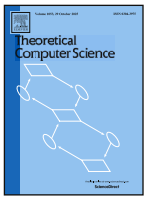
Theoretical Computer Science 1072 (2026) 115890

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs



On properties of languages accepted by deterministic pushdown automata with translucent input letters ⋆

Martin Kutrib [a,*], Andreas Malcher [a], Carlo Mereghetti [b,c], Beatrice Palano [b,c], Priscilla Raucci [a], Matthias Wendlandt [a]

[a] Institut für Informatik, Universität Giessen Arndtstraße 2, 35392, Giessen, Germany
[b] Dipartimento di Informatica Giovanni Degli Antoni, Universitá degli Studi di Milano, via Celoria 18, 20133, Milano, Italy
[c] Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM), Italy

ARTICLE INFO

Editor: Lila Kari

Keywords:
Translucent input letters
Deterministic pushdown automata
Returning/non-returning computations
Computational capability
Closure properties

ABSTRACT

We study deterministic pushdown automata operating with *translucent* input letters. These devices can be obtained by equipping classical deterministic pushdown automata with a *translucency function* which, depending on the current state, establishes the set of invisible input symbols: such invisible symbols are skipped in the current move and dealt with in subsequent sweeps, while the first visible symbol from the current input head position rightward is processed and consumed. Deterministic pushdown automata with translucent input letters can be *returning*, meaning that a new input sweep starts from the leftmost input symbol left on the tape immediately after processing a visible symbol, or *non-returning*.

We show some incomparability results between the acceptance capabilities of returning and non-returning deterministic pushdown automata with translucent input letters and those of non-returning deterministic and nondeterministic finite state automata with translucent input letters. Then, we prove the non-closure of the families of languages accepted by returning and non-returning deterministic pushdown automata with translucent input letters under: concatenation, Kleene star, length-preserving and inverse homomorphism, reversal, and intersection with regular languages. In particular, arguments used to prove the non-closure under intersection with regular languages enable us to answer a question left open in the literature, on the recognition power of non-returning deterministic finite state automata with translucent input letters.

1. Introduction

The standard 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/reject outcome upon sweeping the whole input. Starting from this basic input acquisition mode, more "relaxed" modes have been deeply investigated. So, for instance, two-way motion, the possibility of pausing input reading (stationary or λ-moves) or accepting in the middle of the string, multiple input sweeps, rotating, sweeping and restarting modes, have

⋆ A preliminary version of this work [10] was presented at the *28th International Conference on Implementation and Application of Automata (CIAA)*, September 3–6, 2024, Akita, Japan.

* Corresponding author.
E-mail addresses: kutrib@uni-giessen.de (M. Kutrib), andreas.malcher@informatik.uni-giessen.de (A. Malcher), carlo.mereghetti@unimi.it (C. Mereghetti), beatrice.palano@unimi.it (B. Palano), priscilla.raucci@uni-giessen.de (P. Raucci), matthias.wendlandt@informatik.uni-giessen.de (M. Wendlandt).

https://doi.org/10.1016/j.tcs.2026.115890
Received 16 July 2025; Received in revised form 24 February 2026; Accepted 6 March 2026
Available online 13 March 2026

been considered for a wide variety of machines (the reader is referred to, e.g., [2,4–8,18] for an overview of these and other models). In all these cases, meaningful differences in the accepting and/or descriptional power have been emphasized, stating that the way of acquiring input is truly a computational resource that can be used to tune computational and descriptional power.

Particularly interesting is the *discontinuous* form of parsing, where "jumping" to any position inside the input string is allowed at any move. This paradigm has been suggested for Turing machines in [19], and recently considered for finite state devices. The notion of a *jumping finite automaton* is introduced in [11], where it is proved that the jumping feature increases the accepting power. In fact, even non-context-free languages, such as the context-sensitive language $L = \{ w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c \}$, can be accepted by deterministic jumping finite automata. The restricted variant of "right one-way jumping" automata has been considered in [1,3].

A way of achieving the jumping feature is by using *translucent letters*. The concept of translucent letters has been introduced by Nagy and Otto in [14] for deterministic and nondeterministic finite automata. The key feature for devices with translucent input letters is provided by a translucency function establishing 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 rightward, up to the first non-translucent letter (thus realizing a jump). After processing and consuming 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 rightward, according to its updated current state and the corresponding translucent symbols. In both modes, the input head returns to the left end of the input whenever the right end of the input is reached.

Deterministic and nondeterministic finite automata with translucent input letters are deeply investigated in the literature (see, e.g., [13,15–17]). However, many questions are still open. Interestingly, the jumping and translucency paradigms have been put together in [12], to form the model of a deterministic or nondeterministic jumping finite automaton.

In this paper, we study *deterministic pushdown automata with translucent input letters* in both returning and non-returning mode (DPDAwtl and nrDPDAwtl, respectively). Very roughly speaking, these devices can be obtained by equipping classical deterministic pushdown automata (DPDA) with a translucency function. Returning pushdown automata with translucent input letters have been considered by Nagy and Otto in [15,16] in terms of certain cooperating distributed systems of restarting automata with additional pushdown store. This addresses translucency as a tool to connect several language definition formalisms, besides modeling discontinuous (jump) processing paradigms as above discussed.

We emphasize that, due to research purposes, the definition of returning pushdown automata with translucent input letters provided by Nagy and Otto does not encompass λ-transitions, plus it features acceptance by empty pushdown. Instead, to be closer to the classical definition of DPDA, we choose to define DPDAwtl and nrDPDAwtl possibly performing λ-transitions, and accepting by accepting states. In [9], we have begun investigating DPDAwtl and nrDPDAwtl by first formally defining these models. Next, we have assessed their acceptance capabilities, by showing that the families of languages accepted by such devices, namely $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl), properly lay between deterministic context-free and deterministic context-sensitive languages, with nrDPDAwtl being strictly more powerful than DPDAwtl. To have a better grasp of DPDAwtl and nrDPDAwtl acceptance power, we have also established (incomparability) relations among $\mathscr{L}$(DPDAwtl), $\mathscr{L}$(nrDPDAwtl) and other well known language families between deterministic context-free and deterministic context-sensitive, such as context-free, growing context-sensitive and Church-Rosser languages. We have than tackled Boolean closure properties of $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl), showing that both these families are closed under complementation but not under union and intersection.

Here, we are going to further deepen the study of both accepting capabilities and closure properties of DPDAwtl and nrDPDAwtl. Concerning the first topic, we show that $\mathscr{L}$(DPDAwtl) and the corresponding language family for non-returning deterministic finite state automata with translucent input letters are incomparable. Incomparability is also obtained, between $\mathscr{L}$(nrDPDAwtl) and the corresponding language family for non-returning nondeterministic finite state automata with translucent input letters. These results, together with those in [9], provide a fine-grained analysis of the family of languages sitting between regular, deterministic context-free, and deterministic context-sensitive languages (see Fig. 1, page 18). Therefore, from this viewpoint, translucency might be regarded as a new and promising tool to single out the inner structure of language families.

Next, we turn to study the closure of $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) under very well known language operations, namely: concatenation, Kleene star, length-preserving and inverse homomorphism, and reversal. We obtain that both these language families are not closed under such operations. This, together with results in [9,13,17], yields an almost complete account on the possibility of implementing main language operations on finite state and pushdown devices in presence of translucency (see Table 5, page 26).

We feel it worth remarking that, among closure properties, here we also investigate the closure of $\mathscr{L}$(nrDPDAwtl) under intersection with regular languages, obtaining a negative answer. The arguments we use to get this negative result enable us to prove that even the family of languages accepted by non-returning deterministic finite state automata is not closed under intersection with regular languages. This solves an open problem posed by Mráz and Otto in [13].

The paper is organized as follows. Section 2 contains basic definitions and models formal statement. In Section 3, we show some preliminary results on the acceptance capabilities of deterministic pushdown automata with translucent input letters, useful in our future investigations. In Section 4, we compare the accepting power of DPDAwtl, nrDPDAwtl, and that of corresponding deterministic and nondeterministic finite state automata with translucent input letters. Finally, in Section 5, we analyze closure properties for the language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl).

The present paper updates and expands the conference version [10], by providing the proofs of all the main results in full details. As it will be noticed, some of these proofs are quite lengthy and technical. Walking the reader through proof details, enables a better understanding of techniques for programming translucent devices, beside keeping the paper totally self-contained.

2. Preliminaries

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 $x \notin S$. We use $\subseteq$ for inclusions, and $\subset$ for proper inclusion.

We denote by Σ^* the set of all words on the finite alphabet Σ, including the empty word λ, and let $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. For any $w \in \Sigma^*$, 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. A language on Σ is any subset $L \subseteq \Sigma^*$. The complement of L is the language $\overline{L} = \Sigma^* \setminus L$, its reversal is $L^R = \{ w^R \mid w \in L \}$. Two language families $\mathscr{L}_1$ and $\mathscr{L}_2$ are said to be incomparable whenever $\mathscr{L}_1$ is not a subset of $\mathscr{L}_2$ and vice versa.

2.1. Returning devices

Roughly speaking, pushdown automata with translucent input letters are extensions of classical pushdown automata that do not have to read their inputs strictly sequentially from left to right. In fact, depending on their current state, some of the input letters may be translucent, and hence they become temporarily invisible. Accordingly, the device either performs a λ-transition without reading an input symbol or reads and processes — by deleting, if not the endmarker — the first visible input symbol from the left. Let us formally define pushdown automata with translucent input letters. Here, we are particularly interested in deterministic computations.

