Revisiting Piecewise Testable Separability

Parosh Aziz Abdulla, Mohamed Faouzi Atig
Uppsala University, Sweden

Vrunda Dave, Shankara Narayanan Krishna
IIT Bombay, India

Abstract

The notion of separability has been well-studied for languages. Of these, piecewise testable separability of various language classes have been shown decidable recently. We investigate the piecewise testable separability problem for transducers, and show decidability for the classes of rational as well as unambiguous bounded sweeping transducers. For the class of two way transducers, we obtain decidability of separability with the positive fragment of piecewise testable transformations. We also look at the complexity of the positive piecewise testable separability for some language classes.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Formal languages and automata theory

Keywords and phrases Piecewise testable separability, transducers, decidability

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The separability problem for languages of a class $\mathcal{C}$ by a class $\mathcal{S}$ asks : given two languages $E, I \in \mathcal{C}$, does there exist a language $S \in \mathcal{S}$ separating E and I? That is, $E \subseteq S$ and $S \cap I = \emptyset$. The language S is called the separator of E, I. Separability is a classical problem of fundamental interest in theoretical computer science, and has recently received a lot of attention. For instance, the separability problem of regular languages has been studied wrt piecewise testable languages [19], [12], by levels in the quantifier hierarchy of first-order logic [21], and by first order definable languages [20]. Regular separability has been studied for one-counter automata [10], Parikh automata [8], well-structured transition systems [11] and so on. Going beyond regular languages, the decidability of piecewise testable separability and its characterization has been studied in [13]. Among the various separator classes considered in the literature, the class of piecewise testable languages (PTL) seems to be natural and tractable. PTLs can only reason about the order in which symbols appear in the language. A PTL over a finite alphabet Σ, is a finite Boolean combination of regular languages $\Sigma^* a_1 \Sigma^* a_2 \dots \Sigma^* a_n \Sigma^*$, where all $a_j \in \Sigma$. The tractability of PTL as a separator class comes from the fact that PTL separability of regular languages is in PTIME [19], [12].

In this paper, we focus on the piecewise testable class as the separator. Going beyond languages, we look at transformations, and investigate the separation by piecewise testable transducers (PTT). A PTT $\mathcal{T}$ is defined as a non-deterministic one way transducer s.t. $[\![\mathcal{T}]\!]$, the relations computed by $\mathcal{T}$, consists of *simple relations* $\Sigma^* \times \Gamma^* \cdot (a_1, a_1') \cdot \dots \Sigma^* \times \Gamma^* (a_k, a_k') \cdot \Sigma^* \times \Gamma^*$ where all $a_j \in \Sigma \cup \{\epsilon\}$ and all $a_j' \in \Gamma \cup \{\epsilon\}$, as well as their finite Boolean combinations. This implies that the domain $\mathsf{dom}(\mathcal{T})$ (the input language accepted by the underlying automaton of $\mathcal{T}$) is a PTL, and the set of outputs corresponding to $\mathsf{dom}(\mathcal{T})$ is also a PTL. Many fundamental results which are considered the pillars of formal language theory have been shown to extend seamlessly for transformations. To cite one among many, the seminal result of Büchi, Elgot and Trakhtenbrot, which establishes the equivalence between regular languages and MSO extends to transformations : [14] proves that MSO transductions are equivalent to regular transformations captured by two way transducers. Likewise, FO transductions are equivalent to first order definable transformations captured by aperiodic

two way transducers [7]. In this context, the separability problem which is well-studied for languages has not been investigated in the setting of transformations. The piece-wise testable separability problem for transducers (one-way, two-way) is a natural question which we study in this paper.

Contributions. We show the decidability of PTT separability for the class of rational transformations, which are transformations defined by a non-deterministic one-way transducer. Given rational transducers T_1, T_2, our proof proceeds by first encoding $[\![T_1]\!], [\![T_2]\!]$ as the languages accepted by ordered multipushdown automata (OMPA). This encoding preserves piecewise testable separability : we show that T_1, T_2 are PTT separable iff the encoded ordered multipushdown languages (OMPL) are PTL separable. The decidability of the PTL separability for OMPLs proceeds via a non-trivial language preserving reduction from OMPL to the languages accepted by higher order pushdown automata (HOPDA). Once we get this reduction, the decidability of PTL separability for OMPL follows, using the decidability for HOPDA languages [13]. Moving on, we look at the PTT separability of regular transformations captured by two way transducers. This is much more involved : an evidence pointing to this is a related problem which has remained open for long, viz., the FO-definability of regular transformations, which is considered a difficult problem [17]. We obtain decidability of PTT separability for the class of unambiguous sweeping transducers, where the number of sweeps are bounded. The general case remains open.

Given the difficulty of the general problem, we look at a restricted separability class of transformations where the domain and the computed outputs are defined by the negation free fragment of PTL (PosPTL) : these are obtained as negation free finite Boolean combinations (union and intersection, but no complementation) of regular languages $\Sigma^* a_1 \Sigma^* a_2 \dots \Sigma^* a_n \Sigma^*$, where all $a_i \in \Sigma$. Rational transducers $\mathcal{T}$ respecting this condition are called PosPTT. We look at PosPTT separability, and show that the problem is decidable for the full class of regular transformations. In this context, we also look at the decidability and complexity of PosPTL separability in the language setting. We show that the decidability of PosPTL separability for any class $\mathcal{C}$ of languages is remarkably simple : given two languages I, E from $\mathcal{C}$, it suffices to check the emptiness of $I{\uparrow} \cap E$ (or equivalently, $I \cap E{\downarrow}$) to decide the PosPTL separability of I and E. This is surprising considering the rather involved proof for PTL separability in [13]. We also look at some complexity questions related to PosPTL separability. The most involved one here is the PSPACE-completeness for CFLs. The corresponding question, viz., the complexity of PTL separability for CFLs is open [13].

2 Preliminaries

Let $[i, j]$ denote the set $\{i, i+1, \dots, j\}$ for $i, j \in \mathbb{N}$. Let Σ be a finite alphabet. Σ^* denotes the set of all finite words over Σ and Σ^+ denotes $\Sigma^* \backslash \{\epsilon\}$ where ϵ is the empty word. We denote $\Sigma \cup \{\epsilon\}$ by Σ_ϵ. For $u \in \Sigma^*$, length of u is denoted $|u|$ and the i^{th} symbol of u by $u[i]$.

