Information and Computation 308 (2026) 105403

Contents lists available at ScienceDirect

Information and Computation

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

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, Giessen, 35392, Germany
[b] Dipartimento di Informatica "Giovanni Degli Antoni", Università degli Studi di Milano, via Celoria 18, Milano, 20133, Italy
[c] Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM), Italy

ARTICLE INFO

Article history:
Received 21 February 2025
Received in revised form 30 December 2025
Accepted 7 January 2026
Available online 9 January 2026

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

ABSTRACT

The use of translucent input letters represents a way of implementing a discontinuous input processing in automata. In detail, a translucent automaton performs several sweeps from left to right on the input: according to the current state, some symbols are visible and can be processed, whereas some other symbols are invisible and may be processed in another sweep. We also distinguish between the returning and non-returning mode, which differ in the way the automaton behaves after reading a symbol: in the returning mode, a new sweep starts immediately, while in the non-returning mode, the device processes the next visible symbol.

Here, we investigate *deterministic pushdown automata with translucent letters* both in the returning and non-returning mode. We prove that the non-returning mode strictly outperforms the returning mode, and that the families of the languages accepted by these two types of devices can be ranked strictly between the deterministic context-free languages and the deterministic context-sensitive languages. Moreover, both families are shown to be incomparable to the families of context-free, growing context-sensitive, and Church-Rosser languages. The ability of accepting non-semilinear languages is also emphasized (addressing an open question in the literature). Finally, we study the closure properties of both language families under the Boolean operations, obtaining that they are both closed under complementation but not under union and intersection. Further non-closure results are pointed-out for returning devices.

☆ A preliminary version of this work [1] was presented at the *28th International Conference on Developments in Language Theory (DLT)*, August 12–16, 2024, Göttingen, Germany.

* Corresponding author at: Dipartimento di Informatica "Giovanni Degli Antoni", Università degli Studi di Milano, via Celoria 18, Milano, 20133, Italy.
E-mail addresses: kutrib@informatik.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).
URLs: http://www.informatik.uni-giessen.de/staff/kutrib.html (M. Kutrib), http://www.informatik.uni-giessen.de/staff/malcher (A. Malcher), http://mereghetti.di.unimi.it (C. Mereghetti), http://palano.di.unimi.it (B. Palano), http://www.uni-giessen.de/de/fbz/fb07/fachgebiete/mathematik/informatik/personal/wendlandt (M. Wendlandt).

https://doi.org/10.1016/j.ic.2026.105403

1. Introduction

Automata models with unidirectional motion of the reading head generally process input strings in a continuous way: they read their input string strictly from left to right and accept by entering an accepting state, usually after sweeping the whole input. Typical examples of such one-way devices are: deterministic and nondeterministic, probabilistic and quantum, multihead finite state automata, pushdown and queue automata, transducers.

Recently, several automata models processing their input in a *discontinuous* way are also investigated. As an example, we recall *jumping finite automata*, introduced in [2]. Such devices are similar to classical finite state automata with the difference that, after reading an input symbol, the input head can jump to any position on the left-to-be-processed part of the input string. In their deterministic variant, such automata can accept even non-context-free languages, such as the language $L = \{ w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c \}$. The restricted variant of "right one-way jumping" finite automata is considered in [3,4].

Another example of a discontinuous computation device is given by the notion of an *input-revolving finite automaton*, introduced in [5]. This automaton can move or interchange input symbols to the left end or right end of the input. Also this particular way of discontinuously processing the input increases the computational power of finite automata since, e.g., the above-mentioned context-sensitive language L can be accepted. *Restarting automata*, introduced in [6], are a restricted variant of linear bounded automata. These models process the input string in several left-to-right sweeps, by shifting a read/write window of size $k \geq 1$ along the input: the content of the window is rewritten exactly once into a shorter string at some time during every sweep. After rewriting, the automaton restarts the computation in its initial state and with the read/write window on the leftmost symbol of the string remaining on the input tape. Starting from [6], a consolidated literature explores many variants, generalizations, and aspects of the original model. A summary of the results obtained along this line of research may be found in [7].

In this paper, we study *deterministic pushdown automata with translucent letters* (DPDAwtl). The key feature of this model is the existence of a translucency function that determines, according to the current state, which input letters are translucent. Thus, a DPDAwtl works similarly to a standard deterministic pushdown automaton (DPDA), but with a key difference: at each step it uses the translucency function to skip over the translucent part of the input string from the position of the input head rightward, and processes the first visible (i.e., non translucent) input symbol. After processing a symbol, the automaton behaves according to its mode, which can be:

(i) *returning*, where the input head returns to the left end of the input string;
(ii) *non-returning*, where the input head continues to scan the input rightward.

In both modes, the input head returns to the leftmost input character when the right end of the input string is reached. The concept of translucent letters is introduced by Nagy and Otto in [8], for deterministic and nondeterministic finite state automata. Such automata have connections with restarting automata, since they are, in the returning mode, equivalent to certain cooperating distributed systems of deterministic restarting automata of window size 1. Deterministic and nondeterministic finite state automata with translucent letters are deeply investigated in the literature (see, e.g., [9–12]), and many questions are still open.

The differences between "right one-way jumping" finite automata (ROWJFA) and non-returning deterministic finite automata with translucent letters (nrDFAwtl) are discussed by Otto in the survey paper [12]. In fact, a crucial role is played by an endmarker which is available for automata with translucent letters but not for jumping automata. In [13] various automata models with the jumping paradigm are studied. In particular, ROWJFA equipped with a pushdown are considered. For example, it is shown that the language $\{ a^n b^n c^n \mid n \geq 0 \}$ is not accepted by any right one-way jumping pushdown automaton. However, this language is accepted even by an nrDFAwtl. Therefore, right one-way jumping deterministic pushdown automata are strictly weaker than non-returning deterministic pushdown automata with translucent letters. While we are not aware that right one-way deterministic jumping pushdown automata with endmarker have ever been considered in the literature, they certainly have the same computational power as non-returning deterministic pushdown automata with translucent letters.

Recently, in [14], the translucency and jumping paradigms are put together in the model of a deterministic or nondeterministic jumping finite automaton with translucent letters: in particular, an interesting investigation is proposed on the minimal amount of jumps required for these devices to accept non-regular languages.

Returning pushdown automata with translucent letters are mentioned by Nagy and Otto [11,15] as possible interpretation of cooperating distributed systems of certain restarting automata. Formally, they have been introduced in the survey [12], which also presents some related results. The deterministic and nondeterministic pushdown automata introduced by Nagy and Otto, because of their interpretation, are designed to be real time (i.e., working without λ-transitions) and to accept by empty pushdown. It should be stressed that both these conditions critically limit the computational power of DPDA, since it is well-known that DPDA with λ-transitions and recognition by accepting states are strictly more powerful. In the light of this, to be closer to the original definition and have a fairer comparison with classical DPDA, here we define DPDAwtl that can perform λ-transitions and that accept by accepting states.

- With these two added model assumptions, we are able to show that every deterministic context-free language can be accepted by a *returning* DPDAwtl, and that the family of languages accepted by returning DPDAwtl is closed under complementation. We draw attention to the fact that both these results do not hold true for DPDAwtl in the definition of Otto [12]. In addition, the fact that returning DPDAwtl can accept some non-context-free languages shows that translucency increases the recognition power of deterministic pushdown automata.
- We then investigate DPDAwtl working in the *non-returning* mode (nrDPDAwtl), which is a computational model that has not been investigated before. We show that the family of the languages accepted by nrDPDAwtl strictly contains that of the languages accepted by returning DPDAwtl. This, again, witnesses translucency as a computational enhancer in the returning vs. non-returning question.

It is worth remarking that translucency is proved in [9] to increment recognition power on finite state devices too. Thus, our results extend the relevance of translucency as a mechanism for improving computational power for further kinds of restricted memory automata. To deepen properties of our translucent language families, we investigate relations with well-known language families as well as closure properties.