A *deterministic pushdown automaton with translucent input letters* (DPDAwtl, [9]) formally writes as $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \bot, \tau, \delta \rangle$, where Q is the finite set of states, Σ is the finite set of input symbols, with $\Sigma \cap Q = \emptyset$, Γ is the finite set of pushdown symbols, $q_0 \in Q$ is the initial state, $\triangleleft \notin \Sigma$ is the endmarker, $\bot \notin \Gamma$ is the bottom-of-pushdown symbol, and

- $\delta : Q \times (\Sigma \cup \{\lambda, \triangleleft\}) \times \Gamma_\bot \to (Q \times \Gamma_\bot^*) \cup \{\texttt{accept}\}$ is the partial transition function, enjoying the following properties for any $q \in Q$ and $z \in \Gamma_\bot$:
  - there must never be a choice between using an input symbol and using λ input. So, we require that if $\delta(q, \lambda, z)$ is defined, then $\delta(q, a, z)$ is undefined for all a in $\Sigma_\triangleleft$,
  - to simplify matters, at any time during computations, the bottom-of-pushdown symbol appears exactly once at the bottom of the pushdown store, meaning that it cannot be deleted nor replicated. So, we require that if $\delta(q, a, z) = (p, \beta)$, then either $z \neq \bot$ and $\beta \in \Gamma^*$, or $z = \bot$ and $\beta = \beta' \bot$ with $\beta' \in \Gamma^*$.
- $\tau : Q \to 2^\Sigma$ is the *translucency* mapping, bearing the following meaning: for any state $q \in Q$, the letters from the set $\tau(q)$ are *translucent (invisible) for* q. So, whenever in q, the automaton M does not see (or equivalently, can see through) these letters.

A configuration of M is a pair $(qv\triangleleft, \gamma)$ or $\texttt{accept}$, where

- $q \in Q$ is the current state,
- $v \in \Sigma^*$ is the part of the input left to be processed. Recall that, as above quoted, at each step which is neither a λ-transition nor a move on the endmarker, M deletes/processes the first visible input symbol from the left; so, from this viewpoint, v is the still undeleted/unprocessed part of the input string,
- $\gamma \in \Gamma^* \bot$ denotes the current pushdown content, the leftmost symbol being the top of the pushdown store.

Clearly, the initial configuration of M on any given input word $w \in \Sigma^*$ has the form $(q_0 w \triangleleft, \bot)$. Along its computation, M runs through a sequence of configurations.

Being in some configuration $(qv\triangleleft, z\gamma)$, with $q \in Q$, $v \in \Sigma^*$, $z \in \Gamma$, and $\gamma \in \Gamma^* \bot$, a step of M plays out as follows. First, M checks whether $\delta(q, \lambda, z)$ is defined: if this is the case with $\delta(q, \lambda, z) = (p, \beta)$, then the successor configuration is $(pv\triangleleft, \beta\gamma)$. Otherwise, $\delta(q, \lambda, z)$ being undefined, we have that M spots the next input symbol to be processed by taking the first letter from left that is visible in the state q. Precisely, if $v = xay$ with $x \in \tau(q)^*$ and $a \notin \tau(q)$, then M takes a. Now, M halts and rejects if $\delta(q, a, z)$ is undefined, otherwise it computes $\delta(q, a, z) = (p, \beta)$ and the successor configuration becomes $(pxy\triangleleft, \beta\gamma)$.

Being in $(qv\triangleleft, z\gamma)$ with $v \in \tau(q)^*$, then M sees the endmarker $\triangleleft$, and it halts and accepts if and only if $\delta(q, \triangleleft, z) = \texttt{accept}$, otherwise it rejects.

More formally, one step from a configuration to its successor configuration is denoted by $\vdash$, which is specified as follows. Let $p, q \in Q$, $a \in \Sigma$, $v, x, y \in \Sigma^*$, $z \in \Gamma_\bot$ and $\beta\gamma, z\gamma \in \Gamma^* \bot$. We set:

- $(qv\triangleleft, z\gamma) \vdash (pv\triangleleft, \beta\gamma)$, if $(p, \beta) = \delta(q, \lambda, z)$,
- $(qxay\triangleleft, z\gamma) \vdash (pxy\triangleleft, \beta\gamma)$, if $x \in \tau(q)^*$, $a \notin \tau(q)$, and $(p, \beta) = \delta(q, a, z)$,
- $(qv\triangleleft, z\gamma) \vdash (pv\triangleleft, \beta\gamma)$, if $v \in \tau(q)^*$ and $(p, \beta) = \delta(q, \triangleleft, z)$,
- $(qv\triangleleft, z\gamma) \vdash \texttt{accept}$, if $v \in \tau(q)^*$ and $\texttt{accept} = \delta(q, \triangleleft, z)$.

We let $\vdash^*$ (resp., $\vdash^+$) denote the reflexive and transitive (resp., transitive) closure of $\vdash$. The language accepted by the DPDAwtl M is the set $L(M)$ of those words $w \in \Sigma^*$ for which the computation of M, beginning in the initial configuration with w as input, eventually halts accepting, namely:

$$L(M) = \{ w \in \Sigma^* \mid (q_0 w \triangleleft, \bot) \vdash^+ \texttt{accept} \}.$$

To clarify the behavior of DPDAwtl, we propose the following example.

Example 1. Let the language $L_{dc} \subseteq \{a,b,c,d\}^*$ be defined as

$$L_{dc} = \{\, uvdc^n \mid u \in \{a,b\}^*,\ v \in \{a,c\}^*,\ |uv|_a = |v|_c,\ n = |u|_b \,\}.$$

This language is presented in [16], where its acceptance is shown, by certain cooperating distributed systems of restarting automata equipped with additional pushdown store. As a matter of fact, such systems can be seen as forerunners of the DPDAwtl model we are proposing in the present paper.

We are now going to design a DPDAwtl A for L_{dc}. We will use this result later, in order to show that the family of the languages accepted by DPDAwtl is not closed under inverse homomorphism. Intuitively, the behavior of A on an input string $uvdc^n \in L_{dc}$ develops across the following phases:

- PHASE 1: b symbols are processed, and for each b a symbol B is pushed; so, at the end of this phase, the input string reduces to xdc^n, for $x \in \{a,c\}^*$ and $|x| = |u|_a + |v|$, while the pushdown content is $B^{|u|_b}\bot$.
- PHASE 2: a symbols and c symbols sitting before the symbol d are processed, by alternating reading a c and then reading an a, until the prefix x is completely processed; at the end of this phase, the input string has the form dc^n, while the pushdown content remains untouched as $B^{|u|_b}\bot$.
- PHASE 3: once the d symbol has been processed, every c symbol left in the input is matched against a B symbol in the pushdown, in order to check that the number of c equals the number of b previously occurring in u.

By suitably toggling translucency and fixing undefined moves, the input strings not in L_{dc} can be rejected by A.

Let us formally implement the above three phases on the claimed DPDAwtl $A = \langle Q, \{a,b,c,d\}, \{B\}, q_0, \lhd, \bot, \tau, \delta\rangle$. We let $Q = \{q_0, q_1, q_2, q_3\}$, the translucency $\tau(q_0) = \{a\} = \tau(q_2)$, $\tau(q_1) = \{c\}$, $\tau(q_3) = \emptyset$, and the transition function δ, for $z \in \{B, \bot\}$, as being:

PHASE 1 and PHASE 2	PHASE 3
$\delta(q_0, b, z) = (q_0, Bz)$	$\delta(q_0, d, z) = (q_3, z)$
$\delta(q_0, c, z) = (q_1, z)$	$\delta(q_2, d, z) = (q_3, z)$
$\delta(q_1, a, z) = (q_2, z)$	$\delta(q_3, c, B) = (q_3, \lambda)$
$\delta(q_2, c, z) = (q_1, z)$	$\delta(q_3, \lhd, \bot) =$ `accept`

So, we can conclude that the DPDAwtl A accepts L_{dc}. ■

2.2. Non-returning devices

Deterministic pushdown automata with translucent input letters have been defined in [9] also in 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 rightward. Whenever the endmarker is reached and the transition on the endmarker yields a new state, the computation is continued in this new state, with the input head placed at the left end of the remaining input.

Such *deterministic pushdown automata with translucent input letters working in the non-returning mode* (nrDPDAwtl) generalize non-returning finite state automata with translucent input letters [13]. In their formal definition as an 8-tuple, the components are defined as for DPDAwtl in the previous subsection, the difference showing up in the dynamics.

Thus, given the nrDPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta\rangle$, a configuration of M is a pair $(xqw\lhd, \gamma)$ or `accept` where, as before, $q \in Q$ and $\gamma \in \Gamma^*\bot$ are the current state and pushdown content, respectively, but now

$xw \in \Sigma^*$ is the input still left undeleted on the tape, with w being to the right and x to the left of the input head.

The successor configuration yielded by $\vdash$ is now specified as follows. Let $p, q \in Q$, $a \in \Sigma$, $x, u, v, w \in \Sigma^*$, $z \in \Gamma_\bot$, and $\beta\gamma, z\gamma \in \Gamma^*\bot$. Then:

- $(xqw\lhd, z\gamma) \vdash (xpw\lhd, \beta\gamma)$, if $(p, \beta) = \delta(q, \lambda, z)$,
- $(xquav\lhd, z\gamma) \vdash (xupv\lhd, \beta\gamma)$, if $u \in \tau(q)^*$, $a \notin \tau(q)$, and $(p, \beta) = \delta(q, a, z)$,

`return:` $(xqw\lhd, z\gamma) \vdash (pxw\lhd, \beta\gamma)$, if $w \in \tau(q)^*$ and $(p, \beta) = \delta(q, \lhd, z)$,

- $(xqw\lhd, z\gamma) \vdash$ `accept`, if $w \in \tau(q)^*$ and `accept` $= \delta(q, \lhd, z)$.

The accepted language $L(M)$ can be straighforwardly defined.

Sometimes, we will be saying that an nrDPDAwtl 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 (move of type (`return`) above).

Let us give an intuition on how a nrDPDAwtl works by the following example.

Example 2. Consider the non-context-free language

$$L = \{\, x\#y\#z \mid x, y, z \in \{a,c\}^*,\ |x|_a = |y|_a = |z|_c \geq 1 \,\}.$$

We will use this language in Proposition 7, to show the non-closure under inverse homomorphism of the family of the languges accepted by nrDPDAwtl. A detailed construction of an nrDPDAwtl M for L is presented in our contribution [9], and it is here reported for the sake of completeness.