Well-quasi orders. Given a (possibly infinite set) C, a quasi-order on C is a reflexive and transitive relation $\preceq \subseteq C \times C$. An infinite sequence $c_1, c_2, \dots$ in C is said to be saturating if there exists indices $i < j$ s.t. $c_i \preceq c_j$. A quasi-order $\preceq$ is said to be a well-quasi order (wqo) on C if every infinite sequence in C is saturating. The subword ordering $\preceq$ between words u, v over a finite alphabet Σ is a wqo on Σ^*.

Upward and Downward Closure. Given a wqo $\preceq$ on a set C, a set $U \subseteq C$ is said to be upward closed if for every $a \in U$ and $b \in C$, with $a \preceq b$, we have $b \in U$. The upward closure of a set $U \subseteq C$ is defined as $U{\uparrow} = \{b \in C \mid \exists a \in U, a \preceq b\}$. It is known that every upward closed set U can be characterized by a finite *minor*. A minor $M \subseteq U$ is s.t. (i)

for each $a \in U$, there is a $b \in M$ s.t. $b \preceq a$, and (ii) for all $a, b \in M$ s.t. $a \preceq b$, we have $a = b$. For an upward closed set U, let min be the function that returns the minor of U. Downward closures are defined analogously. The downward closure of a set $D \subseteq C$ is defined as $D \downarrow = \{b \in C \mid \exists a \in D, b \preceq a\}$.

The notion of subword relation and thus upward and downward closures naturally extends to n-tuple of words. The subword relation here is component wise i.e. $(u_1, \ldots, u_n) \preceq_n (v_1, \ldots, v_n)$ iff $u_i \preceq v_i$ for all $i \in [1, n]$.

Separability Problem for Language classes Given two classes of languages $\mathcal{C}$ and $\mathcal{S}$, we consider the separability problem for $\mathcal{C}$ by separator class $\mathcal{S}$. It is defined as following:

Input: Two languages I and E from the class $\mathcal{C}$
Question: Does there exist a separator $S \in \mathcal{S}$ s.t. $I \subseteq S$ and $E \cap S = \emptyset$?

Positive Piece-wise testable language (PosPTL) A piece-language is defined as $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_n \Sigma^*$ where $a_1, a_2, \ldots, a_n \in \Sigma$ for some $n \in \mathbb{N}$. A PosPTL is a finite negation free Boolean combination (except complementation) of piece languages. Note that PosPTLs are upward closed. Arbitrary finite Boolean combinations of piece languages are called piecewise testable languages (PTL). PTL class is as expressive as Boolean combinations of existential first order logic formulae.

Multitape (n-tape) automaton It is an extension of finite state automaton with multiple input tapes. Let Σ_i be the alphabet of tape i and $\tilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. An n-tape automaton is a tuple $A = (Q, \tilde{\Sigma}, \delta, Q_0, F)$ where Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times (\Sigma_1 \cup \{\epsilon\}) \times \cdots \times (\Sigma_n \cup \{\epsilon\}) \times Q$ is the transition relation. A run of A is a sequence $q_0, (a_{(1,1)}, \ldots, a_{(n,1)}), q_1, (a_{(1,2)}, \ldots, a_{(n,2)}), \ldots, (a_{(1,k)}, \ldots, a_{(n,k)}), q_k$ such that $q_0 \in Q_0$, $q_k \in F$ and $(q_i, (a_{(1,i+1)}, \ldots, a_{(n,i+1)}), q_{i+1}) \in \delta$ for all $i \in [0, k-1]$. An n-tuple word $(w_1, \ldots, w_n)$ is accepted if there is a run of A of the form $q_0, (a_{(1,1)}, \ldots, a_{(n,1)}), q_1, (a_{(1,2)}, \ldots, a_{(n,2)}), \ldots, (a_{(1,k)}, \ldots, a_{(n,k)}), q_k$ such that $w_i = a_{(i,1)} a_{(i,2)} \cdots a_{(i,k)}$. The language of A (denoted $\mathcal{L}(A)$) consists of all $n-$tuple words which are accepted by A.

Context Free Languages They are alternatively defined by Context Free Grammars (CFG) or Pushdown Automata(PDA). We recall both briefly as we are going to use them interchangeably.

A *Pushdown Automaton* is an extension of finite state automaton with a stack whose top can be updated on each transitions. A PDA is a tuple $P = (Q, \Sigma, \Gamma, \delta, Q_0, \bot, F)$ where Q, Q_0 and F are finite set of states, initial states and final states, respectively. Σ and Γ are finite input and stack alphabet respectively. $\bot \in \Gamma$ is a special symbol which will be used as the initial bottom symbol of stack. The transition relation is defined as $\delta \subseteq Q \times \Sigma_\epsilon \times \Gamma \times Q \times \text{Op}$ where Op represents set of operations on stack, it can be of the form push(A), pop(A) for any $A \in \Gamma \setminus \{\bot\}$ or *nop*. A configuration of the PDA is defined as (q, w, γ) where q is current state, w is the suffix of input word and input head points to the first symbol of w, and $\gamma \in ((\Gamma \setminus \{\bot\})^* \cdot \{\bot\})$ is the current stack content. The transition relation $\rightarrow_P$ is defined as follows: Given two configurations $(q, w, A\gamma)$ and (q', w', γ'), we have $(q, w, A\gamma) \rightarrow_P (q', w', \gamma')$ if and only if one of the following conditions hold: (1) $(q, a, A, q', push(B)) \in \delta$, then $w = aw'$ and $\gamma' = B \cdot A \cdot \gamma$, (2) $(q, a, A, q', pop(A)) \in \delta$, then $w = aw'$ and $\gamma' = \gamma$, or (3) $(q, a, A, q', nop) \in \delta$, then $w = aw'$ and $\gamma' = A \cdot \gamma$. A word $w \in \Sigma^*$ is accepted by P if there exists a sequence of configurations $c_1, \ldots, c_m$ such that: (1) c_1 is of the form $(q_0, w, \bot)$, with $q_0 \in Q_0$, (2) c_m is of the form $(q_f, \epsilon, \bot)$, with $q_f \in F$, and (3) $c_i \rightarrow_P c_{i+1}$ for all $i \in [1, m-1]$. The language of P, denoted by $\mathcal{L}(P)$ is defined as the set of words accepted by P.