- We show that, differently from standard pushdown automata, nrDPDAwtl can accept even non-semilinear languages. This fact may be seen as a first step toward approaching an open problem stated in [9], where the existence of non-semilinear languages recognized by translucent finite state automata is questioned. On the other hand, as for standard pushdown automata, nrDPDAwtl are shown to accept only semilinear languages (i.e., regular) when restricted to work on unary inputs.
- Next, we assess the recognition power of our translucent devices with respect to well-known types of automata. First, we strictly confine the families of the languages accepted by DPDAwtl and nrDPDAwtl between deterministic context-free and deterministic context-sensitive languages. Then, we show that these two language families are both incomparable with traditional families, such as, context-free, growing context-sensitive and Church-Rosser languages. We obtain incomparability and proper containment results by suitably designing witness languages. Fig. 1 on page 12 summarizes our results, showing that DPDAwtl and nrDPDAwtl may provide new degrees of language complexity between the deterministic context-free and the deterministic context-sensitive languages.
- Finally, we study closure properties under Boolean language operations, for the families of the languages accepted by DPDAwtl and nrDPDAwtl. Both families turn out to be closed under complementation. They are neither closed under union nor under intersection. In addition, we show that the family of the languages recognized by DPDAwtl maintains non-closure even by restricting union and intersection to act with regular languages.

The paper is organized as follows. In Section 2, we overview basic notions and provide the formal definitions of returning DPDAwtl and non-returning DPDAwtl (nrDPDAwtl). In Section 3, to simplify proofs, we show that DPDAwtl and nrDPDAwtl can always, and without loss of generality, execute loop-free computations and completely consume the input. In Section 4 and Section 5, we study the computational capacity of these two translucent devices, by focusing on their relation with well-known recognition devices. The resulting scenario is briefly summarized in Fig. 1 on page 12. In Section 6, we study closure properties under Boolean language operations. Finally, in Section 7, we sum up our results and outline possible research prospects.

2. Definitions and preliminaries

Given a set S, we denote by 2^S its power set, by $|S|$ its cardinality, and by S_x the set $S \cup \{x\}$, for a given $x \notin S$. We use $\subseteq$ for inclusion and $\subset$ for strict inclusion. We write Σ^* for the set of all finite words on the finite alphabet Σ, including the empty word λ, and we let $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of $w \in \Sigma^*$ is denoted by w^R, the *length* of w by $|w|$, and the number of occurrences of a symbol $a \in \Sigma$ in w by $|w|_a$. A language on Σ is any subset $L \subseteq \Sigma^*$, its complement is the language $\overline{L} = \Sigma^* \setminus L$. We say that two language families $\mathscr{L}_1$ and $\mathscr{L}_2$ are incomparable whenever $\mathscr{L}_1$ is not a subset of $\mathscr{L}_2$ and vice versa.

2.1. Translucent pushdown automata

Pushdown automata with translucent letters extend classical pushdown automata by enabling discontinuous input processing steps. Precisely, depending on the current state, some of the input letters may become translucent (invisible), and hence skipped. In a computational step, a pushdown automaton with translucent letters can either perform a λ-transition without reading an input symbol or read and process the first visible input letter from the left. Here, we are particularly interested in *deterministic* computations.

A *deterministic pushdown automaton with translucent letters* (DPDAwtl) is formally defined as an 8-tuple $M=\langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta\rangle$, where:

- Q is the finite set of internal states,
- Σ is the finite alphabet of input symbols, with $\Sigma \cap Q = \emptyset$,

- Γ is the finite alphabet of pushdown symbols,
- $q_0 \in Q$ is the initial state,
- $\lhd \notin \Sigma$ is the *endmarker symbol*,
- $\bot \notin \Gamma$ is the *bottom-of-pushdown symbol*,
- $\tau\colon Q \to 2^{\Sigma}$ is the *translucency mapping*: for each state $q \in Q$, the letters from the set $\tau(q)$ are *translucent for* q, meaning that, whenever in q, the automaton M does not see (or better, looks through and skips) these letters,
- $\delta\colon Q \times (\Sigma \cup \{\lambda, \lhd\}) \times \Gamma_{\bot} \to (Q \times \Gamma_{\bot}^*) \cup \{\texttt{accept}\}$ is the *partial* transition function, for which there must never be a choice between using an input symbol and using λ input. Formally, this condition writes as: for any $q \in Q, z \in \Gamma_{\bot}$, if $\delta(q, \lambda, z)$ is defined, then $\delta(q, a, z)$ is undefined for every $a \in \Sigma_{\lhd}$. In addition, to simplify matters, we require that along computations the bottom-of-pushdown symbol appears exactly once at the bottom of the pushdown store (i.e., it can never be deleted), formally: if $\delta(q, a, z) = (p, \beta)$ then either $z \neq \bot$ and $\beta \in \Gamma^*$, or $z = \bot$ and $\beta = \beta'\bot$ for $\beta' \in \Gamma^*$.

A configuration of M is represented either by $\texttt{accept}$ or by a pair $(qw\lhd, \gamma)$, where $q \in Q$ is the current state, $w \in \Sigma^*$ is the remaining part of the input, and $\gamma \in \Gamma^*\bot$ denotes the current pushdown content, the leftmost symbol being the top of the pushdown store. On input $x \in \Sigma^*$, the initial configuration is defined to be $(q_0x\lhd, \bot)$.

During the course of its computation, M runs through a sequence of configurations. Being in some configuration $(qw\lhd, z\gamma)$, a step of M occurs as follows. First, M checks whether $\delta(q, \lambda, z)$ is defined. If it is the case and $\delta(q, \lambda, z) = (p, \beta)$ (respectively, $\delta(q, \lambda, z) = \texttt{accept}$), then, the successor configuration is $(pw\lhd, \beta\gamma)$ (respectively, $\texttt{accept}$). Otherwise, $\delta(q, \lambda, z)$ is undefined, and M determines the input letter to be processed by taking the first letter from the left that is visible in state q; i.e., if $w = uav$ with $u \in \tau(q)^*$ and $a \notin \tau(q)$, then M processes a. The automaton halts and rejects whenever $\delta(q, a, z)$ is undefined, otherwise it computes $\delta(q, a, z) = (p, \beta)$ (respectively, $\delta(q, a, z) = \texttt{accept}$) and the successor configuration becomes $(puv\lhd, \beta\gamma)$ (respectively, $\texttt{accept}$). Whenever M is in configuration $\texttt{accept}$ it halts accepting. In the case $w \in \tau(q)^*$, the automaton M sees the endmarker $\lhd$, it halts and accepts if and only if $\delta(q, \lhd, z) = \texttt{accept}$. More precisely, one step from a configuration to its successor configuration is denoted by $\vdash$, and is formally defined as follows. Given $p, q \in Q$, $a \in \Sigma$, $u, v, w \in \Sigma^*$, $z \in \Gamma_{\bot}$ and $\beta\gamma, z\gamma \in \Gamma^*\bot$, we let:

- $(qw\lhd, z\gamma) \vdash (pw\lhd, \beta\gamma)$, if $(p, \beta) = \delta(q, \lambda, z)$,
- $(qw\lhd, z\gamma) \vdash \texttt{accept}$, if $\texttt{accept} = \delta(q, \lambda, z)$,
- $(quav\lhd, z\gamma) \vdash (puv\lhd, \beta\gamma)$, if $u \in \tau(q)^*$, $a \notin \tau(q)$ and $(p, \beta) = \delta(q, a, z)$,
- $(quav\lhd, z\gamma) \vdash \texttt{accept}$, if $u \in \tau(q)^*$, $a \notin \tau(q)$ and $\texttt{accept} = \delta(q, a, z)$,
- $(qw\lhd, z\gamma) \vdash (pw\lhd, \beta\gamma)$, if $w \in \tau(q)^*$ and $(p, \beta) = \delta(q, \lhd, z)$,
- $(qw\lhd, z\gamma) \vdash \texttt{accept}$, if $w \in \tau(q)^*$ and $\texttt{accept} = \delta(q, \lhd, z)$.

We denote the reflexive and transitive (respectively, transitive) closure of $\vdash$ by $\vdash^*$ (respectively, $\vdash^+$). Therefore, the language accepted by M is defined to be the set

$$L(M) = \{\, w \in \Sigma^* \mid (q_0w\lhd, \bot) \vdash^+ \texttt{accept} \,\}.$$

The following example clarifies the behavior of DPDAwtl.