For the construction of the nrDPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \bot, \tau, \delta\rangle$, we let $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, c, \#\}$, and $\Gamma = \{a\}$. We explain the translucency function τ and the transition function δ while describing the computation of M. At the beginning of the computation, M starts reading the first block of the input string, i.e. x, while storing in the pushdown as many symbols as the number of a's in x. In this phase, M reads (and erases) all characters in x, until the first # is reached. By letting $t \in \Gamma_\bot$, we set

$$\begin{aligned}\tau(q_0) = \emptyset, \quad & \delta(q_0, a, t) = (q_0, at),\\ & \delta(q_0, c, t) = (q_0, t),\\ & \delta(q_0, \#, t) = (q_1, t).\end{aligned}$$

Then, M continues the computation by reading the second and the third block of the input string, while deleting only c's and a's respectively. Once reading the endmarker, M moves the head back at the beginning of the tape. So, we set

$$\begin{aligned}\tau(q_1) = \{a\}, \quad & \delta(q_1, c, t) = (q_1, t),\\ & \delta(q_1, \#, t) = (q_2, t),\\ \tau(q_2) = \{c\}, \quad & \delta(q_2, a, t) = (q_2, t),\\ & \delta(q_2, \triangleleft, t) = (q_3, t).\end{aligned}$$

In an accepting computation, the configuration of M reached at this point should have the form $(q_3\ a^n c^n \triangleleft, a^n \bot)$. It is now easy to check whether the number of a's and c's coincides with the number of symbols stored in the pushdown: for each a in the first block, M deletes one c in the second block, and pops one symbol from the pushdown. Thus, M accepts whenever it reaches the endmarker with empty pushdown and no symbol left on the tape. We set

$$\begin{aligned}\tau(q_3) = \emptyset, \quad & \delta(q_3, a, t) = (q_4, t),\\ & \delta(q_3, \triangleleft, \bot) = \texttt{accept},\\ \tau(q_4) = \{a\}, \quad & \delta(q_4, c, t) = (q_5, t),\\ \tau(q_5) = \{c\}, \quad & \delta(q_5, \triangleleft, a) = (q_3, \lambda).\end{aligned}$$

Therefore, the whole computation of M can be considered as being divided into two parts:

1. The first part consists of the first sweep, where $|x|_a$ symbols are stored in the pushdown and not relevant characters in the rest of the string are deleted. On a word $w = x\#y\#z \in L$, this first phase simply writes as

$$(q_0\ x\#y\#z\triangleleft, \bot) \vdash^* (q_3\ a^i\ c^j \triangleleft, a^h \bot),$$

   where $i = |y|_a$, $j = |z|_c$ and $h = |x|_a$.
2. In the second part, the head of M moves back and forth on the tape, while deleting symbols in the string and from the pushdown. Note that the block structure of the string $a^n c^n \triangleleft$ at the beginning of this phase is guaranteed by the earlier computation part. Also, the configuration $(q_3\ a^n c^n \triangleleft, a^n \bot)$ is repeated at the beginning of each sweep in this second part.

From the beginning of the second part to the end of the computation, M repeats the following cycle:

$$\begin{aligned}(q_3\ a^i c^j \triangleleft, a^h \bot) &\vdash (q_4\ a^{i-1} c^j \triangleleft, a^h \bot)\\ &\vdash (a^{i-1}\ q_5\ c^{j-1} \triangleleft, a^h \bot)\\ &\vdash (q_3\ a^{i-1} c^{j-1} \triangleleft, a^{h-1} \bot)\\ &\vdash \cdots\end{aligned}$$

It is easy to see the input string is accepted only when $|x|_a = |y|_a = |z|_c$ and the last reached configuration is $(q_3 \triangleleft, \bot)$. In any other case, M rejects. ■

Throughout the rest of the paper, we will be denoting by $\mathscr{L}(\mathsf{X})$ the family of all languages accepted by some device X.

3. Basic results on DPDA with translucent input letters

Let us prepare some preliminary results on the form and computational power of DPDAwtl and nrDPDAwtl, that will turn out to be useful in the following sections, where we are going to investigate some closure properties for the language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl).

3.1. Normal forms

We show a technical lemma stating that, without loss of generality, DPDAwtl and nrDPDAwtl can always be assumed to either pop on λ-transitions only and completely consume accepted inputs. Such assumptions are particularly useful to simplify our proofs.

Lemma 1. *For a given DPDAwtl (resp., nrDPDAwtl), an equivalent DPDAwtl (resp., nrDPDAwtl) can effectively be constructed, in which both the following features hold:*

i pop-moves are performed by λ-transitions only,
ii acceptance only takes place after the whole *input has been processed.*

Proof. Let $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \bot, \tau, \delta\rangle$ be a DPDAwtl or nrDPDAwtl. We are going to design an equivalent DPDAwtl or nrDPDAwtl, respectively, M' featuring both *(i)* and *(ii)*.

1 Let $P = \{p \in Q \mid \delta(q, \sigma, z) = (p, \lambda),\ \sigma \neq \lambda\}$ be the set of states reached by M after pop-moves on input symbols. We define a new set P' as being the primed version of P, and construct an equivalent DPDAwtl or nrDPDAwtl M' having state set $Q' = Q \cup P' \cup \{s\}$, with $s \notin Q \cup P'$. The transition function δ' coincides with δ on those states from Q that are not used in M to pop on input symbols, plus we add pairs of transitions

$$\{\ \delta'(q, \sigma, z) = (p', z),\ \delta'(p', \lambda, z) = (p, \lambda) \mid \delta(q, \sigma, z) = (p, \lambda)\ \}$$

simulating each pop-move of M by two consecutive steps: in particular, the second step $\delta'(p', \lambda, z) = (p, \lambda)$ is the λ-transition actually popping the pushdown. The translucency function τ' coincides with τ on Q, while it can be set to $\emptyset$ for any other state. It is easy to check that M' exhibits feature *(i)*.

2 Let us keep updating M' obtained at the previous point, so that even feature *(ii)* can be accomplished. So, for any instruction in M of the form $\delta(q, \triangleleft, z) = \texttt{accept}$, we add transitions

$$\underbrace{\{\delta'(q, \triangleleft, z) = (s, z)\}}_{(a)} \cup \underbrace{(\bigcup_{\sigma\in\Sigma}\{\delta'(s, \sigma, z) = (s, z)\})}_{(b)} \cup \underbrace{\{\delta'(s, \triangleleft, z) = \texttt{accept}\}}_{(c)}. \tag{1}$$

Recall from the previous point of this proof, that $\tau'(s) = \emptyset$. The added transitions (1), occurring upon acceptance on M, basically:
(a) lead M' into the new state s,
(b) consume the input one symbol per each step from the leftmost input symbol, and
(c) accept from s on the endmarker in a final step.
Therefore, feature *(ii)* is achieved as well.

□

3.2. Some recognition capabilities

Next, we test DPDAwtl and nrDPDAwtl recognition capabilities on particular language families, that will be used later as witness languages.

Lemma 2. *Let Σ be an alphabet, $c \in \Sigma$ a symbol, and $\Sigma^* \supseteq L \in \mathscr{L}$(DPDAwtl) a language. Both languages $L \cap \{c\}(\Sigma \setminus \{c\})^*$ and $L \cap \{c\}(\Sigma \setminus \{c\})^+$ belong to $\mathscr{L}$(DPDAwtl).*

Proof. Let us focus on the language $L \cap \{c\}(\Sigma \setminus \{c\})^* = L'$, and assume the DPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \bot, \tau, \delta\rangle$ for the language L. The DPDAwtl M' for L' we are going to build out of M depends on whether or not $c \in \tau(q_0)$:

- $c \notin \tau(q_0)$: We define $M' = \langle Q', \Sigma, \Gamma, q_0', \triangleleft, \bot, \tau', \delta'\rangle$, with state set $Q' = \{q_0'\} \cup Q$, initial state q_0', and translucency function defined as $\tau'(q_0') = \emptyset$, and $\tau'(q) = \tau(q) \setminus \{c\}$ for any $q \in Q$ (i.e., c is always visible).
  For the transition function, we let $\delta'(q_0', c, \bot) = \delta(q_0, c, \bot)$, while any other transition on q_0' is left undefined. In addition, for $q \in Q$ and $\gamma \in \Gamma_\bot$, we let
  $$\delta'(q, \sigma, \gamma) = \begin{cases} \text{not defined} & \text{if } \sigma = c \\ \delta(q, \sigma, \gamma) & \text{if } \sigma \in (\Sigma \setminus \{c\}) \cup \{\lambda, \triangleleft\}. \end{cases}$$
  It is not hard to see that M' simulates exactly the computation of M on input strings of the form $cx \in \{c\}(\Sigma \setminus \{c\})^*$, thus accepting cx if and only if $cx \in L$. On the other hand, input strings in $(\Sigma \setminus \{c\})\Sigma^* \cup \{c\}\Sigma^*\{c\}\Sigma^*$ are always rejected by M'.
- $c \in \tau(q_0)$: By Lemma 1, we can assume without loss of generality that M always consumes the whole input on accepting computations. Therefore, for accepted input strings beginning with the symbol c, there must exists a state $s \neq q_0$ with $c \notin \tau(s)$, upon which c is consumed.
  Our DPDAwtl M' is designed to consume the symbol c at the beginning of input strings (if any, otherwise it rejects), while it simulates the move of M consuming c by a λ-transition. To this aim, we use a set $\tilde{Q}$ of new states witnessing that this λ-transition has not yet been made. Upon performing this step, M' switches from states in $\tilde{Q}$ to states in Q, and from this point on any further processing of the symbol c leads M' to reject.
  So, we let $M' = \langle Q', \Sigma, \Gamma, q_0', \triangleleft, \bot, \tau', \delta'\rangle$, with $Q' = \{q_0', q_0''\} \cup \tilde{Q} \cup Q$ and $\tilde{Q} = \{\tilde{q} \mid q \in Q\}$, the initial state is q_0', and the translucency function is
  $$\tau'(q) = \begin{cases} \emptyset & \text{if } q \in \{q_0', q_0''\} \\ \tau(q) \setminus \{c\} & \text{if } q \in Q \end{cases} \text{ and } \tau'(\tilde{q}) = \tau(q), \text{ for } \tilde{q} \in \tilde{Q}.$$

Let us now specify the transition function δ', which is set to be undefined whenever not specified. For $\sigma \in (\Sigma \setminus \{c\}) \cup \{\lambda, \triangleleft\}$:

$$\begin{aligned} \delta'(q_0', c, \bot) &= (q_0'', \bot), \\ \delta'(q_0'', \sigma, \bot) &= (\tilde{q}, \gamma) \qquad \text{if } \delta(q_0, \sigma, \bot) = (q, \gamma). \end{aligned}$$

The definition of δ' on states in $\tilde{Q}$ depends on the translucency of c. For any $\tilde{q} \in \tilde{Q}$ with $c \in \tau'(\tilde{q}) = \tau(q)$ and any $\sigma \in (\Sigma \setminus \{c\}) \cup \{\lambda, \triangleleft\}$, we set

$$\delta'(\tilde{q}, \sigma, z) = (\tilde{p}, \gamma) \text{ if } \delta(q, \sigma, z) = (p, \gamma).$$

On the other hand, for any $\tilde{q} \in \tilde{Q}$ with $c \notin \tau'(\tilde{q}) = \tau(q)$, only the following λ-transition is defined:

$$\delta'(\tilde{q}, \lambda, z) = (p, \gamma) \text{ if } \delta(q, c, z) = (p, \gamma).$$

For the remaining states $q \in Q$ and any $\sigma \in (\Sigma \setminus \{c\}) \cup \{\lambda, \triangleleft\}$, we set

$$\delta'(q, \sigma, z) = \delta(q, \sigma, z).$$

To accept the language $L \cap \{c\}(\Sigma \setminus \{c\})^+$, we construct a DPDAwtl M'' which essentially behaves as M', plus M'' checks whether at least another symbol different from c is processed. It is not hard to see that this can be achieved by at most doubling the number of states of M'.

