\Sigma^*$	$\subseteq$	$=$	REG
DPDAwtl	✓	✓	✓	×	?	?
DIDPDawtl	✓	✓	✓	×	?	?
NIDPDawtl	✓	✓	✗	✗	✗	✗
nrDPDAwtl	?	?	?	×	?	?
nrDIDPDawtl	?	?	?	×	?	?
nrNIDPDawtl	?	?	✗	✗	✗	✗

Table 1: A summary of decidability questions for the language families discussed in this paper. The undecidable questions derived from finite automata with translucent letters are marked with '✗', whereas the non-semidecidable questions obtained in this paper are marked with '✗'.

pushdown store. 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. If an error is encountered, the input is accepted and otherwise rejected. If the suitably marked configuration on the pushdown store is not the topmost one after reading  $c$ , then  $M'$  makes all symbols from  $T'' \cup Q'' \cup \{\#\}$  translucent and pops for every input symbol from  $T' \cup Q' \cup \{\#\}$  a symbol from the pushdown store checking the correct length and format as in the test of the second possibility. After this phase handling inputs from  $T' \cup Q' \cup \{\#\}$ ,  $M'$  reads the remaining input and pops for every input symbol from  $T'' \cup Q'' \cup \{\#\}$  a symbol from the pushdown store checking again the correct length and format as in the test of the second possibility until the suitably marked symbols appear on the pushdown store. In this case,  $M'$  pops for every input symbol from  $T'' \cup Q'' \cup \{\#\}$  a symbol from the pushdown store verifying that the current configuration is *not* the reversal of the successor configuration of the configuration stored in the pushdown store. Again, this can be computed and verified due to the information remembered in the finite control, since both configurations differ only locally at the state symbol. If an error is encountered, the input is accepted and otherwise rejected.

The idea to test the fourth possibility is in a first phase identical to the third possibility:  $M'$  reads the input up to the middle marker  $c$  and pushes the input as it is on the pushdown store. Additionally,  $M'$  pushes configuration  $w_{2i+2}$  with suitably marked symbols and remembers the last three symbols read in its finite control until the state symbol of configuration  $w_{2i+2}$  is the middle one of these three. After reading the middle marker  $c$  the task is to identify configuration  $w_{2i+1}$  in the input and to check that  $w_{2i+1}$  is not the successor configuration of  $w_{2i+2}$ . To this end,  $M'$  makes all symbols from  $T' \cup Q' \cup \{\#\}$  translucent and pops for every input symbol from  $T'' \cup Q'' \cup \{\#\}$  a symbol from the pushdown store checking the correct length and format as in the test of the second possibility until the suitably marked symbols appear on the pushdown store. In this case,  $M'$  pops for every input symbol from  $T'' \cup Q'' \cup \{\#\}$  a symbol from the pushdown store verifying that the reversal of the successor configuration of the current configuration is *not* the configuration stored in the pushdown store. Again, this can be computed and verified due to the information remembered in the finite control, since both configurations differ only locally at the state symbol. If an error is encountered, the input is accepted and otherwise rejected. We note that it is implicitly detected by possibilities 3 and 4 if all configurations do not have the same length. This completes the construction of the NIDPDawtl  $M'$  accepting  $\text{INVALC}'(M)$ .  $\square$

The fact that NIDPDawtl accept the set of invalid computations of an LBA is sufficient to obtain the next non-semidecidability results.

**Theorem 14.** *For NIDPDawtl or nrNIDPDawtl as input, the problems of testing universality, inclusion, equivalence, and regularity are not semidecidable.*

## References

- [1] Simon Beier & Markus Holzer (2022): *Nondeterministic right one-way jumping finite automata*. *Inform. Comput.* 284, p. 104687, doi:10.1016/J.IC.2021.104687.
- [2] Suna Bensch, Henning Bordihn, Markus Holzer & Martin Kutrib (2009): *On input-revolving deterministic and nondeterministic finite automata*. *Inform. Comput.* 207, pp. 1140–1155, doi:10.1016/j.ic.2009.03.002.
- [3] Burchard von Braunmühl & Rutger Verbeek (1985): *Input-Driven Languages are Recognized in  $\log n$  Space*. In Marek Karpinski & Jan van Leeuwen, editors: *Topics in the Theory of Computation, Mathematics Studies* 102, North-Holland, Amsterdam, pp. 1–19, doi:10.1016/S0304-0208(08)73072-X.
- [4] Hiroyuki Chigahara, Szilárd Zsolt Fazekas & Akihiro Yamamura (2016): *One-Way Jumping Finite Automata*. *Int. J. Found. Comput. Sci.* 27, pp. 391–405, doi:10.1142/S0129054116400165.