A *Context Free Grammar* is defined by a tuple $G = (N, T, R, S)$ where N and T are finite set of non-terminals and terminals respectively, S is an initial non-terminal. R is a finite set of production rules of the form $(i) A \to BC$, $(ii) A \to a$, or $(iii) S \to \epsilon$ where $A \in N$, $B, C \in N \setminus \{S\}$ and $a \in T$. We define derivation relation $\Rightarrow$ as following: Given $u_1, u_2 \in (N \cup T)^*$, $u_1 \Rightarrow u_2$ iff there exists $A \to w$ in the production rules set R such that $u_1 = vAv'$ and $u_2 = vwv'$ for some $v, v' \in (N \cup T)^*$. Let $\Rightarrow^+$ denote one or more application of derivation relation $\Rightarrow$. A string w belongs to the language generated by grammar G iff it can be generated starting from S, applying derivation rules one or more times. Hence, the language generated by the grammar G is $\mathcal{L}(G) = \{w \in T^* \mid S \Rightarrow^+ w\}$.

Ordered Multi Pushdown Automata (OMPA) If we extend the model of PDA with multiple stacks, then we can push or pop in any stack. This model is known as Multi stack Pushdown Automata (MPA). MPA is Turing powerful, and many decision problems like emptiness are undecidable. We consider a subclass of MPA which regains decidability of emptiness. An n-ordered Multi Pushdown Automaton (OMPA or n-OMPA) $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, Q_0, F)$ is a MPA where Q, Σ, q_0, F are as defined as before for PDA. Γ is the stack alphabet and it contains the special symbol $\bot$. OMPA are restricted in a sense that pop is allowed only from the first non-empty stack. Formally, for each transition $(q, A_1, \ldots, A_n) \to^a (q', \gamma_1, \ldots, \gamma_n) \in \delta$, there is $j \in [1, n]$ such that $A_1 = \ldots = A_{j-1} = \bot$, $A_j \in \Gamma_\epsilon$, and $A_{j+1} = \ldots = A_n = \epsilon$; γ_i represents the symbols which are going to be pushed on the stack i, and A_j on top of stack j will be replaced by γ_j. $A_{j+1} = \ldots = A_n = \epsilon$ represents that the top of stacks $j+1, \ldots n$ are not accessible; input symbol a is from Σ_ϵ. Furthermore, we require as in [4] that: (1) If $A_1 \neq \bot$ then $|\gamma_1| \leq 2$ and $\gamma_j \in \Gamma_\epsilon \setminus \{\bot\}$ for all $j \in [2, n]$, (2) If $A_1 = \bot$ then $\gamma_1 \in \Gamma_\epsilon \setminus \{\bot\}$ and $\gamma_j = A_j$ for all $j \in [2, n]$ such that $A_j \in \{\bot, \epsilon\}$.

A configuration of $\mathcal{A}$ is of the form $(q, w, \alpha_1, \ldots, \alpha_n)$ where $q \in Q$ and $\alpha_1, \ldots, \alpha_n \in (\Gamma \setminus \{\bot\})^* \cdot \{\bot\}$. The transition relation $\to$ between the set of configurations of $\mathcal{A}$ is defined as follows: Given two configurations $(q, w, \alpha_1, \ldots, \alpha_n)$ and $(q', w', \alpha'_1, \ldots, \alpha'_n)$, we have $(q, w, \alpha_1, \ldots, \alpha_n) \to (q', w', \alpha'_1, \ldots, \alpha'_n)$ iff there is a transition $(q, A_1, \ldots, A_n) \to^a (q', \gamma_1, \ldots, \gamma_n) \in \delta$ such that $w = aw'$ and $\alpha'_i = \gamma_i u_i$ where $\alpha_i = A_i u_i$ for all $i \in [1, n]$.

A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if there exists a sequence of configurations $c_1, \ldots, c_m$ such that: (1) c_1 is of the form $(q_0, w, \bot, \ldots, \bot)$, with $q_0 \in Q_0$, (2) c_m is of the form $(q_f, \epsilon, \bot, \ldots, \bot)$, with $q_f \in F$, and (3) $c_i \to c_{i+1}$ for all $i \in [1, m-1]$. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$ is defined as the set of words accepted by $\mathcal{A}$. The languages accepted by an OMPA are referred to as OMPL.

3 Separability by Piecewise Testable Transducers

In this section, we look at the separability of rational and sweeping transducers by piecewise testable transducers. We define two way automata and transducers, and obtain rational and sweeping transducers as special cases.

Two-way Transducers. Two-way transducers extend two-way automata. Let Σ be a finite input alphabet and let $\vdash, \dashv$ be two special symbols not in Σ. We assume that every input string $w \in \Sigma^*$ is presented as $\vdash w \dashv$, where $\vdash, \dashv$ serve as left and right delimiters that appear nowhere else in w. We write $\Sigma_{\vdash\dashv} = \Sigma \cup \{\vdash, \dashv\}$. A two-way automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ has a finite set of states Q, subsets $I, F \subseteq Q$ of initial and final states and a transition relation $\delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q \times \{-1, 0, 1\}$. The -1 represents the reading head moving to the left, while a 1 represents the reading head moving to the right. 0 represents no movement. The reading head cannot move left when it is on $\vdash$, and cannot move right when it is on $\dashv$. A configuration of $\mathcal{A}$ on reading $\vdash w \dashv$ is represented by (q, i) where $q \in Q$ and i is a position

in the input, $0 \leq i \leq |w|+1$. Let $w' = \vdash w \dashv$. If $w' = w_1 a w_2$ and the present configuration is $(q, |w_1|)$, and $(q, a, q', -1) \in \delta$ (hence $a \neq \vdash$), then there is a transition from the configuration $(q, |w_1|)$ to $(q', |w_1| - 1)$. Likewise, if $(q, a, q', 1) \in \delta$ (resp. $(q, a, q', 0) \in \delta$), we obtain a transition from the configuration $(q, |w_1|)$ to $(q', |w_1| + 1)$ (resp. $(q', |w_1|)$). A run of $\mathcal{A}$ on reading $\vdash w \dashv$ is a sequence of transitions; it is accepting if it starts in a configuration $(p, 0)$ with $p \in I$ (the reading head on $\vdash$) and ends in a configuration $(q, |w| + 1)$ (the reading head on $\dashv$) with $q \in F$. The language $\mathcal{L}(\mathcal{A})$ or domain $\mathsf{dom}(\mathcal{A})$ of $\mathcal{A}$ is the set of all words $w \in \Sigma^*$ on which $\mathcal{A}$ has an accepting run.