□

Lemma 3. *Let M be an nrDPDAwtl accepting the language $\{c\}L$, where $L \subseteq \Sigma^*$ and $c \notin \Sigma$. Then, an nrDPDAwtl M' accepting the language L can effectively be constructed.*

Proof. Let $M = \langle Q, \Sigma \cup \{c\}, \Gamma, q_0, \triangleleft, \bot, \tau, \delta \rangle$ be an nrDPDAwtl accepting $\{c\}L$. Let Q' be a primed copy of Q. We define $M' = \langle Q \cup Q', \Sigma, \Gamma, p_0, \triangleleft, \bot, \tau', \delta' \rangle$, with $\tau'(q) = \tau(q) \setminus \{c\}$ for all $q \in Q$ and $\delta'(q, x, z) = \delta(q, x, z)$ for all $q \in Q$, $x \in \Sigma \cup \{\lambda, \triangleleft\}$, and $z \in \Gamma_\bot$. These rules simulate M excluding the behavior on the symbol c at the beginning of the input.

To adapt this step for M', we check whether or not $c \in \tau(q_0)$. If c is not translucent, we set the initial state as $p_0 = q_0'$ and we define the transition $\delta'(q_0', \lambda, \bot) = \delta(q_0, c, \bot)$. Otherwise, we set the initial state as $p_0 = q_0$, and consider all states p which are reached by M after a return, i.e., those p satisfying $\delta(q, \triangleleft, z) = (p, \beta)$ with $p, q \in Q$, $z \in \Gamma_\bot$ and $\beta \in \Gamma_\bot^*$, for which $c \notin \tau(p)$. The "returning" transition is now set to be $\delta'(q, \triangleleft, z) = (p', \beta)$, plus we set $\delta'(p', \lambda, z') = \delta(p, c, z')$ for every defined transition $\delta(p, c, z')$ with $z' \in \Gamma_\bot$.

Hence, the behavior of M on the input symbol c is simulated in M' by a λ-transition on a primed state. Altogether, M' is a nrDPDAwtl whose accepted language coincides with L. □

For the next result, we consider the two languages

$$L_1 = \{\, b^n \# b^n \mid n \geq 0 \,\} \quad \text{and} \quad L_2 = \{\, b^n \# b^{2n} \mid n \geq 0 \,\}.$$

It has been shown in [9] that the language $L_1 \cup L_2$ is not in $\mathscr{L}$(nrDPDAwtl).

Lemma 4. *The language $\{c\}(L_1 \cup L_2)$ does not belong to $\mathscr{L}$(nrDPDAwtl).*

Proof. By *absurdum*, assume the language $\{c\}(L_1 \cup L_2)$ is accepted by some nrDPDAwtl M. Since $L_1 \cup L_2$ is defined over the alphabet $\{b, \#\}$, by applying Lemma 3 to M we get that $L_1 \cup L_2$ is accepted by some nrDPDAwtl M'. This contradicts the fact that $L_1 \cup L_2$ cannot be accepted by any nrDPDAwtl, as above recalled from [9]. □

4. DPDA vs. finite automata in the translucent setting

For the recognition capability of DPDAwtl, nrDPDAwtl, and that of classical DPDA, the following proper hierarchy is shown in [9], where DCFL (resp., DCSL) denotes the family of deterministic context-free (resp., -sensitive) languages:

$$\text{DCFL} = \mathscr{L}(\text{DPDA}) \subset \mathscr{L}(\text{DPDAwtl}) \subset \mathscr{L}(\text{nrDPDAwtl}) \subset \text{DCSL}.$$

The recognition power of deterministic and nondeterministic finite automata with translucent input letters working in the returning and non-returning mode (DFAwtl, NFAwtl, nrDFAwtl, nrNFAwtl) are compared in [13]. In particular, all families of languages accepted by these devices properly include the family of regular languages. Moreover, the following proper inclusions are also shown:

$$\begin{aligned} &\mathscr{L}(\text{nrDFAwtl}) \subset \mathscr{L}(\text{nrNFAwtl}), && \mathscr{L}(\text{NFAwtl}) \subset \mathscr{L}(\text{nrNFAwtl}), \\ &\mathscr{L}(\text{DFAwtl}) \subset \mathscr{L}(\text{NFAwtl}), && \mathscr{L}(\text{DFAwtl}) \subset \mathscr{L}(\text{nrDFAwtl}). \end{aligned}$$

So, the resources *nondeterminism* and *non-returning mode* have an impact on the accepting capabilities. However, this impact is incomparable if the resources are added to DFAwtl. This raises the natural question on how the resource *pushdown store* compares to *nondeterminism* and *non-returning mode*. In the following, we are going to explore these relationships, enabling us to draw a complete picture depicted in Fig. 1, page 18.

Some of the comparisons among acceptance capabilities can be derived almost immediately from structural reasons and known results. For instance, comparing the acceptance with or without the resource *pushdown store*, yields the incomparability of $\mathscr{L}$(DPDA)

with all of the language families $\mathscr{L}$(DFAwtl), $\mathscr{L}$(NFAwtl), $\mathscr{L}$(nrDFAwtl), and $\mathscr{L}$(nrNFAwtl). In fact, by letting the homomorphism $\varphi : \{ a, b \}^* \to \{ c, d \}^*$ with $\varphi(a) = c$ and $\varphi(b) = d$, the deterministic context-free language

$$\{ w \varphi(w^R) \mid w \in \{a, b\}^* \}$$

is not accepted by any nrNFAwtl [13]. Also, it is not hard to see that the language $\{ b^n \# b^n \mid n \geq 0 \}$ is not accepted by any nrNFAwtl. On the other hand, the family $\mathscr{L}$(DFAwtl) includes non-context-free languages [14]. Due to the inclusions above recalled, the incomparabilities follow. The same witness languages show that the trivial inclusions $\mathscr{L}$(DFAwtl) $\subset$ $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDFAwtl) $\subset$ $\mathscr{L}$(nrDPDAwtl) are proper.

So, the comparisons between $\mathscr{L}$(nrDFAwtl) and $\mathscr{L}$(DPDAwtl), that is, between the resources *non-returning mode* and *pushdown store*, as well as between $\mathscr{L}$(nrNFAwtl) and $\mathscr{L}$(nrDPDAwtl), that is, between the resources *non-returning mode and nondeterminism* and *non-returning mode and pushdown store*, are left to be considered, and will be dealt with in the following Proposition 1 and Proposition 2.

Proposition 1. *The families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDFAwtl) are incomparable.*

Proof. As shown in [13], an nrDFAwtl M exists, which accepts the context-sensitive language $L = \{ a^n b^n c^n \mid n \geq 0 \}$. Very roughly speaking, the behavior of M is as follows: on any given input string $x \in \{ a, b, c \}^*$, M deletes in succession first a symbol a, then a symbol b, then a symbol c. By iterating this process, we have that M completely deletes x whenever $|x|_a = |x|_b = |x|_c$. In addition, the non-returning mode is easily seen to also ensure that $x \in a^* b^* c^*$.

On the other hand, it is also easy to see that L does not contain any letter-equivalent context-free sub-language. This fact implies that L cannot be accepted by any DPDAwtl, due to Proposition 5 in [9]. So, the incomparability follows from the incomparability between $\mathscr{L}$(DPDA) and $\mathscr{L}$(nrDFAwtl). □

In preparation for the last incomparability result, we define the language

$$L_{tr} = \{ w \in \#^+ (a\#^+)^* (b\#^+)^* (c\#^+)^* \mid |w|_a = |w|_b = |w|_c \}.$$

The words of L_{tr} are essentially words of the form $a^n b^n c^n$ where, at the beginning and after each symbol from $\{a, b, c\}$, arbitrarily many but at least one symbol # occur. For the language L_{tr}, the following holds:

Theorem 1. *The language L_{tr} cannot be accepted by any nrDPDAwtl.*

Proof. We feel it worth emphasizing the following preliminary consideration:

> Let M' be an arbitrary deterministic pushdown automaton, starting in any of its states with an arbitrary pushdown content. On a sufficiently long unary input word, M' will eventually run into a state loop. Clearly, the length of this unary word depends on the inner architecture of M' and on the height of the pushdown. In an ordinary computation, this pushdown height can be bounded above, depending on the length of the input prefix processed so far. The same applies to nrDPDAwtl, unless the unary symbol becomes translucent, so that the nrDPDAwtl jumps over the unary input factor.

With this being said, assume by contradiction that L_{tr} is accepted by some nrDPDAwtl $M = \langle Q, \{a, b, c, \#\}, \Gamma, q_0, \triangleleft, \bot, \tau, \delta \rangle$. By Lemma 1, we may safely assume that pop-moves in M occur on λ-transitions only, and that M accepts only after processing the whole input. Let $w \in L_{tr}$ be a word of the form

$$w = \#^{m_0} a \#^{m_1} \cdots a \#^{m_n} b \#^{m_{n+1}} b \#^{m_{n+2}} \cdots b \#^{m_{2n}} c \#^{m_{2n+1}} c \#^{m_{2n+2}} \cdots c \#^{m_{3n}},$$

where m_i's, with $0 \leq i \leq 3n$, are large enough (see the initial consideration) to trigger a state loop in M while processing $\#^{m_i}$, provided that M does not jump on #. Since M is non-returning, it may process the input in several sweeps from left to right. Let us focus on accepting computations of M on w, and distinguish between two cases considered below.