Example 1. Let $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta\rangle$ be a DPDAwtl with $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{c\}$, and the functions τ and δ defined as follows. Given $z \in \Gamma_{\bot}$, we let:

$$\begin{array}{lrcl}
\tau(q_0) = \{b\}, & \delta(q_0, a, z) & = & (q_1, z),\\
 & \delta(q_0, c, c) & = & (q_2, \lambda),\\
 & \delta(q_0, \lhd, \bot) & = & \texttt{accept},\\
\tau(q_1) = \{a\}, & \delta(q_1, b, z) & = & (q_0, cz),\\
\tau(q_2) = \emptyset, & \delta(q_2, c, c) & = & (q_2, \lambda),\\
 & \delta(q_2, \lhd, \bot) & = & \texttt{accept}.
\end{array}$$

For all other cases, δ is undefined. To get an intuition of the dynamics of M, we show its execution with input word *abbacc*. Then, the computation occurs as:

$$\begin{array}{lll}
(q_0abbacc\lhd, \bot) & \vdash (q_1bbacc\lhd, \bot) & \vdash (q_0bacc\lhd, c\bot)\\
 & \vdash (q_1bcc\lhd, c\bot) & \vdash (q_0cc\lhd, cc\bot)\\
 & \vdash (q_2c\lhd, c\bot) & \vdash (q_2\lhd, \bot) \qquad \vdash \texttt{accept}.
\end{array}$$

More generally, it is not hard to see that the language accepted by M is the set

$$L(M) = \{\, xy \mid x \in \{a, b\}^* \wedge\ n = |x|_a = |x|_b\ \wedge\ y = c^n \,\}.$$

We point out that this language cannot be recognized by any DFAwtl, since it does not contain any letter-equivalent subset that is regular [10].

In the realm of translucent finite state automata, the *non-returning* mode is also considered in [9]. According to this paradigm, after processing a visible letter, the head of the automaton does not return to the left end of the input but it continues from the position of the letter just processed. Whenever the endmarker is reached and the transition on the endmarker yields a state q, the computation is continued in q with the head set on the leftmost symbol of the remaining input. We here adopt this alternative way of processing input for our DPDAwtl and obtain *deterministic pushdown automata with translucent letters working in the non-returning mode* (nrDPDAwtl).

An nrDPDAwtl is formally defined as an 8-tuple $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \bot, \tau, \delta \rangle$, where all the components are defined as in the DPDAwtl model. However, the computational step now reflects the non-returning nature. A configuration of M is represented either by `accept` or by a pair $(xqw\triangleleft, \gamma)$, where $q \in Q$ is the current state, $xw \in \Sigma^*$ is the remaining part of the input, with x to the left and w to the right of the input head, and $\gamma \in \Gamma^*\bot$ denotes the current pushdown content, the leftmost symbol being the top of the pushdown store. The successor configuration induced by $\vdash$ is now formally specified as follows. Given $p, q \in Q$, $a \in \Sigma$, $x, u, v, w \in \Sigma^*$, $z \in \Gamma_\bot$ and $\beta\gamma, z\gamma \in \Gamma^*\bot$, we have:

- $(xqw\triangleleft, z\gamma) \vdash (xpw\triangleleft, \beta\gamma)$, if $(p, \beta) = \delta(q, \lambda, z)$,
- $(xqw\triangleleft, z\gamma) \vdash \texttt{accept}$, if $\texttt{accept} = \delta(q, \lambda, z)$,
- $(xquav\triangleleft, z\gamma) \vdash (xupv\triangleleft, \beta\gamma)$, if $u \in \tau(q)^*$, $a \notin \tau(q)$ and $(p, \beta) = \delta(q, a, z)$,
- $(xquav\triangleleft, z\gamma) \vdash \texttt{accept}$, if $u \in \tau(q)^*$, $a \notin \tau(q)$ and $\texttt{accept} = \delta(q, a, z)$,
- $(xqw\triangleleft, z\gamma) \vdash (pxw\triangleleft, \beta\gamma)$, if $w \in \tau(q)^*$ and $(p, \beta) = \delta(q, \triangleleft, z)$,
- $(xqw\triangleleft, z\gamma) \vdash \texttt{accept}$, if $w \in \tau(q)^*$ and $\texttt{accept} = \delta(q, \triangleleft, z)$.

The notions of $\vdash^*$, $\vdash^+$ and of the language $L(M)$ accepted by M are defined as above. Sometimes, we say 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 return move on the endmarker (if it takes place).

Let us give an example of language recognition by nrDPDAwtl.

Example 2. We are going to design an nrDPDAwtl M accepting the non-context-free language

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

Let $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \bot, \tau, \delta \rangle$ be such that $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, c, \#\}$, $\Gamma = \{a\}$. We clarify the translucent function τ and the transition function δ while describing the dynamics of M. At the beginning of the computation, M starts processing the first block of the input string, namely x, and stores $|x|_a$ many symbols a in the pushdown. In this phase, M reads (and erases) all characters in x, until the first # is reached. To implement this behavior, we proceed as follows, where $z \in \Gamma_\bot$:

$$\begin{array}{lrcl} \tau(q_0) = \emptyset, & \delta(q_0, a, z) & = & (q_0, az), \\ & \delta(q_0, c, z) & = & (q_0, z), \\ & \delta(q_0, \#, z) & = & (q_1, z). \end{array}$$

Then, M continues the computation by processing the second and the third block of the string, y and v respectively, deleting c's from y and a's from v. Eventually, M reads the right endmarker and moves the head back to the beginning of the tape:

$$\begin{array}{lrcl} \tau(q_1) = \{a\}, & \delta(q_1, c, z) & = & (q_1, z), \\ & \delta(q_1, \#, z) & = & (q_2, z), \\ \tau(q_2) = \{c\}, & \delta(q_2, a, z) & = & (q_2, z), \\ & \delta(q_2, \triangleleft, z) & = & (q_3, z). \end{array}$$

Assuming that M is running an accepting computation, it is not hard to see that at this point the configuration of M is of the form $(q_3\, a^n c^n \triangleleft, a^n)$. To check whether the number of a's and c's coincides with the number of symbols stored in the pushdown, M deletes one c in the second block, and pops one symbol of the pushdown for each symbol a read in the first block. Finally, M accepts if and only if it processes the endmarker with empty pushdown and no symbol left on the tape:

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

Summing up, at a high level, the whole computation of M can be considered as being divided into two parts:

- The first part, taking place in the first sweep, stores $|x|_a$ many symbols a in the pushdown, while non-relevant characters in the rest of the string are deleted. E.g., on a word $x\#y\#v$, this first phase simply looks as follows:

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

where $i = |y|_a$, $j = |v|_c$ and $h = |x|_a$.

- In the second part, the head of M moves back and forth on the tape, while deleting symbols from both the tape string and the pushdown. Precisely, along this phase, M basically repeats the following cycle:

$$\begin{array}{lll} (q_3\, a^i c^j \lhd, a^h \bot) & \vdash (q_4\, a^{i-1} c^j \lhd, a^h \bot) & \vdash (a^{i-1}\, q_5\, c^{j-1} \lhd, a^h \bot) \vdash \\ (q_3\, a^{i-1} c^{j-1} \lhd, a^{h-1} \bot) & \vdash (q_4\, a^{i-2} c^{j-1} \lhd, a^{h-1} \bot) & \vdash (a^{i-2}\, q_5\, c^{j-2} \lhd, a^{h-1} \bot) \vdash \cdots. \end{array}$$

It is not hard to see that the input string is accepted if and only if $|x|_a = |y|_a = |v|_c$: in this case, in fact, M reaches the configuration $(q_3 \lhd, \bot)$, leading to acceptance by the rule $\delta(q_3, \lhd, \bot) = \texttt{accept}$. In any other case, M rejects.

Throughout the rest of the paper, we will be saying that two automata A and B are (language) equivalent whenever $L(A) = L(B)$. Moreover, we denote the family of all languages accepted by some accepting device of type X by $\mathscr{L}(\mathsf{X})$.

3. Model technical assumptions

According to the definitions given in Section 2, our DPDAwtl and nrDPDAwtl may in principle trigger both the following dynamics:

(i) They may halt while still leaving some characters on the input tape.
(ii) They may reject by looping with λ-transitions. In fact, they may enter a sequence of λ-transitions beginning from and ending into the same state and with the same pushdown top, along which no input symbol is deleted (this is obvious since only λ-transitions are performed) and the pushdown height is not decreased in total. This clearly implies that the λ-loop will continue forever, not being stopped by input or pushdown consumption.