To extend the definition of a two-way automaton $\mathcal{A}$ into a two-way transducer, $(Q, \Sigma, \delta, I, F)$ is extended to $(Q, \Sigma, \Gamma, \delta, I, F)$ by adding a finite output alphabet Γ and the definition of the transition relation as a *finite* subset $\delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q \times \Gamma^* \times \{-1, 0, 1\}$. The output produced on each transition is appended to the right of the output produced so far. $\mathcal{A}$ defines a relation $[\![\mathcal{A}]\!] = \{(u, w) \mid u \in \mathcal{L}(\mathcal{A})$ and w is the output produced on an accepting run of $u\}$. The acceptance condition is the same as in two-way automata. Two-way transducers are called 2NFTs (two way non-deterministic transducer). A two-way transducer is unambiguous (2NUFT) if each string $u \in \Sigma^*$ has at most one accepting run.

A 1NUFT represents a non-deterministic unambiguous transducer where the reading head only moves to the right. A *rational transducer* is a 1NFT (one way non-deterministic transducer) where the reading head only moves to the right or remains stationary. Initial and accepting configurations are defined as for 2NFTs. *n-Rational* transducers for $n > 1$, are those where the transition relation is defined as $\delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q \times [\Gamma^*]^n$ (obtained by extending rational transducers having a n-tuple of outputs, $n > 1$). Rational transducers correspond 1-*rational* transducers.

An *unambiguous sweeping transducer* is an unambiguous two-way transducer where the reading head changes directions only on $\vdash$ and $\dashv$. That is, the reading head keeps on moving right from $\vdash$ until it reads $\dashv$, at which point it reverses, and moves left, and continues moving left until it reads $\vdash$. The output produced on each transition, as above is appended to the output so far. The transducer makes a "right sweep" over its input each time it moves from $\vdash$ to $\dashv$, and a "left sweep" each time it moves from $\dashv$ to $\vdash$. When the number of sweeps is bounded by K, the transducer is called K-sweeping unambiguous transducer.

The class of transformations defined by a two-way transducer are called regular transformations (Reg). Likewise, the class of transformations defined by n-rational transducers and unambiguous K-sweeping transducers are called n-rational (n-Rat) and K-sweeping (K-Sweep) transformations respectively. When $n = 1$, we use Rat to denote 1-Rat. It is obvious that $\mathsf{dom}(\mathcal{A})$ is regular in all these cases.

Piecewise Testable Transducers (PTT). Let $\Sigma = \{a_1, \ldots, a_n\}$ and $\Gamma = \{a'_1, \ldots, a'_m\}$. Let $R_\Sigma = \{\Sigma^* a_{i_1} \Sigma^* \ldots \Sigma^* a_{i_k} \Sigma^* \mid$ all $a_{i_j} \in \Sigma\}$ and $R_\Gamma = \{\Gamma^* a'_{j_1} \Gamma^* \ldots \Gamma^* a'_{j_l} \Gamma^* \mid$ all $a'_{j_k} \in \Gamma\}$ be two sets containing regular expressions. A *simple relation* is any subset of $R_\Sigma \times R_\Gamma$. A PTT $\mathcal{T}$ is a rational transducer s.t. $[\![\mathcal{T}]\!]$ contains simple relations, as well as finite boolean combinations of simple relations. The class of transformations defined by piecewise testable transducers are called piecewise testable transformations (PTT). It can be seen that for a PTT $\mathcal{T}$, $\mathsf{dom}(\mathcal{T})$ and $\mathrm{im}(\mathcal{T})$ are PTLs over alphabets Σ and Γ respectively, with $\mathrm{im}(\mathcal{T}) = \{v \mid (u, v) \in [\![\mathcal{T}]\!]\}$.

We also consider the class of positive piecewise testable transducers (PosPTT). A PosPTT $\mathcal{T}$ is a rational transducer s.t. $[\![\mathcal{T}]\!]$ contains simple relations, as well as finite, negation free boolean combinations (unions and intersections, no complementation) of simple relations. In a PosPTT, $\mathsf{dom}(\mathcal{T})$ and $\mathrm{im}(\mathcal{T})$ are positive PTLs. Transformations defined by positive piecewise testable transducers are called positive piecewise testable transformations (PosPTT).

These definitions are extended in the straightforward manner to n-rational transducers.

The Piecewise Testable (PTT) Separability Problem for Transducers

Input: Two transducers T_1 and T_2 from class $\mathcal{R} \in \{n\text{−Rat}, n\text{−Sweep}\}$
Question: Does there exist $T_3 \in \mathsf{PTT}$ s.t. $[\![T_1]\!] \subseteq [\![T_3]\!]$ and $[\![T_2]\!] \cap [\![T_3]\!] = \emptyset$?

We show the problem decidable for $\mathcal{R} \in \{n\text{−Rat}, n\text{−Sweep}\}$, $n \in \mathbb{N}$. The problem is still open for Reg as well as sweeping transducers without a bound on the number of sweeps. In section 4, we obtain the decidability of PosPTT separability for Reg. We prove Theorem 3 below, for the class Rat. The case of n-Rat, $n > 1$ as well as n−Sweep are in Appendix A.

▶ **Example 1.** Consider the piecewise testable transformation $\{(abaa, babb)\}$ with $\Sigma = \Gamma = \{a, b\}$. The PTT T realizing this, that is, $[\![T]\!] = \{(abaa, babb)\}$ is obtained as a finite boolean combination of simple relations, starting with the simple relation $\mathcal{R}$ where $\mathcal{R} = (\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^* a \Sigma^*) \times (\Sigma^* b \Sigma^* a \Sigma^* b \Sigma^* b \Sigma^*)$. The relation which contains only $\{(abaa, babb)\}$ is equal to the upward closure of $(abaa, babb)$, but which does not contain a *bad pair*, that is, a pair with at least one component being a subsequence of length 5. An example of such a bad pair is $(aabaa, babb)$, since $(abaa, babb) \prec (aabaa, babb)$. The latter relation is captured by the simple relation $\mathcal{R}' = (\Sigma^* a \Sigma^* a \Sigma^* b \Sigma^* a \Sigma^* a \Sigma^*) \times (\Sigma^* b \Sigma^* a \Sigma^* b \Sigma^* b \Sigma^*)$. $[\![T]\!]$ is obtained as an intersection of $\mathcal{R}$ with the negation of all relations $\mathcal{R}'$ which characterize bad pairs.

▶ **Example 2.** As an example for PTT separability, consider the rational transformations $T_1 : a^i \to b^{i+1}$ and $T_2 : a^i \to (bc)^{i+1}$ for $i \geq 0$, with $\Sigma = \{a\}$ and $\Gamma = \{b, c\}$. It can be seen that the PTT T_3 s.t. $[\![T_3]\!] = (\Sigma^* \times \Gamma^* b\ \Gamma^*) \cap \neg[\Sigma^* \times \Gamma^* c\ \Gamma^*]$ is a separator for T_1, T_2.