CASE 1: Assume M runs into a state loop while processing some #-block. This means that M does not jump over the block, so that for none of the states in the loop the symbol # is translucent. Furthermore, assume that the loop involves a state q for which the symbol $x \in \{a, b, c\}$ following the #-block is translucent. We may safely assume that such an x is not the last x in w. Next, we can suitably "adjust" the length of the #-block so that its processing ends in the state q. This implies that the symbol x remains unprocessed, but the following #-block is processed by the same state loop. More precisely, M does not realize the existence of a jumped-over symbol. So, we can adjust the length of all subsequent #-blocks in the same way, as long as such #-blocks are followed by the symbol x. Let an input string be accepted in a computation beginning as

$$\begin{aligned}(up\#^{k_0} x \#^{k_1} x \#^{k_2} v \triangleleft, \gamma_1) &\vdash^* (uqx\#^{k_1} x \#^{k_2} v \triangleleft, \gamma_2) \\ &\vdash^* (uxqx\#^{k_2} v \triangleleft, \gamma_3) \\ &\vdash (uxxq'\#^{k_2 - 1} v \triangleleft, \gamma_4).\end{aligned}$$

Therefore, also the computation

$$(up\#^{k_0 + k_1} xx \#^{k_2} v \triangleleft, \gamma_1) \vdash^* (uqxx\#^{k_2} v \triangleleft, \gamma_3) \vdash (uxxq'\#^{k_2 - 1} v \triangleleft, \gamma_4)$$

ends accepting, but now the input string contains the factor xx, and thus it cannot belong to L_{tr}.

For the second part of CASE 1, we first assume that M runs into a state loop while processing a #-block followed by an a, but such loop does not involve a state for which the symbol a is translucent. Then the #-block and the following a are consumed entirely. We precede with the next #-blocks followed by a. If the state loop on these #-blocks contains a state for which the symbol a is translucent, then we have the same situation as in the first part of CASE 1, yielding a contradiction. Otherwise, all #-blocks together with corresponding a's at their ends are consumed entirely, and a symbol b appears in the computation.

Again, we repeat this argument, now for the #-blocks followed by b's, yielding either a contradiction as in the first part of CASE 1, or the complete annihilation of all #-blocks together with corresponding b's at their ends.

Finally, we use this argument once more, for the #-blocks followed by c's.

In the end, either we have a contradiction as in the first part of CASE 1, or all these #-blocks together with corresponding c's at their ends are entirely consumed. In this latter case, the input has been read almost entirely (actually, there might be some unprocessed input left on the tape), and it is then not hard to construct a deterministic pushdown automaton that accepts the same input. This implies the existence of a context-free sub-language of L_{tr} containing infinitely many words in which the symbols in $\{a, b, c\}$ do not show up with the same number of occurrences. This is a contradiction, and concludes CASE 1.

CASE 2: The contradictions in CASE 1 reveals the existence of at least one #-block in an accepted input upon which the processing of M hits a state, say q, for which the symbol # becomes translucent. Hence, M jumps over the still unprocessed part of this #-block.

First, we can exclude the trivial case where all input symbols become translucent for state q. Second, if the symbol from $\{a, b, c\}$ following the #-block is not translucent for q, then M jumps on it. Now, we can remove the part of the #-block which has been jumped over. This actually implies that no real jump is performed. By applying this reasoning to all the #-blocks on which M jumps over # symbols to the following symbol from $\{a, b, c\}$, we end up either in CASE 1, whence a contradiction, or in a #-block on which M jumps to a symbol in $\{b, c, \triangleleft\}$, which differs from the symbol following the #-block.

Let us assume this latter possibility and, in addition, that there exists a sweep in which M performs two of these jumps. So, let an input string be accepted in a computation containing a sub-computation of the form

$$\begin{aligned}
&(p\#^{k_0}x\#^{k_1}x\cdots\#^{k_{i-1}}x\#^{k_i}y\#^{\ell_1}y\#^{\ell_2}y\cdots\#^{\ell_j}zv,\gamma_1)\\
&\quad\vdash^* (q\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}x\#^{k_i}y\#^{\ell_1}y\#^{\ell_2}y\cdots\#^{\ell_j}zv,\gamma_2)\\
&\quad\vdash (\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}x\#^{k_i}p'y\#^{\ell_1}y\#^{\ell_2}y\cdots\#^{\ell_j}zv,\gamma_3)\\
&\quad\vdash^* (\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}x\#^{k_i}q'\#^{\ell_h-c_1}y\#^{\ell_{h+1}}y\cdots\#^{\ell_j}zv,\gamma_4)\\
&\quad\vdash (\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}x\#^{k_i}\#^{\ell_h-c_1}y\#^{\ell_{h+1}}y\cdots\#^{\ell_j}p''zv,\gamma_5),
\end{aligned}$$

where $(x, y, z) \in \{(a, b, c), (a, b, \triangleleft), (a, c, \triangleleft), (b, c, \triangleleft)\}$, c_0 and c_1 are positive integer constants, and p, q, p', q', p'' are states of M. Then, also the computation

$$\begin{aligned}
&(p\#^{k_0}x\#^{k_1}x\cdots\#^{k_{i-1}}xy\#^{\ell_1}y\#^{\ell_2}y\cdots y\#^{\ell_h+k_i}y\cdots\#^{\ell_j}zv,\gamma_1)\\
&\quad\vdash^* (q\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}xy\#^{\ell_1}y\#^{\ell_2}y\cdots y\#^{\ell_h+k_i}y\cdots\#^{\ell_j}zv,\gamma_2)\\
&\quad\vdash (\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}xp'y\#^{\ell_1}y\#^{\ell_2}y\cdots y\#^{\ell_h+k_i}y\cdots\#^{\ell_j}zv,\gamma_3)\\
&\quad\vdash^* (\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}xq'\#^{\ell_h-c_1+k_i}y\cdots\#^{\ell_j}zv,\gamma_4)\\
&\quad\vdash (\#^{k_0-c_0}x\#^{k_1}x\cdots\#^{k_{i-1}}x\#^{\ell_h-c_1+k_i}y\cdots\#^{\ell_j}p''zv,\gamma_5)
\end{aligned}$$

ends accepting, but the input string contains the factor xy, and thus it cannot belong to L_{tr}.

Finally, the case where in each sweep only one of these jumps over #-blocks is performed remains to be considered. To this aim, we construct a certain language accepted by a nondeterministic pushdown automaton (NPDA), and show that this language cannot be context-free, clearly a contradiction. From the cases considered so far, we get that infinitely many n exist, for which words of the form

$$w = \#^{m_0}a\#^{m_1}\cdots a\#^{m_n}b\#^{m_{n+1}}b\#^{m_{n+2}}\cdots b\#^{m_{2n}}c\#^{m_{2n+1}}c\#^{m_{2n+2}}\cdots c\#^{m_{3n}}$$

lead M to accept by computations where each sweep features only one of these jumps. Each of these words can be written as utv, where t is the factor over which M jumps in its first sweep. So, the second sweep will take place on the remaining word t. We use a new symbol and construct $w'_1 = uv\&t$. In its second sweep, M processes t which again can be written as $u_t t' v_t$, where t' is the factor over which M jumps in its second sweep. We construct $w'_2 = uv\&u_t v_t\&t'$. We iterate this construction for each sweep of M, finally obtaining some word w'.

We can build an NPDA M' out of M, which accepts exactly these words w'. To this aim, M' simulates M on the factor u, and guesses when u has been processed: let M be in state q at this moment. Then, M' remembers all letters that are translucent for q. Now, M' keeps on simulating M after its jump to v if and only if the first letter of v is not translucent for q. This part of the simulation ends when the symbol & appears in the input. On &, M' simulates the transition of M on the endmarker. Next, M' simulates the second sweep of M on $t = u_t t' v_t$ in the same way. In addition, it checks that t only contains letters that are translucent for the state q. By iterating this dynamics, all sweeps of M are simulated, and M' accepts if and only if the simulation ends accepting.

It remains to be shown that the language accepted by M' is not context-free. Recall from above that the jumps are over all #-blocks followed by a certain symbol. This, together with the fact that the words of L_{tr} are essentially words of the form $a^n b^n c^n$ with #-symbols shuffled in, suffices to get that $L(M')$ is not context-free by a simple application of the pumping lemma. This is the last contradiction in CASE 2, and concludes the proof. □

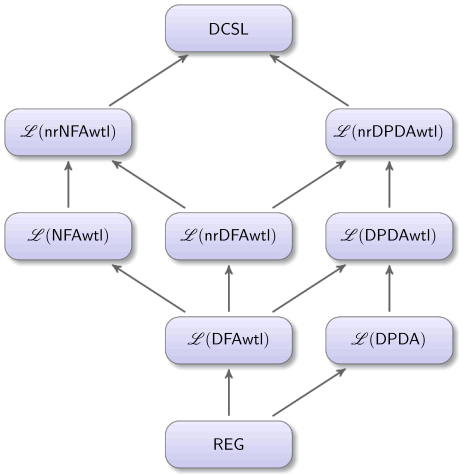


Fig. 1. Relationships between language families. An arrow between two families indicates a strict inclusion. Any pair of families not connected by a path is incomparable.