By considering DPDAwtl and nrDPDAwtl where these two dynamics do not occur, proofs become more readable. Thus, in what follows, we show that our translucent devices can always, and without loss of generality, be assumed free from dynamics (i) and (ii).[1] Let us begin by removing (i):

Lemma 1. *From any given DPDAwtl (respectively, nrDPDAwtl), an equivalent DPDAwtl (respectively, nrDPDAwtl) can effectively be constructed that halts only after having processed the input entirely.*

Proof. Given a DPDAwtl or nrDPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta \rangle$, we construct an equivalent DPDAwtl or nrDPDAwtl M', which accepts only after completely consuming the whole input. We let $Q' = Q \cup \{r_+, r_-, s_+, s_-\}$ with $r_+, r_-, s_+, s_- \notin Q$, τ' coincide with τ on Q, and set $\tau'(r_+) = \tau'(r_-) = \Sigma$ and $\tau'(s_+) = \tau'(s_-) = \emptyset$. Next, we first let δ' coincide with δ. Then, any instruction in M of the form $\delta(q, a, z) = \texttt{accept}$, $a \in \Sigma \cup \{\lambda, \lhd\}$, is replaced by the transitions

$$\underbrace{\{\delta'(q, a, z) = (r_+, z)\}}_{(a)} \cup \underbrace{\{\delta'(r_+, \lhd, z) = (s_+, z)\}}_{(b)} \cup \underbrace{\left(\cup_{\sigma \in \Sigma} \{\delta'(s_+, \sigma, z) = (s_+, z)\}\right)}_{(c)} \cup \underbrace{\{\delta'(s_+, \lhd, z) = \texttt{accept}\}}_{(d)}. \quad (1)$$

Similarly, for each $a \in \Sigma_\lhd$, if the instructions $\delta(q, \lambda, z)$ and $\delta(q, a, z)$ are undefined, we add transitions

$$\underbrace{\{\delta'(q, a, z) = (r_-, z)\}}_{(a)} \cup \underbrace{\{\delta'(r_-, \lhd, z) = (s_-, z)\}}_{(b)} \cup \underbrace{\left(\cup_{\sigma \in \Sigma} \{\delta'(s_-, \sigma, z) = (s_-, z)\}\right)}_{(c)}, \quad (2)$$

and let $\delta'(s_-, \lhd, z)$ be undefined.

The transitions (1), occurring upon acceptance of M, basically: (a) drive M' into the new state r_+, (b) return the input head to the left and drive M' into state s_+, (c) consume the input symbol by symbol, and (d) accept in a final step on the endmarker.

Similarly, the added transitions (2) deal with situations in which M halts rejecting. □

Now, let us get rid of (ii):

Lemma 2. *From any given DPDAwtl (respectively, nrDPDAwtl), an equivalent DPDAwtl (respectively, nrDPDAwtl) that halts on any input can effectively be constructed.*

[1] A similar result for classical deterministic pushdown automata can be found in [16].

Proof. Given a DPDAwtl or nrDPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta \rangle$, we are going to show how to detect and delete from M any possible infinite loop consisting of λ-transitions, thus obtaining the claimed equivalent halting DPDAwtl or nrDPDAwtl M'.

Suppose there exists an input string (clearly, not in $L(M)$) upon which M runs into a non-halting computation. Eventually, M runs through a sequence of λ-transitions starting and ending into the same state, and with the same pushdown top again and again. During this sequence, some finite number of topmost symbols may be accessed by popping, but the total pushdown height must not be decreased. Therefore, other symbols have to be pushed again. Since there are only finitely many possibilities to push this finite number of symbols, we can choose the sequence such that these topmost symbols are the same. In this phase of the computation, the configurations of M can safely be specified in the "restricted" form $(q, z\gamma)$, where $z \in \Gamma_\bot$, $z\gamma \in \Gamma^*\bot$, and the input tape does not show up.

It is not hard to see that the following *reachability property* turns out to be a necessary and sufficient condition for a non-halting computation to exist:

$$(q, z\beta\gamma) \vdash^* (q, z\beta\alpha\gamma) \text{ by } \lambda\text{-transitions only,}$$

where the factor β is popped and the factor $\beta\alpha$ is pushed along an iteration of the sequence, while γ is the pushdown content not affected. (Note that $|\beta\alpha\gamma| \geq |\beta\gamma|$, since otherwise at least one symbol in the pushdown would be consumed, and this would eventually lead M to hit the $\bot$ symbol.)

So, to construct M', we inspect the transition function δ of M for chains of λ-transitions witnessing the reachability property mentioned above, for given $q \in Q$ and $z \in \Gamma_\bot$. If this is the case, we let $\delta'(q, \lambda, z)$ undefined, thus leading M' to reject. Apart from this modification, δ' coincides with δ. □

In conclusion, from Lemma 1 and Lemma 2, the following proposition comes straightforwardly:

Proposition 1. *From any given DPDAwtl (respectively, nrDPDAwtl), an equivalent DPDAwtl (respectively, nrDPDAwtl) that halts on any input by accepting or rejecting after the* whole *input has been processed, can effectively be constructed.*

4. Comparing returning and non-returning translucency

In order to start getting some understanding of the computational capabilities of our translucent devices, we begin by anayizing the impact on their recognition power of the two input processing modes, namely returning and non-returning. In other words, we are going to investigate the relation between the language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl), and start by showing that the non-returning mode can simulate the returning mode.

Lemma 3. *From any given DPDAwtl, an equivalent nrDPDAwtl can effectively be constructed.*

Proof. By Lemma 1, from any given DPDAwtl an equivalent DPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta \rangle$ that accepts only after processing the input entirely can be constructed. In turn, from M, we construct an equivalent nrDPDAwtl M' which, 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 δ modifies to δ', so that the state $q' \in Q'$ is entered if and only if M enters the state $q \in Q$. The translucency mapping τ extends to τ' 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, δ' is extended by $\delta'(q', \lhd, z) = (q, z)$, for all $q' \in Q'$ and all $z \in \Gamma_\bot$, thus bringing the input head to the leftmost position. One may easily verify that δ' leads to acceptance if and only if δ does as well. □

Lemma 3 clearly implies $\mathscr{L}$(DPDAwtl) $\subseteq \mathscr{L}$(nrDPDAwtl). Now, we are going to show that this inclusion is proper. The following result has been proved in [8] for finite automata with translucent letters and can be adapted to translucent deterministic pushdown automata:

Proposition 2. *From any given DPDAwtl M, a DPDA M' can effectively be constructed such that $L(M') \subseteq L(M)$ and $L(M')$ is letter-equivalent to $L(M)$.*

Proof. By Lemma 1, from M an equivalent DPDAwtl N that accepts only after processing the input entirely can be constructed. Next, we consider the DPDA N' obtained from N by simply removing the translucency relation (equivalently, by setting to the empty set the translucency from every state in N). As a result, we have that N' is a classical DPDA that receives its input with an endmarker. Since the family of deterministic context-free languages is effectively closed under right quotient with a singleton, a DPDA M' can be constructed from N', and the new device accepts $L(N')$ without endmarker (formally, $L(M') = L(N') \cdot \lhd^{-1}$). So, it is easy to see that the symbols of any input string (except the final endmarker) accepted by M can be rearranged to form an input string accepted by M'. Moreover, any input string accepted by M' is

clearly accepted by N as well (upon equipping it with the final endmarker). This shows that $L(M')$ is letter-equivalent to $L(N) = L(M)$. □

Example 3. As an immediate corollary to Proposition 2, we get that all languages in $\mathscr{L}(\mathsf{DPDAwtl})$ are semilinear, given the well-known result that all the languages in $\mathsf{CFL} \supset \mathscr{L}(\mathsf{DPDA})$ are semilinear [17]. The context-sensitive language $L_{abc} = \{a^n b^n c^n \mid n \geq 0\}$ is easily seen to not contain any letter-equivalent deterministic context-free sub-language. Therefore, we can conclude that $L_{abc} \notin \mathscr{L}(\mathsf{DPDAwtl})$.

We point out that Proposition 2 does not hold for the non-returning mode. Moreover, from [9], we have that L_{abc} is accepted by a deterministic finite automaton with translucent letters working in the non-returning mode, and so by some nrDPDAwtl as well. This, together with Lemma 3 and Example 3, yields

Theorem 1. *The family $\mathscr{L}(\mathsf{DPDAwtl})$ is properly included in $\mathscr{L}(\mathsf{nrDPDAwtl})$.*

Since the proper inclusion $\mathscr{L}(\mathsf{DPDAwtl}) \subset \mathscr{L}(\mathsf{nrDPDAwtl})$ is witnessed by the semilinear language L_{abc}, a natural question arises of whether all languages in $\mathscr{L}(\mathsf{nrDPDAwtl})$ are semilinear. The following theorem negatively answers this question, and to prove it we consider the language

$$L_{sq} = \{\, a\,\#\,a^3\,\#^2\,a^5\,\#^3 \cdots \#^k\,a^{2k+1} \mid k \geq 0 \,\}.$$

Theorem 2. *The family $\mathscr{L}(\mathsf{nrDPDAwtl})$ includes a non-semilinear language.*

Proof. The language L_{sq} is not semilinear, since the number of a's in its words is a square number. Let us now show that $L_{sq} \in \mathscr{L}(\mathsf{nrDPDAwtl})$. We define the nrDPDAwtl $M = \langle Q, \{a, \#\}, \Gamma, p_0, \lhd, \bot, \tau, \delta \rangle$, where:

- $Q = \{\, p_i \mid 0 \leq i \leq 3 \,\} \cup \{\, q_i \mid 0 \leq i \leq 5 \,\}$,
- $\Gamma = \{A\}$,
- $\tau(p_0) = \tau(p_1) = \emptyset$, $\tau(p_2) = \{\#\}$, $\tau(p_3) = \{a\}$,
  $\tau(q_0) = \tau(q_1) = \tau(q_2) = \tau(q_4) = \emptyset$, $\tau(q_3) = \{\#\}$, $\tau(q_5) = \{a\}$.