▶ **Theorem 3.** *The* PTT *separability problem is decidable for* n-Rat *and* n-Sweep.

PTT separability of Rat. We first describe the idea behind the proof. Given a rational transducer T, we construct an ordered multipushdown automaton ($\mathsf{OMPA}(T)$) which accepts the language $\{u\#v \mid (u, v) \in [\![T]\!]\}$ with $\#$ being a fresh symbol (Lemma 4). Then, we show that if T is a PTT, then the language accepted by $\mathsf{OMPA}(T)$ is a PTL (Lemma 5). Thanks to this, the original problem, namely, the PTT separability problem of two rational transducers T_1, T_2 is reduced to the piecewise testable separability of ordered multipushdown languages $\mathsf{OMPA}(T_1), \mathsf{OMPA}(T_2)$. We prove the decidability of PTL separability for ordered multipushdown languages in Theorem 7, obtaining the result.

▶ **Lemma 4.** *Let* T *be a rational transducer. We can construct an ordered multi pushdown automaton* $\mathsf{OMPA}(T)$ *which accepts the language* $\{w_1\#w_2 \mid (w_1, w_2) \in [\![T]\!]\}$.

Proof. Let $T = (Q, \Sigma, \Gamma, \delta, I, F)$ be a rational transducer. The ordered multi pushdown automaton $\mathsf{OMPA}(T)$ is constructed as follows. The alphabet of $\mathsf{OMPA}(T)$ is $(\Sigma \cup \Gamma) \uplus \{\#\}$, where $\#$ is a fresh symbol; $\mathsf{OMPA}(T)$ has all states of T and two extra states q_{temp} and q_{fin}, with initial states same as those of T and final state q_{fin}. $\mathsf{OMPA}(T)$ has two stacks and mimics T as follows. The initial states of T and $\mathsf{OMPA}(T)$ are the same. For each transition (q, a, q', γ) in T, if the $\mathsf{OMPA}(T)$ is in state q, on reading a, it pushes the output from $\gamma \in \Gamma^*$ to stack 1, and moves to state q'. This continues as long as symbols from Σ are read. When T reaches an accepting state, $\mathsf{OMPA}(T)$, which is in the same state as T, either enters a new state q_{temp}, or continues mimicking T. If it enters state q_{temp}, then it reads only $\#\Gamma^*$. From q_{temp}, it reads $\#$, pops all symbols from stack 1, and pushes them onto stack 2. When the stack 1 becomes empty, it enters state q_{fin}, and reads only symbols of Γ. The contents of stack 2 (from top to bottom) represents the output generated by T on the input from Σ^* read so far. From q_{fin}, $\mathsf{OMPA}(T)$ reads symbols $\gamma \in \Gamma$, and pops the same γ from stack 2. When $\mathsf{OMPA}(T)$ finishes reading its input, stack 2 becomes empty and $\mathsf{OMPA}(T)$ accepts its input. It is easy to see that the language accepted by $\mathsf{OMPA}(T)$ is $\{w_1\#w_2 \mid (w_1, w_2) \in [\![T]\!]\}$. ◀

▶ **Lemma 5.** *Let T be a PTT. Then the language accepted by* $\mathsf{OMPA}(T)$ *is a piecewise testable language.*

Proof. We first show that for a PTT T, $[\![T]\!]$ can be defined as a union of the product of two PTLs. This is proved inductively. It is easy to see that simple relations are of the form $L_1 \times L_2$ where L_1, L_2 are PTLs over Σ, Γ respectively. Proceeding inductively, if the PTT T is an intersection of PTTs T_1, T_2, then $[\![T]\!] = [\![T_1]\!] \cap [\![T_2]\!] = [\bigcup(L_{11} \times L_{12})] \cap [\bigcup(L_{21} \times L_{22})]$. That is, $[\![T]\!] = \bigcup[(L_{11} \cap L_{21}) \times (L_{12} \cap L_{22})]$, where the union is over all possible combinations of PTLs arising from PTTs T_1 and T_2. If $[\![T]\!] = \neg([\![T']\!]) = \neg(\bigcup(L_1 \times L_2))$, then $[\![T]\!]$ can also be defined as $\bigcup[(\overline{L_1} \times L_2) \cup (L_1 \times \overline{L_2}) \cup (\overline{L_1} \times \overline{L_2})]$, where $\overline{L}$ represents the complement of the language. In all cases, we obtain $[\![T]\!]$ as the finite union of the product of PTLs.

By Lemma 4, $\mathsf{OMPA}(T) = \{w_1 \# w_2 \mid (w_1, w_2) \in [\![T]\!]\}$. Since $[\![T]\!]$ is a union of the product of PTLs, $\mathsf{OMPA}(T)$ can be written as $\bigcup[L_1 \cdot \# \cdot L_2]$ where L_1, L_2 are PTLs. Since $L_1 \cdot \# \cdot L_2$ is a PTL over the alphabet $\Sigma \cup \Gamma \cup \{\#\}$, so is $\mathsf{OMPA}(T)$. ◀

▶ **Lemma 6.** *The* PTT *separability problem for* Rat *is decidable.*

Proof. Let T_1, T_2 be two rational transducers, and let $\mathsf{OMPA}(T_1), \mathsf{OMPA}(T_2)$ respectively be the ordered multi pushdown automata constructed as in Lemma 4. We claim that T_1 and T_2 are separable by a PTT iff the languages $\mathsf{OMPA}(T_1), \mathsf{OMPA}(T_2)$ are separable by a PTL.

Assume that T_3 is a PTT separating T_1, T_2. Then we have $[\![T_1]\!] \subseteq [\![T_3]\!]$. By Lemma 4, it can be seen that for all $(w, w') \in [\![T_1]\!]$, $w \# w' \in \mathcal{L}(\mathsf{OMPA}(T_1))$, and, similarly for all $(w, w') \in [\![T_3]\!]$, $w \# w' \in \mathcal{L}(\mathsf{OMPA}(T_3))$. It is easy to see that $\mathcal{L}(\mathsf{OMPA}(T_1)) \subseteq \mathcal{L}(\mathsf{OMPA}(T_3))$. We also have $[\![T_2]\!] \cap [\![T_3]\!] = \emptyset$. In exactly the same way as above, we can show that $w \# w' \in \mathcal{L}(\mathsf{OMPA}(T_2))$ iff $w \# w' \notin \mathcal{L}(\mathsf{OMPA}(T_3))$. This shows that $\mathsf{OMPA}(T_1), \mathsf{OMPA}(T_2)$ are separable by the piecewise testable language $\mathsf{OMPA}(T_3)$.