By this result, we can obtain:

Proposition 2. *The families $\mathscr{L}$(nrDPDAwtl) and $\mathscr{L}$(nrNFAwtl) are incomparable.*

Proof. As above quoted, the deterministic context-free languages

$$\{\, w\varphi(w^R) \mid w \in \{a,b\}^*,\ \varphi(a) = c,\ \varphi(b) = d \,\} \quad \text{and} \quad \{\, b^n \# b^n \mid n \geq 0 \,\}$$

cannot be accepted by any nrNFAwtl. For the converse, we consider the complement of the language L_{tr}. Such a complement consists of the union of the three languages

$$\{\, w \in \{a,b,c,\#\}^* \mid w \notin \#^+(a\#^+)^*(b\#^+)^*(c\#^+)^* \,\},$$
$$\{\, w \in \{a,b,c,\#\}^* \mid |w|_a \neq |w|_b \,\}, \text{ and}$$
$$\{\, w \in \{a,b,c,\#\}^* \mid |w|_a \neq |w|_c \,\}.$$

All these three languages are accepted by some NFAwtl. So, the complement of L_{tr} is accepted by some NFAwtl as well, since $\mathscr{L}$(NFAwtl) is closed under union [13]. However, the complement of L_{tr} cannot be accepted by any nrDPDAwtl, since L_{tr} does not belong to $\mathscr{L}$(nrDPDAwtl) by Theorem 1, and $\mathscr{L}$(nrDPDAwtl) is closed under complementation [9]. □

We conclude this section by displaying in Fig. 1 the complete situation of the relationships among language families defined by finite state and pushdown devices with translucent input letters, showing their positioning with respect to the well known language families REG (regular languages) and DCSL.

5. Closure properties

We are now going to investigate some closure properties for the language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl). In doing this, we solve an open problem on nrDFAwtl posed by Mráz and Otto in [13]. Let us begin by

Proposition 3. *The language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) are not closed under intersection with regular languages.*

Proof. We consider the following language, with ⧢ denoting the shuffle operator (see, e.g., [4]):

$$L_{tri} = \{\, w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c \,\} \sqcup\!\sqcup \{\, \#^n \mid n \geq 0 \,\}.$$

Since $L = \{\, w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c \,\}$ is accepted by a DFAwtl [15], we have that L can be accepted by a DPDAwtl, and thus by an nrDPDAwtl as well. From these acceptors, modified acceptors of the same type for L_{tri} can be easily constructed. Basically, these new acceptors simulate the old ones, but ignoring the symbols #.

Now, we define the regular language $R = \#^+(a\#^+)^*(b\#^+)^*(c\#^+)^*$ and consider the language $L_{tri} \cap R = L_{tr}$. By Theorem 1, L_{tr} cannot be accepted by any nrDPDAwtl, and so it cannot be accepted by any DPDAwtl as well. □

We emphasize that, as above observed, the language L_{tri} can be accepted by some DFAwtl, and so it is accepted by some nrDFAwtl as well. Therefore, the proof of Proposition 3 shows that the family $\mathscr{L}$(nrDFAwtl) is not closed under intersection with regular languages.

This solves a problem left open by Mráz and Otto in [13]:

Corollary 1. *The language family $\mathscr{L}$(nrDFAwtl) is not closed under intersection with regular languages.*

Let us now turn to non-closure under concatenation, Kleene star, and length-preserving homomorphism:

Proposition 4. *The language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) are neither closed under concatenation nor under Kleene star. They are not closed under length-preserving homomorphism, either.*

Proof. From Section 3, we recall the languages $L_1 = \{ b^n\#b^n \mid n \geq 0 \}$ and $L_2 = \{ b^n\#b^{2n} \mid n \geq 0 \}$, from which we construct the languages $\{c\}L_1 \cup L_2$ and $\{c\}^*$. These two languages are, respectively, a deterministic context-free and a regular language. Therefore, they can both be accepted by DPDAwtl.

Assume that $\mathscr{L}$(DPDAwtl) is closed under concatenation. Then, the language $(\{c\}^* \cdot (\{c\}L_1 \cup L_2)) \cap \{c\}\{b,\#\}^* = \{c\}(L_1 \cup L_2)$ belongs to $\mathscr{L}$(DPDAwtl) due to Lemma 2. However, Lemma 4 shows that $\{c\}(L_1 \cup L_2)$ cannot be accepted by any nrDPDAwtl, whence a contradiction.

For non-closure under Kleene star, consider the language $\{c\}^* \cup \{c\}L_1 \cup L_2$, which is accepted by a DPDAwtl. Assume $\mathscr{L}$(DPDAwtl) is closed under Kleene star. Then, the language $(\{c\}^* \cup \{c\}L_1 \cup L_2)^* \cap \{c\}\{b,\#\}^+ = \{c\}(L_1 \cup L_2)$ belongs to $\mathscr{L}$(DPDAwtl) due to Lemma 2, which again contradicts Lemma 4.

For non-closure under length-preserving homomorphism, consider the language $\{c\}L_1 \cup \{d\}L_2$, accepted by a DPDAwtl. By applying to this language the length-preserving homomorphism h such that $h(b) = b$, $h(\#) = \#$, $h(c) = c$, and $h(d) = c$, we obtain as homomorphic image the language $\{c\}(L_1 \cup L_2)$. Thus, assuming $\mathscr{L}$(DPDAwtl) is closed under length-preserving homomorphism would again contradict Lemma 4.

By a result in [9], every DPDAwtl can be transformed into an equivalent nrDPDAwtl. So, we obtain the non-closure results also for $\mathscr{L}$(nrDPDAwtl). □

For non-closure of $\mathscr{L}$(DPDAwtl) under inverse homomorphism, we need to recall the following property from [14]:

Proposition 5. *Let M be a DPDAwtl. Then a DPDA M' can effectively be constructed such that $L(M') \subseteq L(M)$ and $L(M')$ is letter-equivalent to $L(M)$.*

With this result in our hands, we are able to show

Proposition 6. *The language family $\mathscr{L}$(DPDAwtl) is not closed under inverse homomorphism.*

Proof. Let $L_{dc} = \{ uvdc^n \mid u \in \{a,b\}^*, v \in \{a,c\}^*, |uv|_a = |v|_c, n = |u|_b \}$ be the language considered in Example 1, where its membership in $\mathscr{L}$(DPDAwtl) is shown. From [16], we have that the inverse homomorphic image of L_{dc} by the homomorphism h such that $h(a) = ab$, $h(c) = c$, and $h(d) = d$ is the language $h^{-1}(L_{dc}) = \{ a^nc^ndc^n \mid n \geq 0\}$. It is not hard to see that this language does not contain any letter-equivalent context-free sub-language. Therefore, by Proposition 5, we can conclude that $h^{-1}(L_{dc}) \notin \mathscr{L}$(DPDAwtl). □

For non-closure of $\mathscr{L}$(nrDPDAwtl) under inverse homomorphism, we focus on the language $L_{tb} = \{ b^n\#b^n\#b^n \mid n \geq 1 \}$, and prove

Lemma 5. *The language L_{tb} is not accepted by any nrDPDAwtl.*

Proof. By contradiction, assume that L_{tb} can be accepted by some nrDPDAwtl $M = \langle Q, \{b,\#\}, \Gamma, q_0, \lhd, \bot, \tau, \delta\rangle$. By Lemma 1, we may safely assume that pop-moves in M occur on λ-transitions only, and that M accepts only after processing the whole input. We consider accepting computations on strings in L_{tb}, with n large enough. In particular, we focus on the part of these computations up to M seeing the endmarker for the first time. We distinguish among four cases considered below.

CASE 1 - *For infinitely many strings in L_{tb}, both symbols # remain in the input when M sees the endmarker for the first time:*

Basically, M has to start its computation by reading zero or a constant amount of b's from the prefix b^n of the input. If, in this phase, M reaches a state for which b's and #'s are translucent, then it sees the endmarker and the input head returns to the left of the input. Hence, beside possibly swithcing state and performing λ-transitions, nothing changes. If, in this phase, M reaches a state for which only b's are translucent, then it sees and processes the left # symbol. This contradicts the initial assumption defining the current case.

So, M must reach a state for which #'s are translucent when the left # appears in the input. But this implies that M continues to read and process b's without knowing that it has possibly already read the factor between the #'s. The same argument applies with respect to the right #. So, we conclude that in the current case M reaches the endmarker for the first time after having processed all b's from the input, and only the two symbols # remain.

Now, by omitting τ, ignoring the #'s, and finally simulating the behavior of M on the remaining input ##, we obtain a classical DPDA that accepts an infinite sub-language of L_{tb}. However, any infinite sub-language of L_{tb} is not context-free, thus contradicting Proposition 5. So, CASE 1 cannot apply.

CASE 2 - *For infinitely many strings in L_{tb}, the left but not the right symbol # remains in the input when M sees the endmarker for the first time:*

Essentially, in this case, the arguments of CASE 1 apply for the first two factors b^n and the # in between. So, if $(q_0b^n\#b^n\#b^n\lhd, \bot) \vdash^* (\#qb^{n-1}\#b^n\lhd, \gamma)$ is the initial phase of an accepting computation, then the same holds for an initial phase of the form $(q_0b^{n+1}\#b^{n-1}\#b^n\lhd, \bot) \vdash^* (\#qb^{n-1}\#b^n\lhd, \gamma)$. However, since $b^{n+1}\#b^{n-1}\#b^n \notin L_{tb}$, we obtain a contradiction and CASE 2 cannot apply.

CASE 3 - *For infinitely many strings in L_{tb}, the right but not the left symbol # remains in the input when M sees the endmarker for the first time:*

As before, M has to start the computation by reading a constant amount of b's or all b's from the prefix b^n of the input, and afterwards it sees and processes the left # symbol. Note that, by the initial assumption defining the current case, the # symbol must be processed.

So, let us first assume that M reads only a constant amount t_1 of b's from the prefix, and then it reaches a state for which the b's are translucent. Formally, $(q_0 b^n \# b^n \# b^n \lhd, \bot) \vdash^* (b^{n-t_1} q_1 \# b^n \# b^n \lhd, \gamma_1) \vdash (b^{n-t_1} q_2 b^n \# b^n \lhd, \gamma_2)$. Then, we have to distinguish two sub-cases.

In the first sub-case, the computation continues without reaching a state for which b's are translucent. By assumption of the current CASE 3, the right symbol # is not read and, thus, is translucent. This eventually yields a configuration $(b^{n-t_1} \# p_3 \lhd, \beta_3)$. Since M is deterministic, we would get the accepting computation $(q_0 b^n \# b^{n+1} \# b^{n-1} \lhd, \bot) \vdash^* (b^{n-t_1} q_2 b^{n+1} \# b^{n-1} \lhd, \gamma_2) \vdash^* (b^{n-t_1} \# p_3 \lhd, \beta_3)$. Therefore, the string $b^n \# b^{n+1} \# b^{n-1}$ would be accepted, a contradiction.