- [5] Juris Hartmanis (1967): *Context-free languages and Turing machine computations*. *Proc. Symposia in Applied Mathematics* 19, pp. 42–51, doi:10.1090/psam/019/0235938.
- [6] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts.
- [7] Petr Jančar, František Mráz, Martin Plátek & Jörg Vogel (1995): *Restarting automata*. In Horst Reichel, editor: *Fundamentals of Computation Theory (FCT 1995)*, LNCS 965, Springer, pp. 283–292, doi:10.1007/3-540-60249-6\_60.
- [8] Martin Kutrib, Andreas Malcher, Carlo Mereghetti & Beatrice Palano (2025): *Two-Way Finite Automata with Translucent Input Letters*. In Andreas Malcher & Luca Prigioniero, editors: *Descriptive Complexity of Formal Systems (DCFS 2025)*, LNCS 15759, Springer, pp. 151–165, doi:10.1007/978-3-031-97100-6\_11.
- [9] Martin Kutrib, Andreas Malcher, Carlo Mereghetti, Beatrice Palano, Priscilla Raucci & Matthias Wendlandt (2024): *Deterministic Pushdown Automata with Translucent Input Letters*. In Joel D. Day & Florin Manea, editors: *Developments in Language Theory (DLT 2024)*, LNCS 14791, Springer, pp. 203–217, doi:10.1007/978-3-031-66159-4\_15.
- [10] Martin Kutrib, Andreas Malcher, Carlo Mereghetti, Beatrice Palano, Priscilla Raucci & Matthias Wendlandt (2024): *On Properties of Languages Accepted by Deterministic Pushdown Automata with Translucent Input Letters*. In Szilárd Zsolt Fazekas, editor: *Implementation and Application of Automata (CIAA 2024)*, LNCS 15015, Springer, pp. 208–220, doi:10.1007/978-3-031-71112-1\_15.
- [11] Alexander Meduna & Petr Zemek (2012): *Jumping finite automata*. *Int. J. Found. Comput. Sci.* 23, pp. 1555–1578, doi:10.1142/S0129054112500244.
- [12] Kurt Mehlhorn (1980): *Pebbling Mountain Ranges and its Application of DCFL-Recognition*. In J. W. de Bakker & Jan van Leeuwen, editors: *International Colloquium on Automata, Languages and Programming (ICALP 1980)*, LNCS 85, Springer, pp. 422–435, doi:10.1007/3-540-10003-2\_89.
- [13] František Mráz & Friedrich Otto (2023): *Non-returning deterministic and nondeterministic finite automata with translucent letters*. *RAIRO Theor. Informatics Appl.* 57, p. 8, doi:10.1051/ITA/2023009.
- [14] Benedek Nagy & Friedrich Otto (2011): *CD-systems of stateless deterministic  $R(1)$ -automata governed by an external pushdown store*. *RAIRO Theor. Informatics Appl.* 45, pp. 413–448, doi:10.1051/ITA/2011123.
- [15] Benedek Nagy & Friedrich Otto (2011): *Finite-state Acceptors with Translucent Letters*. In G. Bel-Enguix, V. Dahl & A.O. De La Puente, editors: *International Workshop on AI Methods for Interdisciplinary Research in Language and Biology (BILC 2011)*, SciTePress, pp. 3–13, doi:10.5220/0003272500030013.
- [16] Benedek Nagy & Friedrich Otto (2012): *On CD-systems of stateless deterministic  $R$ -automata with window size one*. *J. Comput. Syst. Sci.* 78, pp. 780–806, doi:10.1016/J.JCSS.2011.12.009.
- [17] Benedek Nagy & Friedrich Otto (2013): *Deterministic pushdown-CD-systems of stateless deterministic  $R(1)$ -automata*. *Acta Inform.* 50, pp. 229–255, doi:10.1007/S00236-012-0175-X.
- [18] Benedek Nagy & Friedrich Otto (2013): *Globally deterministic CD-systems of stateless  $R$ -automata with window size 1*. *Int. J. Comput. Math.* 90(6), pp. 1254–1277, doi:10.1080/00207160.2012.688820.
- [19] Benedek Nagy & Friedrich Otto (2024): *Finite Automata with Sets of Translucent Words*. In Joel D. Day & Florin Manea, editors: *Developments in Language Theory (DLT 2024)*, LNCS 14791, Springer, pp. 236–251, doi:10.1007/978-3-031-66159-4\_17.
- [20] Alexander Okhotin & Kai Salomaa (2014): *Complexity of input-driven pushdown automata*. *SIGACT News* 45, pp. 47–67, doi:10.1145/2636805.2636821.
- [21] Friedrich Otto (2015): *On Visibly Pushdown Trace Languages*. In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater & Roger Wattenhofer, editors: *SOFSEM 2015*, LNCS 8939, Springer, pp. 389–400, doi:10.1007/978-3-662-46078-8\_32.
- [22] Friedrich Otto (2023): *A Survey on automata with translucent letters*. In Benedek Nagy, editor: *Implementation and Application of Automata (CIAA 2023)*, LNCS 14151, Springer, pp. 21–50, doi:10.1007/978-3-031-40247-0\_2.

- [23] Friedrich Otto (2025): *Restarting Automata*. Springer, Cham, Switzerland, doi:10.1007/978-3-031-78701-0.
- [24] Rohit J. Parikh (1966): *On Context-Free Languages*. *J. ACM* 13, pp. 570–581, doi:10.1145/321356.321364.