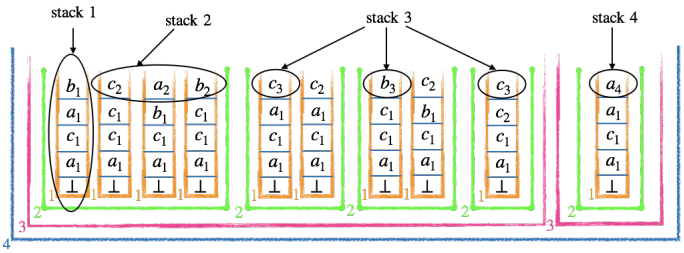
For the converse, assume that $\mathsf{OMPA}(T_1)$ and $\mathsf{OMPA}(T_2)$ are separable by a PTL L_3. Then $\mathcal{L}(\mathsf{OMPA}(T_1)) \subseteq L_3$ and $\mathcal{L}(\mathsf{OMPA}(T_2)) \cap L_3 = \emptyset$. Since L_3 is PTL, it is definable as a boolean combination of existential first-order logic formulae. Let φ be the formula s.t. $L_3 = \mathcal{L}(\varphi)$. Define $\varphi' = \varphi \wedge \exists x.\ \#(x) \wedge \neg \exists x\ \exists y.\ (\#(x) \wedge \#(y) \wedge x \neq y)$. φ' restricts the number of #s to be exactly 1 in $\mathcal{L}(\varphi)$. The PTL $\mathcal{L}(\varphi')$ is also a separator of $\mathsf{OMPA}(T_1)$ and $\mathsf{OMPA}(T_2)$, since $\mathcal{L}(\varphi') \subseteq \mathcal{L}(\varphi)$, $\mathcal{L}(\mathsf{OMPA}(T_2)) \cap \mathcal{L}(\varphi') = \emptyset$, and $\mathcal{L}(\mathsf{OMPA}(T_1)) \subseteq \mathcal{L}(\varphi')$. The last inclusion holds good since $\mathcal{L}(\mathsf{OMPA}(T_1)) \subseteq \mathcal{L}(\varphi)$, all words in $\mathcal{L}(\mathsf{OMPA}(T_1))$ have a unique #, and $\mathcal{L}(\varphi') = \{w \in \mathcal{L}(\varphi)$ s.t. the number of # in w is $1\}$.

To construct the PTT T_3 separating T_1, T_2, consider the DFA $\mathcal{A}$ s.t. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi') \subseteq \Sigma^* \# \Gamma^*$. We can construct a PTT T_3, whose state space consists of two copies of $\mathcal{A}$, with one copy having a flag 1. T_3 mimics the transitions of $\mathcal{A}$ by reading symbols of Σ till it reaches # producing no output. # is treated like $\dashv$. It then moves on to a state of $\mathcal{A}$ setting flag 1. From this state, it mimics transitions of $\mathcal{A}$ on Γ, remaining stationary and outputting the symbol from Γ that was read by $\mathcal{A}$. The final states of T_3 are same as that of $\mathcal{A}$ with the flag 1. It can be seen that $[\![T_3]\!] = \{(u, v) \mid u \# v \in \mathcal{L}(\varphi')\}$. To see why T_3 separates T_1, T_2, assume there exists $(u, v) \in [\![T_1]\!]$, s.t. $(u, v) \notin [\![T_3]\!]$. Then $u \# v \in \mathcal{L}(\mathsf{OMPA}(T_1))$ and $u \# v \notin \mathcal{L}(\mathsf{OMPA}(T_3))$. This gives the contradiction $u \# v \notin \mathcal{L}(\varphi')$. Similarly, if there exists $(u, v) \in [\![T_2]\!] \cap [\![T_3]\!]$, we obtain $u \# v \in \mathcal{L}(\mathsf{OMPA}(T_2)) \cap \mathcal{L}(\mathsf{OMPA}(T_3))$, which contradicts $\mathcal{L}(\mathsf{OMPA}(T_2)) \cap \mathcal{L}(\varphi') = \emptyset$ since $\mathcal{L}(\mathsf{OMPA}(T_3)) \subseteq \mathcal{L}(\varphi')$.

It remains to show the PTL separability of ordered multi pushdown languages (OMPL) to obtain the result. This is shown in Theorem 7. ◀

▶ **Theorem 7.** *PTL separability of OMPL is decidable.*

We give a sketch of the proof here, the details are in Appendix B. The key idea behind the proof is to simulate an n-OMPA by a higher order pushdown automata of order n (HOPDA). HOPDA are a generalization of PDA where we can have nested stacks: order-0 stack is a symbol from Γ, order-1 stack is a stack of order-0 stacks same as PDA, order-2 stack is a stack of order-1 stacks and so on.



By translating the transition rules of the OMPA appropriately to HOPDA transitions, we obtain an encoding of the stack contents of the OMPA in terms of the higher order stack contents of the HOPDA. For instance, the first stack of the OMPA can be obtained from the topmost order-1 stack of the HOPDA; the second stack of the OMPA can be obtained from the top elements of order-1 stacks which are below the topmost order-1 stack, and so on. A formal argument about the encoding can be found in Appendix B. The proof of Theorem 7 follows thanks to the encoding of the OMPA into HOPDA, and using the decidability of PTL separability for languages of HOPDA [13].

4 PosPTL and PosPTT Separability

Recall that we have defined PosPTL in Section 2 and PosPTT in Section 3. In this section, we first focus on the decidability and complexity of PosPTL separability for various language classes. The PTL separability for the class $\mathcal{C}$ of full trios has been investigated in [13], and the decidability of this problem has been shown equivalent to the decidability of the diagonal problem as well as the SUP problem of $\mathcal{C}$. While in [13], the proof of decidability goes through the sequence of these equivalences, we show that if we restrict to PosPTL, the decidability of PosPTL separability has a remarkably simple proof, for any class $\mathcal{C}$ of languages (full trios or otherwise!) as shown by Theorem 8. The rest of the section focuses on the complexity of PosPTL separability for various language classes $\mathcal{C}$. In this context, we look at a related question which has been left open in [13], viz., the complexity of PTL separability for context-free languages. We show here, via a non-trivial proof, that PosPTL separability of CFLs is PSPACE-complete. An interesting question to investigate is whether our proof can be extended to PTL separability.