In the second sub-case, the computation continues by reading a constant amount t_2 of b's from the factor b^n between the two #'s, and then reaches a state for which b's and #'s are translucent. Therefore, we can formally write $(q_0 b^n \# b^n \# b^n, \bot) \vdash^* (b^{n-t_1} q_2 b^n \# b^n \lhd, \gamma_2) \vdash^* (b^{n-t_1} b^{n-t_2} \# b^n q_3 \lhd, \gamma_3)$. But then, the computation $(q_0 b^{n+1} \# b^{n-1} \# b^n \lhd, \bot) \vdash^* (b^{n+1-t_1} q_2 b^{n-1} \# b^n \lhd, \gamma_2) \vdash^* (b^{n+1-t_1} b^{n-1-t_2} \# b^n q_3 \lhd, \gamma_3)$ turns out to be admissible, and so $b^{n+1} \# b^{n-1} \# b^n$ would be accepted, a contradiction also in this sub-case.

Therefore, for both sub-cases we have contradictions and, thus, a contradiction to the assumption that M reads only a constant amount t_1 of b's from the prefix and then reaches a state for which b's are translucent.

So, to conclude CASE 3, we now assume that M reads all b's from the prefix b^n, and then it must read and process the left # symbol. Formally, we have $(q_0 b^n \# b^n \# b^n, \bot) \vdash^* (q_1 \# b^n \# b^n \lhd, \gamma_1) \vdash (q_2 b^n \# b^n \lhd, \gamma_2)$. Then, again, we have to distinguish the two sub-cases as before.

In the first sub-case, the computation continues without reaching a state for which b's are translucent. This eventually yields a configuration $(\# p_3 \lhd, \beta_3)$. As before, we obtain a contradiction by considering the admissible computation $(q_0 b^n \# b^{n+1} \# b^{n-1} \lhd, \bot) \vdash^* (q_2 b^{n+1} \# b^{n-1} \lhd, \gamma_2) \vdash^* (\# p_3 \lhd, \beta_3)$ which implies that the string $b^n \# b^{n+1} \# b^{n-1}$ is accepted.

In the second sub-case, the computation continues by reading an amount of $f(n)$ symbols b from the factor b^n between the two #'s, for some function $f(n) \leq n$, and then reaches a state for which b's and #'s are translucent. Formally, we write $(q_0 b^n \# b^n \# b^n \lhd, \bot) \vdash^* (q_2 b^n \# b^n \lhd, \gamma_2) \vdash^* (q_3 b^{n-f(n)} \# b^n \lhd, \gamma_3) \vdash^* (q_4 b^{n-f(n)} \# b^n \lhd, \gamma_4)$.

Here, one can effectively construct a classical DPDA M' from M as follows. On the prefix $b^n \# b^{f(n)}$, M' directly simulates M, and this drives M' into the configuration $(q_3 b^{n-f(n)} \# b^n \lhd, \gamma_3)$. As long as M' remembers in its states whether or not it has processed the right # symbol, it can determine by the translucency of the current simulated state of M whether or not M sees the endmarker. For example, for the state q_3, the b's and #'s are translucent, and thus M' simulates M on the endmarker yielding the correct configuration $(q_4 b^{n-f(n)} \# b^n \lhd, \gamma_4)$. So, what if M' simulates a state of M for which either b's or #'s are translucent but not both.

First, we observe that M cannot read a b to the right of the rightmost # symbol, without having processed the # symbol. Now, if #'s are translucent and the next input symbol is a b, then the transition of M can be simulated directly. The same holds whenever b's are translucent and the next input symbol is #. If #'s are translucent and the next input symbol is a #, then M' reads the # and stores it in its states. Note that this means that the remaining input for M and M' was $\# b^n$. If b's are translucent and the # is still not processed, then M must see the #. Assume that some $g_1(n)$ symbols b are invisible to M. Now, M' expects the # as next input symbol and can simulate the corresponding transition of M, by remembering that the # has been processed by M. This basically concludes the construction of M' except for one point. If M has skipped $g_1(n)$ symbols b, then these are visible again after a transition on the endmarker. So, they are placed at the end of the input for M', separated by a new symbol \$. Similarly, if M enters a state for which b's are translucent after the right # symbol, then the skipped $g_2(n)$ symbols b are placed to the end of the input for M' as well. The \$ is necessary to synchronize the computations of M and M' on the endmarker. Exactly when M' sees the \$, M must see the endmarker.

So, we get that M' accepts the non-context-free language

$$\{\, b^n \# b^{n-g_1(n)} \# b^{n-g_2(n)} \$ b^{g_1(n)+g_2(n)} \mid n \geq 1 \,\},$$

for two functions $g_1(n) \leq n$ and $g_2(n) \leq n$. This is a contradiction for the second sub-case, and thus a contradiction to the assumption that M reads all b's from the prefix b^n. Therefore, CASE 3 cannot apply.

CASE 4 - *For infinitely many inputs from L_{tb}, none of the symbols # remains in the input when M sees the endmarker for the first time:*

As in CASE 3, M has to start the computation by reading some constant amount of b's or all b's from the prefix b^n of the input, and then it sees and processes the left # symbol. After having processed the left # symbol, M starts to read zero or more b's from the infix b^n of the input. While doing so, it either reaches a state for which b's are translucent, or it reads all such b's. In both cases, M has to read and process the right # symbol. After having processed the right # symbol, M starts to read zero or more b's from the suffix b^n of the input. While doing so, it either reaches a state for which b's are translucent, or it reads all these b's. Again, in both cases, M reaches the endmarker.

Summarizing so far, on each of the three factors b^n, M can either reach a state for which b's are translucent, or read the factor entirely. Assume now that M reaches a state for which b's are translucent for at least two of the factors, say the first and the last one, and its computation so far is $(q_0 b^n \# b^n \# b^n \lhd, \bot) \vdash^* (b^{f(n)} q \lhd, \gamma)$, for some function $f(n) \leq n$.

Since M is a deterministic device, we obtain that the branch of computation $(q_0 b^{n+1} \# b^n \# b^{n-1} \lhd, \bot) \vdash^* (b^{f(n)} q \lhd, \gamma)$ turns out to be admissible. Therefore, the string $b^{n+1} \# b^n \# b^{n-1}$ would be accepted, a contradiction.

We conclude by considering the case in which M reaches a state for which b's are translucent for at most one of the factors, say the second one. Then, by omitting τ, one can effectively construct a classical DPDA for the non-context-free language $\{\, b^n \# b^{n-f(n)} \# b^n \$ b^{f(n)} \mid n \geq 1 \,\}$, a contradiction. Therefore, even CASE 4 cannot apply.

In conclusion, from the contradictions in all the four cases above discussed, we get that the nrDPDAwtl M can only accept a finite subset of L_{tb}, whence the claimed result follows. □

By this impossibility result, L_{tb} qualifies as a witness language to show

Proposition 7. *The language family $\mathscr{L}$(nrDPDAwtl) is not closed under non-erasing inverse homomorphisms.*

Proof. By Example 2, the language

$$L = \{\, x\#y\#z \mid x, y, z \in \{a, c\}^*, |x|_a = |y|_a = |z|_c \geq 1 \,\}$$

can be accepted by an nrDPDAwtl. So, we let the non-erasing homomorphism $h\colon \{b, \#\}^* \to \{a, c, \#\}^*$ be defined as $h(b) = ac$ and $h(\#) = \#$, thus getting $h^{-1}(L) = L_{tb}$. Since, by Lemma 5, $L_{tb} \notin \mathscr{L}(\text{nrDPDAwtl})$, the non-closure of $\mathscr{L}(\text{nrDPDAwtl})$ under non-erasing inverse homomorphisms follows. □

Finally, the reversal operation is dealt with in the following

Proposition 8. *The language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) are not closed under reversal.*

Proof. We consider the witness language

$$L_{sel} = \{\, b^{j_1}\#b^{j_2}\#\cdots b^{j_m}\#^{m-i+1}b^{j_i} \mid m \geq 1,\ j_1, j_2, \ldots, j_m \geq 0,\ 1 \leq i \leq m \,\}.$$

Since L_{sel}^R is a deterministic context-free language, it is trivially accepted by some DPDAwtl, and thus by some nrDPDAwtl. So, it remains to show that L_{sel} cannot be accepted by any nrDPDAwtl.

By contradiction, we assume that the language L_{sel} is accepted by some nrDPDAwtl $M = \langle Q, \{b, \#\}, \Gamma, q_0, \lhd, \bot, \tau, \delta\rangle$. By Lemma 1, we may safely assume that M accepts only after having processed the whole input. We consider the behavior of M along accepting computations on input strings where m and all j_i's are large enough. We call the $m+1$ maximal factors of the form b^* *blocks*. Basically, M has to start the computation by reading zero or more b's from the first block. The following three cases are then to be considered.

CASE 1: If, in this phase, M reaches a state for which b's and #'s are translucent, then it sees next the endmarker and the input head returns to the left of the input. So, apart from changing possibly the state and performing λ-transitions, nothing changes.

CASE 2: If, in this phase, M reaches only states for which b's are visible, then it continues to process b's until it runs into a state loop of length, say, ℓ. Assume that for one of the states in the loop, say q_1, #'s are translucent. Then we set the numbers j_1 and j_2 such that M is in q_1 exactly after having read b^{j_1} and $b^{j_1+j_2}$ entirely. Since now the following symbol # is translucent, M continues in its state loop. Now, we consider the beginning of an accepting computation

$$\begin{aligned}(q_0 b^{j_1}\#b^{j_2}\#\cdots\#^m b^{j_1}\lhd, \bot) &\vdash^* (q_1\#b^{j_2}\#\cdots\#^m b^{j_1}\lhd, \gamma_1)\\ &\vdash^* (\#q_1\#b^{j_3}\#\cdots\#^m b^{j_1}\lhd, \gamma_2).\end{aligned}$$

This implies that also the computation

$$\begin{aligned}(q_0 b^{j_1+\ell}\#b^{j_2-\ell}\#\cdots\#^m b^{j_1}\lhd, \bot) &\vdash^* (q_1\#b^{j_2-\ell}\#\cdots\#^m b^{j_1}\lhd, \gamma_1')\\ &\vdash^* (\#q_1\#b^{j_3}\#\cdots\#^m b^{j_1}\lhd, \gamma_2)\end{aligned}$$

ends accepting, a contradiction.