We design the transition function δ so that, along the first sweep, M first checks whether the input string starts with an a by deleting it, and then it counts the number of blocks of the form $\#^i a^j$, with $i, j > 0$. To this latter aim, a symbol A is pushed per each of such blocks encountered, and each block is shrunk to $\#^{i-1} a^{j-1}$. To implement this first sweep, we formally define δ as:

$$\begin{array}{llcllllcl}
(1) & \delta(p_0, a, \bot) & = & (p_1, \bot), & (4) & \delta(p_2, a, A) & = & (p_3, A), \\
(2) & \delta(p_1, \lhd, \bot) & = & \mathtt{accept}, & (5) & \delta(p_3, \lhd, A) & = & (q_0, A), \\
(3) & \delta(p_1, \#, \bot) & = & (p_2, A\bot), & (6) & \delta(p_3, \#, A) & = & (p_2, AA).
\end{array}$$

After the first sweep, an input string $a\,\#\,a^3\,\#^2\,a^5\,\#^3\,a^7 \cdots \#^{k+1}\,a^{2(k+1)+1} \in L_{sq}$ reduces to $a^2\,\#\,a^4\,\#^2\,a^6 \cdots \#^k\,a^{2(k+1)}$. In the next sweep, M begins by deleting the first two occurrences of a from the input, while popping A from the pushdown. Then, each block of the form $\#^i a^j$ encountered, with $i, j > 0$, shrinks to $\#^{i-1} a^{j-2}$. At the end of this second sweep, the input string reduces to $a^2\,\#\,a^4\,\#^2\,a^6 \cdots \#^{k-1}\,a^{2k}$. Notice that this remaining string has the same form of the initial string processed along this sweep, but without the last block $\#^k\,a^{2(k+1)}$. As the input form is preserved, this sweep can be iterated, completely consuming the input string and emptying the pushdown if and only if the original input string belongs to L_{sq}. To implement these latter deleting sweeps, we formally define δ as:

$$\begin{array}{llcllllcl}
(7) & \delta(q_0, a, A) & = & (q_1, A), & (11) & \delta(q_3, a, A) & = & (q_4, A), \\
(8) & \delta(q_1, a, A) & = & (q_2, \lambda), & (12) & \delta(q_4, a, A) & = & (q_5, A), \\
(9) & \delta(q_2, \lhd, \bot) & = & \mathtt{accept}, & (13) & \delta(q_5, \#, A) & = & (q_3, A), \\
(10) & \delta(q_2, \#, A) & = & (q_3, A), & (14) & \delta(q_5, \lhd, A) & = & (q_0, A).
\end{array}$$

As a final observation, we point out that the block-counting performed in the first sweep and recorded in the pushdown ensures that each block $\#^i a^j$ in the input, with $i, j > 0$, does not vanish in the middle of the input string. In fact, if this happened, a block $\#^{i'} a^{j'}$, with $i' > i$, $j' > j$, occurring before $\#^i a^j$ would exist, thus implying that the input string does not belong to the language L_{sq}. □

Remark 1. We notice that the result in Theorem 2 may provide some inspiration on approaching an open question posed by Mráz and Otto in [9], concerning the possibility of accepting non-semilinear languages by non-returning deterministic finite state automata with translucent letters.

5. Comparing translucency and other classical paradigms

In order to assess the recognition power of DPDAwtl and nrDPDAwtl, we are now going to investigate the relation among the families $\mathscr{L}(\text{DPDAwtl})$ and $\mathscr{L}(\text{nrDPDAwtl})$ and other well-known language families. For a quick glance of the resulting scenario, the reader is referred to Fig. 1 on page 12.

Since our devices are equipped with a pushdown storage, it is natural to study the relationships with context-free languages. As an immediate observation, we have that the family $\mathscr{L}(\text{DPDAwtl})$ is a proper superset of the family of deterministic context-free languages (DCFL). In fact, a DPDAwtl without translucency relation is a classical deterministic pushdown automaton working with endmarker-terminated inputs; since the family DCFL is closed under right quotient by a singleton, the inclusion follows straightforwardly. In addition, Example 1 shows that such an inclusion is proper. This "recognition power lower bound" does not seem far from being tight, in the sense that there exist very simple languages, namely linear context-free ones, that do not belong to $\mathscr{L}(\text{nrDPDAwtl})$.

Lemma 4. *Let $L \subseteq \{ b^m \# b^n \mid m, n \geq 0 \}$ be a language where, for all $m, n \geq 0$, there exist $x \geq m$ and $y \geq n$ satisfying $b^x \# b^y \in L$. If L is accepted by some nrDPDAwtl, then at least one of the two languages L and L^R is a deterministic context-free language.*

Proof. Let $M = \langle Q, \{b, \#\}, \Gamma, q_0, \lhd, \bot, \tau, \delta \rangle$ be an nrDPDAwtl that accepts L. By Lemma 1, we assume that M accepts only after processing the input entirely. We consider accepting computations on inputs $b^m \# b^n$ with m, n large enough. Basically, M has to start the computation by reading some constant number of b's from the prefix b^m. If, in this phase, M reaches a state where b and # are translucent, then possible λ-transitions can be directly simulated and the transition on the endmarker can be simulated in the finite control of M. If this transition is not accepting, the input head returns to the left of the input, where it currently is, so its position does not change. Therefore, we can modify M so that it does not enter any state where b and # are translucent, as long as the input head stands at the beginning of the input string. Now, we distinguish between three cases:

- CASE 1: M does not reach a state where b is translucent while processing the prefix b^m, but it reaches a state where # is translucent exactly when it shows up in the input. In this case, the computation goes on processing the suffix b^n. In particular, while processing b^n, no state can be reached for which the symbol b is translucent. When M reaches the endmarker, only #'s are left in the input. So, one can effectively construct a classical DPDA M' from M that just simulates M. As soon as # appears in the input, M' reads and memorizes it, and when the endmarker becomes visible, M' simulates the remaining branch of the computation of M in its states.
- CASE 2: M does not reach a state where b is translucent while processing the prefix b^m, and sees # when it appears. Also in this case, one can effectively construct a classical DPDA M' from M that simulates M until # is processed. Now, the remaining input is b^n. So, M' can continue with the simulation. If, along this phase, b becomes translucent, M' knows that M sees the endmarker. Therefore, it can simulate the next step if M halts, or if M returns the input head and changes its state.

By inspecting the original nrDPDAwtl M, we can check whether or not the conditions for CASE 1 or CASE 2 are satisfied. In the affirmative, we can effectively construct from M a classical DPDA M' that accepts if and only if M accepts. Thus, we conclude that in these two cases L is a deterministic context-free language.