The second part of the section shows the decidability of the PosPTT separability of two way transducers.

4.1 PosPTL Separability

Let $\mathcal{C}$ be a class of languages. We prove the necessary and sufficient conditions for PosPTL separability of class $\mathcal{C}$ as follows.

▶ **Theorem 8.** *Two languages I and E from a class $\mathcal{C}$ are* PosPTL *separable iff $I{\uparrow} \cap E = \emptyset$ iff $I \cap E{\downarrow} = \emptyset$.*

Proof. We first prove that languages I and E from $\mathcal{C}$ are PosPTL separable iff $I{\uparrow} \cap E = \emptyset$. Let I, E be languages over a finite alphabet Σ. Assume I and E are separable by the PosPTL S. Then $I \subseteq S$ and $S \cap E = \emptyset$. Since S is upward closed, $I{\uparrow} \subseteq S$ and hence $I{\uparrow} \cap E = \emptyset$.

Assume $I{\uparrow} \cap E = \emptyset$. We claim that $I{\uparrow}$ is a separator. $I \subseteq I{\uparrow}$ and $I{\uparrow} \cap E = \emptyset$. Indeed $I{\uparrow} \in \mathsf{PosPTL} : \mathsf{min}(I{\uparrow})$ is a finite set since the subword ordering $\preceq$ is a wqo on Σ^*. Let $\mathsf{min}(I{\uparrow}) = \{w_1, \ldots, w_n\}$. Then $I{\uparrow}$ is a finite union of $\{w_i\}{\uparrow}$ for $i \in \{1, 2, \ldots, n\}$. For $w_i = a_{i_1} \ldots a_{i_m}$, with $a_{i_j} \in \Sigma$, $\{w_i\}{\uparrow}$ is the piece language $\Sigma^* a_{i_1} \Sigma^* \ldots \Sigma^* a_{i_m} \Sigma^*$. Thus, $I{\uparrow}$ is indeed in PosPTL, being the finite union of piece languages.

To see $I{\uparrow} \cap E = \emptyset$ iff $I \cap E{\downarrow} = \emptyset$. Let $w \in I{\uparrow} \cap E$. Let $w' \preceq w$ be s.t. $w' \in I$. Then $w' \in I \cap E{\downarrow}$. The converse works similarly. ◀

We can restate Theorem 8 using downward closed languages as separators in place of PosPTL. One such example of a separator class is simple regular expressions defined in [1].

▶ **Corollary 9.** PosPTL *separability of regular languages is in* PTime.

▶ **Theorem 10.** PosPTL *separability of the languages of multitape automata is* NP-complete.

Proof. Let A and B be two n-tape automata over $\widetilde{\Sigma} = (\Sigma_1 \times \cdots \times \Sigma_n)$. We assume w.l.o.g. that all the transitions of B are of the form $(q, (a_1, \ldots, a_n), q')$ with $|a_1 \cdots a_n| = 1$. We show that deciding $\mathcal{L}(A){\uparrow} \cap \mathcal{L}(B) \neq \emptyset$ is NP-complete. The idea is to guess a minimal n-tuple word $\widetilde{w} = (w_1, \ldots, w_n) \in \mathsf{min}(\mathcal{L}(A))$ (which can be accepted by a run of A that visits any state of A at most once) and then check whether $\{\widetilde{w}\}{\uparrow} \cap \mathcal{L}(B) \neq \emptyset$. To do that we first guess a sequence $s = (a_{(1,1)}, \ldots, a_{(n,1)}), \ldots, (a_{(1,k)}, \ldots, a_{(n,k)})$ such that (1) $|a_{(1,j)}, \ldots, a_{(n,j)}| = 1$ for all $j \in [1, k]$ and (2) $w_i = a_{(i,1)} a_{(i,2)} \cdots a_{(i,k)}$ for all $i \in [1, n]$. Then, we treat the n-tape automaton B as a finite state automaton over the alphabet $(\Sigma_1 \cup \{\epsilon\}) \times \cdots \times (\Sigma_n \cup \{\epsilon\})$ and check whether $\{s\}{\uparrow} \cap \mathcal{L}(B) \neq \emptyset$.

Next, we reduce the shortest common supersequence (SCP) problem which is known to be NP-complete [22], to our problem. SCP is defined as follows: Given $m > 2$ strings $S_1, S_2, \ldots, S_m$ and an integer k, decide whether there exists a string S such that $|S| \leq k$ and $S_i \preceq S$ for all i. For that, we construct m-tape automata A and B s.t. $\mathcal{L}(A) = \{(S_1, S_2, \ldots, S_m)\}$ and $\mathcal{L}(B) = \{(w, w, \ldots, w) \mid |w| \leq k\}$. Both of these automata can be constructed in polynomial time. It is indeed the case that a common supersequence of length $\leq k$ exists iff $\mathcal{L}(A){\uparrow} \cap \mathcal{L}(B) \neq \emptyset$. ◀

▶ **Theorem 11.** PosPTL *separability of context free languages is* PSPACE − complete.

Proof. Given context free grammars G_1, G_2, $\mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2) \neq \emptyset$ iff $\mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2){\downarrow} \neq \emptyset$. If there is a witness of $\mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2)$, the same witness trivially works for $\mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2){\downarrow}$. Now suppose $w \in \mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2){\downarrow}$, then there exists $w' \in \mathcal{L}(G_2)$ s.t. $w \preceq w'$. Also, $w' \in \mathcal{L}(G_1){\uparrow}$. Hence, we have a witness of the non-emptiness of $\mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2)$. We prove that the non-emptiness of $\mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2){\downarrow}$ is PSPACE − complete.

PSPACE-membership. The non-emptiness check of $\mathcal{L}(G_1){\uparrow} \cap \mathcal{L}(G_2){\downarrow}$ can be seen to be in PSPACE as follows. We construct PDAs $A_{G_1{\uparrow}}$, $A_{G_2{\downarrow}}$ respectively for $\mathcal{L}(G_1){\uparrow}$ and $\mathcal{L}(G_2){\downarrow}$. This construction takes polynomial time; moreover, $A_{G_1{\uparrow}}$, $A_{G_2{\downarrow}}$ use only a bounded stack which is polynomial in the size of the CFG (G_1 or G_2). This fact is rather easy to see for $A_{G_1{\uparrow}}$ (Lemma 12), but is quite involved for $A_{G_2{\downarrow}}$ (Lemma 13). Assuming the construction of $A_{G_1{\uparrow}}$, $A_{G_2{\downarrow}}$, we give an NPSPACE algorithm as follows. Guess a word w, one symbol at a time, and run $A_{G_1{\uparrow}}$, $A_{G_2{\downarrow}}$ in parallel. Since both the PDAs require only a polynomially bounded stack size, we need polynomial space to store the information pertaining to the states, stacks and the current input symbol. If $A_{G_1{\uparrow}}$, $A_{G_2{\downarrow}}$ accept w, we are done. The PSPACE-membership follows from Savitch's theorem.