CASE 3: If, in this phase, M reaches a state q_1 for which only b's are translucent, then it sees and reads the first # symbol. Now, we repeat the argument for the second block. Again, M reads zero or more b's from the second block. If, in this phase, M reaches a state q_2 for which b's and #'s are translucent, then it sees next the endmarker and the input head returns to the left of the input. Since the input $b^{j_1}\#b^{j_2}\#\cdots\#^m b^{j_1}$ is accepted in a computation that begins as

$$\begin{aligned}(q_0 b^{j_1}\#b^{j_2}\#\cdots\#^m b^{j_1}\lhd, \bot) &\vdash^* (b^{j_1-k_1}q_2 b^{j_2-k_2}\#\cdots\#^m b^{j_1}\lhd, \gamma_1)\\ &\vdash (q_3 b^{j_1-k_1}b^{j_2-k_2}\#\cdots\#^m b^{j_1}\lhd, \gamma_2),\end{aligned}$$

also the input $b^{j_1+1}\#b^{j_2-1}\#\cdots\#^m b^{j_1}$ is accepted in a computation starting with

$$\begin{aligned}(q_0 b^{j_1+1}\#b^{j_2-1}\#\cdots\#^m b^{j_1}\lhd, \bot) &\vdash^* (b^{j_1-k_1+1}q_2 b^{j_2-k_2-1}\#\cdots\#^m b^{j_1}\lhd, \gamma_1)\\ &\vdash (q_3 b^{j_1-k_1+1}b^{j_2-k_2-1}\#\cdots\#^m b^{j_1}\lhd, \gamma_2),\end{aligned}$$

a contradiction.

Similarly, if in the phase of processing the second block M reaches only states for which b's are visible, then M continues to process b's until it runs into a state loop of length, say, ℓ. Assume that for one of the states in the loop, say q_2, #'s are translucent. Then, we set the numbers j_2 and j_3 so that M is in q_2 exactly after having read b^{j_2} and $b^{j_2+j_3}$ entirely. Since the input $b^{j_1}\#b^{j_2}\#b^{j_3}\#\cdots\#^{m-1}b^{j_2}$ is accepted in a computation that starts with

$$\begin{aligned}(q_0 b^{j_1}\#b^{j_2}\#b^{j_3}\#\cdots\#^{m-1}b^{j_2}\lhd, \bot) &\vdash^* (b^{j_1-k_1}q_2\#b^{j_3}\#\cdots\#^{m-1}b^{j_2}\lhd, \gamma_1)\\ &\vdash (b^{j_1-k_1}\#\#q_2\cdots\#^{m-1}b^{j_2}\lhd, \gamma_2),\end{aligned}$$

also $b^{j_1}\#b^{j_2+\ell}\#b^{j_3-\ell}\#\cdots\#^{m-1}b^{j_2}$ is accepted in a computation beginning with

$$\begin{aligned}(q_0 b^{j_1}\#b^{j_2+\ell}\#b^{j_3-\ell}\#\cdots\#^{m-1}b^{j_2}\lhd, \bot) &\vdash^* (b^{j_1-k_1}q_2\#b^{j_3-\ell}\#\cdots\#^{m-1}b^{j_2}\lhd, \gamma_1')\\ &\vdash (b^{j_1-k_1}\#\#q_2\cdots\#^{m-1}b^{j_2}\lhd, \gamma_2),\end{aligned}$$

a contradiction.

Next, if in the phase of processing the second block M reaches a state q_2 for which only b's are translucent, then the input $b^{j_1}\#b^{j_2}\#\cdots\#^m b^{j_1}$ is accepted in a computation that begins as

$$(q_0 b^{j_1}\#b^{j_2}\#\cdots\#^m b^{j_1}\lhd, \bot) \vdash^* (b^{j_1-k_1}q_2 b^{j_2-k_2}\#\cdots\#^m b^{j_1}\lhd, \gamma_1)$$

Table 1
A summary of closure properties for the language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl). The properties not shown in this paper are from [9], and can be found in [4] for DCFL (deterministic context-free languages).

Language Family	-	$\cup$	$\cap$	$\cap_{\text{reg}}$	$\cdot$	$*$	$h_{\text{len.pres.}}$	h^{-1}	R
$\mathscr{L}$(DPDAwtl)	✓	✗	✗	✗	✗	✗	✗	✗	✗
$\mathscr{L}$(nrDPDAwtl)	✓	✗	✗	✗	✗	✗	✗	✗	✗
DCFL	✓	✗	✗	✓	✗	✗	✗	✓	✗

$$\vdash (b^{j_1-k_1} b^{j_2-k_2} q_3 \cdots \#^m b^{j_1} \lhd, \gamma_2).$$

So, also $b^{j_1+1}\#b^{j_2-1}\#\cdots\#^m b^{j_1}$ is accepted in a computation beginning with

$$\begin{aligned}(q_0 b^{j_1+1}\#b^{j_2-1}\#\cdots\#^m b^{j_1}\lhd, \bot) &\vdash^* (b^{j_1-k_1+1} q_2 b^{j_2-k_2-1}\#\cdots\#^m b^{j_1}\lhd, \gamma_1)\\ &\vdash (b^{j_1-k_1+1} b^{j_2-k_2-1} q_3 \cdots \#^m b^{j_1}\lhd, \gamma_1),\end{aligned}$$

a contradiction. This concludes CASE 3.

So far, we have contradicted the assumption that M accepts L_{sel} for all possibilities, except for the case where M starts to read the blocks entirely, without facing translucent letters.

Clearly, the above arguments also apply for the cases where M reads the first k blocks without facing translucent letters, and then encountering translucent letters in block $k+1$ for the first time. In this case, simply, block 1 is replaced by block k and block 2 by block $k+1$.

So, we are left only with the case in which M reads the first m blocks without translucent letters. In particular, this means that the computation on input $b^{j_1}\#b^{j_2}\#\cdots b^{j_m}\#^1 b^{j_{m+1}}$ accepts without translucent letters if and only if $j_m = j_{m+1}$. This is a contradiction since the words of this form cannot be deterministically context-free generated.

In conclusion, we have that L_{sel} cannot be accepted by any nrDPDAwtl. □

For the sake of completeness and reader's ease of mind, we summarize in the following table the complete scenario of closure properties for the language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl).

CRediT authorship contribution statement

Martin Kutrib: Writing – review & editing, Writing – original draft, Investigation; **Andreas Malcher:** Writing – review & editing, Writing – original draft, Investigation; **Carlo Mereghetti:** Writing – review & editing, Writing - original draft, Investigation; **Beatrice Palano:** Writing – review & editing, Writing – original draft, Investigation; **Priscilla Raucci:** Writing – review & editing, Writing – original draft, Investigation; **Matthias Wendlandt:** Writing – review & editing, Writing – original draft, Investigation.

Data availability

No data was used for the research described in the article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Beier, M. Holzer, Nondeterministic right one-way jumping finite automata, Inform. Comput. 284 (2022) 104687.
[2] S. Bensch, H. Bordihn, M. Holzer, M. Kutrib, On input-revolving deterministic and nondeterministic finite automata, Inform. Comput. 207 (2009) 1140–1155.
[3] K. Chigahara, Z. Fazekas, M. Yamamura, One-way jumping finite automata, Int. J. Found. Comput. Sci. 27 (2016) 391–405.
[4] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Massachusetts 1979.
[5] P. Jančar, F. Mráz, M. Plátek, J. Vogel, Restarting automata, in: A. Reichel (Ed.), Fundamentals of Computation Theory (FCT 1995), Springer, (1995), 283–292.
[6] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Descriptional complexity of iterated uniform finite-state transducers, Inform. Comput. 284 (2022) 104691.
[7] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Iterated uniform finite-state transducers: descriptional complexity of nondeterminism and two-way motion, J. Autom. Lang. Comb. 28 (2023) 59–88.
[8] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Iterated uniform finite-state transducers on unary languages, Theor. Comput. Sci. 969 (2023) 114049.
[9] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, M. Raucci, M. Wendlandt, Deterministic pushdown automata with translucent input letters, in: D. Day, F. Manea (Eds.), Developments in Language Theory (DLT 2024), Springer, 2024, pp. 203–217.
[10] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, M. Raucci, M. Wendlandt, On properties of languages accepted by deterministic pushdown automata with translucent input letters, in: Z. Fazekas (Ed.), Implementation and Application of Automata (CIAA 2024), Springer, 2024, pp. 208–220.
[11] A. Meduna, P. Zemek, Jumping finite automata, Int. J. Found. Comput. Sci. 23 (2012) 1555–1578.
[12] V. Mitrana, G. Păun, R. Păun, R.S. Couso, Jump complexity of finite automata with translucent letters, Theor. Comput. Sci. 992 (2024) 114450.
[13] F. Mráz, F. Otto, Non-returning deterministic and nondeterministic finite automata with translucent letters, RAIRO Inform. Théor. 57 (2023) 8.
[14] B. Nagy, F. Otto, Finite-state acceptors with translucent letters, in: G. Bel Enguix, V. Dahl, D.L. Puente (Eds.), AI Methods for Interdisciplinary Research in Language and Biology (BILC 2011), SciTePress, 2011, pp. 3–13.

[15] B. Nagy, F. Otto, On CD-systems of stateless deterministic R-automata with window size one, J. Comput. Syst. Sci. 78 (2012) 780–806.
[16] B. Nagy, F. Otto, Deterministic pushdown-CD-systems of stateless deterministic r(1)-automata, Acta Inform. 50 (2013) 229–255.
[17] F. Otto, A survey on automata with translucent letters, in: B. Nagy (Ed.), Implementation and Application of Automata (CIAA 2023), Springer, 2023, pp. 21–50.
[18] F. Otto, Restarting Automata, Springer, Cham, Switzerland 2025.
[19] W.J. Savitch, P.M.B. Vitányi, Linear time simulation of multihead turing machines with head-to-head jumps, in: A. Salomaa, M. Steinby (Eds.), International Colloquium on Automata, Languages and Programming (ICALP 1977), Springer, 1977, pp. 453–464.