- CASE 3: M reaches a state where b becomes translucent after processing a constant number k_1 of b's from the prefix b^m. Also, from the previous considerations, we know that # is not translucent for this very same state. Thus, M reads and processes the #.
  - SUB-CASE 3.1: let us assume that M continues the computation by processing a constant number k_2 of b's and reaches again a state where b is translucent. Then, after reading the endmarker and returning to the leftmost input symbol, M sees only b's in the input. Again, from the modifications addressed at the beginning of this proof, we get that from now on the input head keeps staying onto the leftmost input position. So, the initial part of this computation phase looks like
$$\begin{aligned}(q_0 b^m \# b^n \lhd, \bot) \quad &\vdash^* (q_1 b^{m-k_1} \# b^n \lhd, \gamma_1) \quad \vdash^* (b^{m-k_1} q_2 b^n \lhd, \gamma_2) \\ &\vdash^* (b^{m-k_1} q_3 b^{n-k_2} \lhd, \gamma_3) \quad \vdash^* (q_4 b^{m-k_1+n-k_2} \lhd, \gamma_4) \\ &\vdash^* (q_5 b^{m-k_1} \lhd, \gamma_5).\end{aligned}$$
    We design a classical DPDA M' for L^R to behave as follows. M' starts with the state and pushdown content reached by M after processing k_1 many b's, plus processing # (this configuration can initially be created by M' through λ-transitions). Then, it directly simulates M until k_2 further b's are processed. At this point, M sees the endmarker and M' memorizes this fact as well as the return of M's input head. After processing the next $n - k_2$ many b's, i.e., the number of symbols M' sees before reaching #, it simulates the behavior of M on the endmarker. Next, M' processes #, reads k_1 many b's, and enters the state in which M would be in the same situation:
$$\begin{aligned}(q_2 b^n \# b^m \lhd, \gamma_2) \quad &\vdash^* (q_3 b^{n-k_2} \# b^m \lhd, \gamma_3) \quad \vdash^* (q_4' b^{n-k_2} \# b^m \lhd, \gamma_4) \\ &\vdash^* (q_5' \# b^m \lhd, \gamma_5) \vdash^* \quad \vdash^* (q_5 b^{m-k_1} \lhd, \gamma_5).\end{aligned}$$

Therefore, M' accepts an input string if and only if it belongs to L^R.

– SUB-CASE 3.2: let us assume that M processes # and then does not reach a state where b is translucent. This sub-case is treated similarly as the SUB-CASE 3.1. The initial part of an accepting computation looks like

$$(q_0 b^m \# b^n \lhd, \bot) \vdash^* (q_1 b^{m-k_1} \# b^n \lhd, \gamma_1) \vdash^* (b^{m-k_1} q_2 b^n \lhd, \gamma_2)$$
$$\vdash^* (b^{m-k_1} q_3 \lhd, \gamma_3) \vdash^* (q_4 b^{m-k_1} \lhd, \gamma_4).$$

We design a classical DPDA M' for L^R to behave as follows. On the #, we make M' behave as M on the endmarker, removing k_1 symbols from the input. The remaining computation reproduces that of M, where we know that the input head stands on the leftmost input position:

$$(q_2 b^n \# b^m \lhd, \gamma_2) \vdash^* (q_3 \# b^m \lhd, \gamma_3) \vdash^* (q_4' b^m \lhd, \gamma_4)$$
$$\vdash^* (q_4'' b^{m-k_1} \lhd, \gamma_4) \vdash^* (q_4 b^{m-k_1} \lhd, \gamma_4).$$

In conclusion, in CASE 3, a classical DPDA M' for L^R can effectively be constructed, thus showing that L^R is a deterministic context-free language. □

Remark 2. A precondition for Lemma 4 states that m and n must be unbounded at the same time. If both are bounded, we have a finite language, which is of course regular. If only one of them is bounded, then we have a unary language concatenated with a finite language. Lemma 5 below shows that each unary language in the family $\mathscr{L}$(nrDPDAwtl) is semilinear. This result can straightforwardly be generalized to also hold for the concatenation of a unary language and a finite language. Therefore, the precondition can actually be relaxed.

Now, we apply Lemma 4 to witness the existence of a context-free language that does not belong to $\mathscr{L}$(nrDPDAwtl):

Theorem 3. *The families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) are both incomparable with the family of context-free languages.*

Proof. Example 1 provides a non-context-free language accepted by a DPDAwtl. So, it remains to show the existence of a context-free language which is not accepted by any nrDPDAwtl. To this end, we consider the linear context-free language $L = \{ b^n \# b^n \mid n \geq 1 \} \cup \{ b^n \# b^{2n} \mid n \geq 1 \}$. In [18], it is proved that L is not a deterministic context-free language, either. Moreover, since its reversal can be written as $L^R = \{ b^n \# b^n \mid n \geq 1 \} \cup \{ b^n \# b^{\frac{n}{2}} \mid n \geq 2 \text{ even} \}$, the same argument in [18] yields that L^R is not a deterministic context-free language either. Therefore, by Lemma 4, we conclude that L cannot be accepted by any nrDPDAwtl. □

As a "recognition power upper bound" for our devices, we are able to spot the family of deterministic context-sensitive languages. In fact:

Proposition 3. *The family $\mathscr{L}$(nrDPDAwtl) is properly included in the family of deterministic context-sensitive languages.*

Proof. By Lemma 2 or Proposition 1, we can assume that the computations of an nrDPDAwtl take a number of steps which is linearly bounded by the input length. This, in turn, implies that the pushdown store cannot grow beyond linear space. Therefore, an nrDPDAwtl can be simulated in linear space by a deterministic one-tape one-head Turing machine, i.e., by a deterministic linear bounded automaton. So, the family $\mathscr{L}$(nrDPDAwtl) is included in the family of deterministic context-sensitive languages. Since $\mathscr{L}$(nrDPDAwtl) is incomparable with the family of context-free languages, this inclusion turns out to be proper. □

Let us now turn to another well-known language family. The so-called *growing context-sensitive grammars*, that is, context-sensitive grammars for which each production rule is strictly length-increasing, have been introduced in [19]. The induced language family GCSL lies strictly between CFL and DCSL. This family turns out to be interesting in our context, since each growing context-sensitive language can be accepted in polynomial time by some one-way auxiliary pushdown automaton with a logarithmic space bound [20].

Theorem 4. *The families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) are both incomparable with the family of growing context-sensitive languages.*

Proof. Due to Theorem 3, it suffices to show the existence of a language in $\mathscr{L}$(DPDAwtl), not belonging to GCSL. This can be achieved from [9, Example 2.3], where a DFAwtl is presented, that accepts a language which is not growing context-sensitive. Clearly, each DFAwtl can be simulated by a DPDAwtl, whence the result follows.

Here, to provide an alternative proof aiming to further explore the acceptance capabilities of DPDAwtl, we are going to explicitly devise another language in $\mathscr{L}$(DPDAwtl), not belonging to GCSL.

Let $h_1 \colon \{a, b\}^* \to \{a', b'\}^*$ and $h_2 \colon \{a, b\}^* \to \{a'', b''\}^*$ be homomorphisms defined as $h_1(a) = a'$, $h_2(a) = a''$ and $h_1(b) = b'$, $h_2(b) = b''$. We construct a DPDAwtl M whose accepted language $L(M)$ contains all the strings of the following language

L_{Gl} and, additionally, those strings where the initial block before the first occurrence of # possibly contains some primed symbols (the operator $\sqcup\!\sqcup$ denotes the shuffle operator (see, e.g., [16])):

$$L_{Gl} = \{\, w\#(h_1(w) \sqcup\!\sqcup h_2(w)^R \sqcup\!\sqcup \{\#\}) \mid w \in \{a,b\}^* \,\}.$$

The closure of GCSL under intersection with regular languages and non-erasing homomorphisms is shown in [21]. Therefore, if $L(M)$ belonged to GCSL, then so would

$$L(M) \cap (\{a,b\}^*\#\{a'',b''\}^*\#\{a',b'\}^*) = L_{Gl} \cap (\{a,b\}^*\#\{a'',b''\}^*\#\{a',b'\}^*).$$

In turn, by applying a non-erasing homomorphism that removes primed and double primed symbols, the language $L'_{Gl} = \{\, w\#w^R\#w \mid w \in \{a,b\}^* \,\}$ would belong to GCSL as well. However, by [22], the language L'_{Gl} cannot be generated by any context-sensitive grammar with $o(n^2)$ derivation steps, yielding that L'_{Gl} does not belong to GCSL. So, let us construct the claimed DPDAwtl M. We define

$$M = \langle Q, \{a,b,a',b',a'',b'',\#\}, \Gamma, q_0, \lhd, \bot, \tau, \delta \rangle,$$

where

- $Q = \{q_0, q_1, q_a, q_b, q_\#\}$,
- $\Gamma = \{a,b\}$,
- $\tau(q_0) = \tau(q_1) = \emptyset$, $\tau(q_a) = \tau(q_b) = \{a,b,a'',b'',\#\}$, $\tau(q_\#) = \{a',b',a'',b''\}$.

The transition functions δ is defined as follows, with $z \in \Gamma_\bot$:

$$\begin{array}{llcllcll}
(1) & \delta(q_0,a,z) & = & (q_a,az), & (6) & \delta(q_\#,\#,z) & = & (q_1,z),\\
(2) & \delta(q_0,b,z) & = & (q_b,bz), & (7) & \delta(q_1,a'',a) & = & (q_1,\lambda),\\
(3) & \delta(q_0,\#,z) & = & (q_\#,z), & (8) & \delta(q_1,b'',b) & = & (q_1,\lambda),\\
(4) & \delta(q_a,a',z) & = & (q_0,z), & (9) & \delta(q_1,\lhd,\bot) & = & \texttt{accept}.\\
(5) & \delta(q_b,b',z) & = & (q_0,z), & & & &
\end{array}$$

Let us give intuition for the behavior of M:

- In a first phase, M reads symbol by symbol the input prefix w, while pushing each symbol (transitions 1, 2). In addition, upon processing a symbol, M switches to the corresponding state q_a or q_b in which all but the corresponding single-primed symbols are invisible. The computation continues if and only if the first scanned single-primed symbol matches the symbol just read from the prefix (transitions 4, 5). This first phase ends when the first # shows up in the input. Basically, in this first phase, M verifies whether or not $w \in \{a,b\}^*$ and there exists a matching $h_1(w)$ shuffled in the suffix.
- The next short phase reads # from the input (transition 3) while entering the state $q_\#$ in which all but # symbols are invisible. The computation continues if and only if another # is found anywhere, that is, shuffled in the remaining input (transition 6). We emphasize that, in accepting computations, only double primed symbols survive in the input at this stage.
- Finally, the remaining input is read and matched against the pushdown content, in order to check whether $h_2(w)$ in the input matches w^R in the pushdown (transitions 7, 8, 9).

As an example, on input the word $abb\#b''a'b'\#b''a''b'\lhd \in L_{Gl}\lhd$, the accepting computation of M goes as follows:

$$\begin{array}{lll}
(q_0, abb\#b''a'b'\#b''a''b'\lhd, \bot) & \vdash (q_a, bb\#b''a'b'\#b''a''b'\lhd, a\bot) & \vdash (q_0, bb\#b''b'\#b''a''b'\lhd, a\bot)\\
& \vdash (q_b, b\#b''b'\#b''a''b'\lhd, ba\bot) & \vdash (q_0, b\#b''\#b''a''b'\lhd, ba\bot)\\
& \vdash (q_b, \#b''\#b''a''b'\lhd, bba\bot) & \vdash (q_0, \#b''\#b''a''\lhd, bba\bot)\\
& \vdash (q_\#, b''\#b''a''\lhd, bba\bot) & \vdash (q_1, b''b''a''\lhd, bba\bot)\\
& \vdash (q_1, b''a''\lhd, ba\bot) & \vdash (q_1, a''\lhd, a\bot)\\
& \vdash (q_1, \lhd, \bot) & \vdash \texttt{accept}.\ \square
\end{array}$$

Another interesting language family to be compared with $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) is the family of *Church-Rosser languages* (CRL), which has been introduced in [23]. The family CRL lies strictly between the deterministic context-free and the growing context-sensitive languages. Church-Rosser languages are defined *via* finite, confluent, and length-reducing Thue systems. The family is incomparable with context-free languages [20] and has some neat properties, e.g., Church-Rosser languages parse rapidly, in linear time, contain non-semilinear and inherently ambiguous languages [23], are characterized by deterministic automata models [20,24], and properly contain deterministic context-free languages as well as their reversals [23].

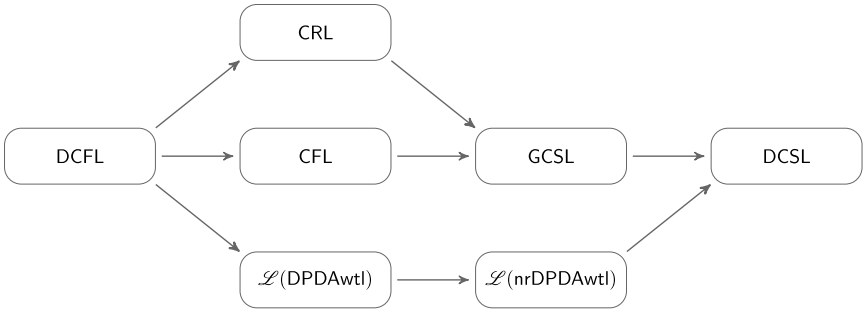

Fig. 1. Relationships between language families. An arrow between families indicates a strict inclusion. Any pair of families not connected by a path is incomparable.

In order to locate CRL with respect to $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl), it suits our goal to have a small digression on unary languages, i.e., languages built on single-letter alphabets. Though, by Theorem 2, the family $\mathscr{L}$(nrDPDAwtl) contains non-semilinear languages, the situation radically changes when the special case of unary languages is considered.

Lemma 5. *Each unary language in the family $\mathscr{L}$(nrDPDAwtl) is semilinear and, thus, regular.*

Proof. Let M be an nrDPDAwtl accepting the unary language $L(M) \subseteq \{a\}^*$. As an immediate observation, any non λ-transition from a state q with nonempty translucency, i.e., satisfying $\tau(q) = \{a\}$, necessarily sees the endmarker. A deterministic pushdown automaton M' can simulate M directly. Whenever such a transition on q is to be simulated, $\delta'(q, \lambda, z)$ is defined to be $\delta(q, \lhd, z)$. So, M' accepts $L(M)$ with endmarker that can be removed since the language family DCFL is closed under right quotient by a singleton. We conclude the unary language $L(M)$ is context-free and, thus, semilinear and regular. □

This enables us to show the following incomparability result for CRL:

Theorem 5. *The families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) are both incomparable with the family of Church-Rosser languages.*

Proof. According to [23], the family CRL is known to include non-semilinear unary languages, such as the language $\{a^{2^n} \mid n \geq 0\}$, which does not belong to $\mathscr{L}$(nrDPDAwtl) by Lemma 5. Conversely, by Theorem 4, there exists a language in $\mathscr{L}$(DPDAwtl) which is not even growing context-sensitive. □

For legibility, Fig. 1 on page 12 shows at a glance the situation concerning the recognition power of our translucent devices with respect to other well-known standard devices.

6. Closure properties

In the following, we refer to Example 3, according to which the language $L_{abc} = \{a^n b^n c^n \mid n \geq 0\}$ is not accepted by any DPDAwtl. Moreover, we use the fact that language $L = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$ can be accepted by a DPDAwtl, since in [10] it is shown that L is accepted by a deterministic finite automaton with translucent letters.

Proposition 4. *The language families $\mathscr{L}$(DPDAwtl) and $\mathscr{L}$(nrDPDAwtl) are closed under complementation, but they are neither closed under union nor under intersection. The language family $\mathscr{L}$(DPDAwtl) is, moreover, neither closed under union with regular languages nor under intersection with regular languages.*

Proof. For complementation, let M be a DPDAwtl (respectively, nrDPDAwtl) which, by Proposition 1, halts on any input by accepting or rejecting after the *whole* input has been processed. We build the DPDAwtl (respectively, nrDPDAwtl) M' for $\overline{L(M)}$ by obtaining the transition function δ' of M' from δ of M as follows: for any instruction of the form $\delta(q, \lhd, z) = \texttt{accept}$, we let $\delta'(q, \lhd, z)$ undefined; in addition, for any undefined instruction of the form $\delta(q, \lhd, z)$, we let $\delta'(q, \lhd, z) = \texttt{accept}$. Apart from this modification, δ' and τ' coincide with δ and τ, respectively.

As observed, the language $L = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$ is accepted by a DPDAwtl. Moreover, the regular language $a^*b^*c^*$ can be accepted by a DPDAwtl as well. We assume that $\mathscr{L}$(DPDAwtl) is closed under intersection with regular languages. Then, $L \cap a^*b^*c^* = L_{abc}$ belongs to $\mathscr{L}$(DPDAwtl), and this contradicts Example 3. Since $\mathscr{L}$(DPDAwtl) and REG are closed under complementation, it follows from the non-closure under intersection with regular languages that $\mathscr{L}$(DPDAwtl) is not closed under union with regular languages, either. Since REG $\subset \mathscr{L}$(DPDAwtl), we obtain that $\mathscr{L}$(DPDAwtl) is neither closed under union nor under intersection.

Finally, we use the language $\{ b^n\#b^n \mid n \geq 1 \} \cup \{ b^n\#b^{2n} \mid n \geq 1 \}$, which does not belong to $\mathscr{L}(\mathsf{nrDPDAwtl})$ by Theorem 3. Since both the languages $\{ b^n\#b^n \mid n \geq 1 \}$ and $\{ b^n\#b^{2n} \mid n \geq 1 \}$ are deterministic context-free, and hence they are both accepted by an nrDPDAwtl, we get the non-closure under union for $\mathscr{L}(\mathsf{nrDPDAwtl})$. Since $\mathscr{L}(\mathsf{nrDPDAwtl})$ is closed under complementation, then $\mathscr{L}(\mathsf{nrDPDAwtl})$ is not closed under intersection, either. □

7. Summary and future work

In this paper, we have considered deterministic pushdown automata with translucent input letters, in both returning and non-returning mode (DPDAwtl and nrDPDAwtl). Differently from the original definition of DPDAwtl provided by Otto [12], we have chosen to allow our devices to perform λ-transitions and accept by final states. This choice adheres to the standard definition of a deterministic pushdown automaton and enables a fairer comparison.

First of all, we have shown that nrDPDAwtl strictly outperform DPDAwtl, and that both families $\mathscr{L}(\mathsf{DPDAwtl}) \subset \mathscr{L}(\mathsf{nrDPDAwtl})$ are strictly framed within deterministic context-free and context-sensitive languages. For the non-returning model, we have proved that, differently from classical pushdown automata, $\mathscr{L}(\mathsf{nrDPDAwtl})$ contains non-semilinear languages too. On the other hand, as for classical pushdown automata, the unary version of $\mathscr{L}(\mathsf{nrDPDAwtl})$ contains only semilinear languages (and hence, it coincides with the family of unary regular languages).

Then, we have compared $\mathscr{L}(\mathsf{DPDAwtl})$ and $\mathscr{L}(\mathsf{nrDPDAwtl})$ with well-known language families, namely, context-free, growing context-sensitive and Church-Rosser languages. In all cases, we obtained incomparability results by suitably designing witness languages.

We have finally investigated closure properties of $\mathscr{L}(\mathsf{DPDAwtl})$ and $\mathscr{L}(\mathsf{nrDPDAwtl})$ under Boolean operations. Both families turn out to be closed under complementation, while neither are closed under union and intersection. Moreover, by restricting union and intersection to take place with regular languages, we still have non-closure for DPDAwtl, while the case of nrDPDAwtl still remains to be investigated.

Among possible future lines of research, we would like to emphasize the following.

A natural generalization of *deterministic pushdown automata with translucent input letters* is nondeterministic ones. As for the deterministic case, nondeterministic returning pushdown automata with translucent letters are defined formally in [12]. In this definition, again, the devices are designed to work in real time and to accept by empty pushdown. However, it is known that any context-free language is accepted by some nondeterministic pushdown automaton of this type, therefore, the computational power is not limited a priori as in the deterministic case. Another detail in the definition of [12] is that the computation halts immediately when the endmarker is seen. Here, we are more general by relaxing this condition. For example, the language $\{ vw\#u \mid vw \in \{a,b\}^*, |u|_a = |w|_a, |u|_b = |w|_b \}$ belongs to $\mathscr{L}(\mathsf{DPDAwtl})$ and it is a candidate for a language not being accepted by any nondeterministic pushdown automata with translucent input letters in the definition of [12].

In [13] various automata models are studied when they process their input as jumping automata. In particular, one-way jumping automata with a pushdown are considered. For example, it is shown that the language $\{ a^n b^n c^n \mid n \geq 0 \}$ is not accepted by any one-way jumping pushdown automaton. However, this language is accepted even by an nrDFAwtl. Therefore, one-way jumping deterministic pushdown automata are strictly weaker than nrDPDAwtl. An anonymous referee raised the interesting natural question: how do returning DPDAwtl compare to one-way jumping deterministic pushdown automata?

Another interesting suggestion has been provided by an anonymous DLT referee of [1], addressing natural algorithmic investigations. Precisely: the membership problem for languages in $\mathscr{L}(\mathsf{DPDAwtl})$ is certainly decidable in quadratic time as, on an input of length n, a DPDAwtl executes $O(n)$ transitions only (see Proposition 1). Is it possible to use the technique by Nagy and Kovács [25] to reduce this complexity to $O(n \cdot \log n)$? The same question could be re-stated for nrDPDAwtl as well: can the complexity of membership for languages in $\mathscr{L}(\mathsf{nrDPDAwtl})$ be reduced to $O(n \cdot \log^2 n)$, as for $\mathscr{L}(\mathsf{nrDFAwtl})$ (see [9])?

CRediT authorship contribution statement

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

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.

Acknowledgments

The authors wish to thank the anonymous referees for their valuable and helpful comments.

References

[1] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, P. Raucci, M. Wendlandt, Deterministic pushdown automata with translucent input letters, in: J. Day, F. Manea (Eds.), Developments in Language Theory (DLT 2024), in: LNCS, vol. 14791, Springer, 2024, pp. 203–217.
[2] A. Meduna, P. Zemek, Jumping finite automata, Int. J. Found. Comput. Sci. 23 (2012) 1555–1578.
[3] S. Beier, M. Holzer, Nondeterministic right one-way jumping finite automata, Inf. Comput. 284 (2022) 104687.
[4] H. Chigahara, S.Z. Fazekas, A. Yamamura, One-way jumping finite automata, Int. J. Found. Comput. Sci. 27 (2016) 391–405.
[5] S. Bensch, H. Bordihn, M. Holzer, M. Kutrib, On input-revolving deterministic and nondeterministic finite automata, Inf. Comput. 207 (2009) 1140–1155.
[6] P. Jančar, F. Mráz, M. Plátek, J. Vogel, Restarting automata, in: Fundamentals of Computation Theory, FCT, 1995, in: LNCS, vol. 965, Springer, 1995, pp. 283–292.
[7] F. Otto, Restarting Automata, Springer, Cham, Switzerland, 2025.
[8] B. Nagy, F. Otto, Finite-state acceptors with translucent letters, in: G. Bel-Enguix, V. Dahl, A. De La Puente (Eds.), International Workshop on AI Methods for Interdisciplinary Research in Language and Biology (BILC 2011), SciTePress, 2011, pp. 3–13.
[9] F. Mráz, F. Otto, Non-returning deterministic and nondeterministic finite automata with translucent letters, RAIRO Theor. Inform. Appl. 57 (2023) 8.
[10] B. Nagy, F. Otto, On CD-systems of stateless deterministic R-automata with window size one, J. Comput. Syst. Sci. 78 (2012) 780–806.
[11] B. Nagy, F. Otto, Deterministic pushdown-CD-systems of stateless deterministic R(1)-automata, Acta Inform. 50 (2013) 229–255.
[12] F. Otto, A survey on automata with translucent letters, in: B. Nagy (Ed.), Implementation and Application of Automata (CIAA 2023), in: LNCS, vol. 14151, Springer, 2023, pp. 21–50.
[13] S.Z. Fazekas, K. Hoshi, A. Yamamura, The effect of jumping modes on various automata models, Nat. Comput. 21 (2022) 17–30.
[14] V. Mitrana, A. Păun, M. Păun, J.R.S. Couso, Jump complexity of finite automata with translucent letters, Theor. Comput. Sci. 992 (2024) 114450.
[15] B. Nagy, F. Otto, CD-systems of stateless deterministic R(1)-automata governed by an external pushdown store, RAIRO Theor. Inform. Appl. 45 (2011) 413–448.
[16] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Massachusetts, 1979.
[17] S. Ginsburg, The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York, 1966.
[18] S. Ginsburg, S.A. Greibach, Deterministic context-free languages, Inf. Control 9 (1966) 620–648.
[19] E. Dahlhaus, M.K. Warmuth, Membership for growing context-sensitive grammars is polynomial, J. Comput. Syst. Sci. 33 (1986) 456–472.
[20] G. Buntrock, F. Otto, Growing context-sensitive languages and Church-Rosser languages, Inf. Comput. 141 (1998) 1–36.
[21] G. Buntrock, K. Loryś, On growing context-sensitive languages, in: International Colloquium on Automata, Languages and Programming (ICALP 2003), in: LNCS, vol. 623, Springer, 1992, pp. 77–88.
[22] A.V. Gladkii, On complexity of inference in phase-structure grammars, Algebra Log. Sem. 3 (1964) 29–44 (in Russian).
[23] R. McNaughton, P. Narendran, F. Otto, Church-Rosser Thue systems and formal languages, J. ACM 35 (1988) 324–344.
[24] G. Niemann, F. Otto, The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages, Inf. Comput. 197 (2005) 1–21.
[25] B. Nagy, L. Kovács, Finite automata with translucent letters applied in natural and formal language theory, in: Transactions on Computational Collective Intelligence XVII, in: LNCS, vol. 8790, Springer, 2014, pp. 107–127.