PSPACE-hardness. We reduce the halting problem of a Linear Bounded Turing Machine (LBTM) to our problem. Given a LBTM M and a word w, we construct PDAs P_1 and P_2 such that $\mathcal{L}(P_1){\uparrow} \cap \mathcal{L}(P_2){\downarrow} \neq \emptyset$ iff M accepts w. Our construction is in Appendix C.1. ◀

Bounded Stack Size of $A_{G\uparrow}$, $A_{G\downarrow}$

We first consider the case of $A_{G\uparrow}$. The proof follows by examining the height of derivation trees of words in $\mathsf{min}(\mathcal{L}(G))$, and showing that they have a bounded height. This bound gives the height of the stack size in $A_{G\uparrow}$. The proof of Lemma 12 is in Appendix C.4.

▶ **Lemma 12.** *Given a CFG $G = (N, T, R, S)$, the PDA $A_{G\uparrow}$ can be constructed in polynomial time, using a stack whose height is polynomial in the size of G. $\mathcal{L}(A_{G\uparrow}) = \mathcal{L}(G)\uparrow$.*

▶ **Lemma 13.** *Given CFG $G = (N, T, R, S)$, one can construct PDA $A_{G\downarrow}$ in polynomial time which uses polynomially bounded stack for computation such that $\mathcal{L}(A_{G\downarrow}) = \mathcal{L}(G)\downarrow$.*

Proof. First, we compute a CFG G' using G, and then construct $A_{G\downarrow}$ using G, G'. The construction of G' is described below. We recall the approach described in [9] to compute downward closure, where a regular expression is computed for $\mathcal{L}(G)\downarrow$.

For every language L, let $\alpha(L)$ be the set of terminal symbols occurring in L (hence $\alpha(L) = \emptyset$ iff $L \subseteq \{\epsilon\}$). For $L, L' \subseteq T^*$ we have: (i) $\alpha(L \cup L') = \alpha(LL') = \alpha(L) \cup \alpha(L')$, (ii) $(L \cup L')\downarrow = L\downarrow \cup L'\downarrow$, and (iii) $(LL')\downarrow = L\downarrow\, L'\downarrow$. For $m \in (N \cup T)^*$, let $L(G, m)$ denote the language generated by G, starting from the word m. For every $A, B \in N$, define:

1. $B <_1 A$ iff $A \Rightarrow_G^+ mBm'$ for some $m, m' \in (N \cup T)^*$,
2. $B <_2 A$ iff $A \Rightarrow_G^+ mBm'Bm''$ for some $m, m', m'' \in (N \cup T)^*$,
3. $B =_1 A$ iff $B <_1 A <_1 B$.

A proof of the first implication in Claim 14 is in Appendix C.2, the second implication has a similar proof.

▷ Claim 14. $A <_2 A \Rightarrow L(G, A)\downarrow = (\alpha(L(G, A)))^*$, $A =_1 B \Rightarrow L(G, A)\downarrow = L(G, B)\downarrow$

Computing $L(G, A)\downarrow$. Now, we explain how $L(G, A)\downarrow$ can be computed for any given $A \in N$. If $A <_2 A$, then claim 14 yields the answer. Otherwise, we compute $L(G, A)\downarrow$ in terms of $L(G, B)\downarrow$ for $B <_1 A$ and $B \neq_1 A$, assuming $L(G, B)\downarrow$ is given by previously computed rules for $L(G, B)$. Let $p : A \to m$ be a production rule. We define the words $R_0(p), R_1(p)$ and $R_2(p)$ as follows, depending on the production rule $p : A \to m$.

First case: m does not contain any non-terminal $B =_1 A$. We let $R_0(p) := m$, and $R_1(p)$ and $R_2(p)$ be the empty words.

Second case: m contains a unique non-terminal B with $B =_1 A$ and $m = m'Bm''$. We let $R_1(p) := m'$ and $R_2(p) := m''$. Since we assume that $A \not<_2 A$, the word m cannot contain two occurrences of non-terminals $=_1 A$. In this case, $R_0(p)$ is the empty word.

▷ Claim 15. For every A such that $A \not<_2 A$, we have:

$$L(G, A)\downarrow = (\cup\, \alpha(L(G, R_1(p))))^*\ (\cup L(G, R_0(p))\downarrow)\ (\cup\, \alpha(L(G, R_2(p))))^*$$

where the union extend to all production rules p with lefthand side B such that $B =_1 A$.

Since the words $R_0(p), R_1(p)$ and $R_2(p)$ contain only non-terminals C with $C <_1 A$ and $C \neq_1 A$, we have achieved our goal. By this we end the brief recall of the the approach described in [9] to compute downward closure.

Based on the above facts, we can construct intermediate CFG $G' = (N', T', R', S\downarrow)$, which helps to construct PDA for $\mathcal{L}(G)\downarrow$. For each non-terminal $A \in N$, we introduce five non-terminals $A\downarrow, A_l, A_m, A_r$ and A_{alph} as well as dummy terminals a_l, a_r and a_{alph} in G'. Construction of G' is shown in Algorithm 1 in Appendix C.3. The non-terminal A_{alph} as well as the terminal a_{alph} are used to simulate $\alpha(L(G, A))$. Likewise, the non-terminal A_l (A_r)

and the terminal a_l (a_r) are used to simulate $\alpha(L(G,m'))$ ($\alpha(L(G,m''))$) when $m = m'Cm''$ in the production $A \to m$ and $C =_1 A$. The non-terminal A_m is used when $m = A_1A_2$ for non-terminals $A_1 \neq_1 A, A_2 \neq_1 A$. Finally, $A{\downarrow}$ works in place of A, and depending on m in the production $A \to m$, gets rewritten either as A_{alph} or $A_l A_m A_r$ or $A_1{\downarrow}A_2{\downarrow}$.

Constructing the PDA $A_{G\downarrow}$ from G, G'. The PDA $A_{G\downarrow}{=}(Q, T, N', \delta, \{q\}, S{\downarrow}, \{q\})$ from G' and G is constructed below. This works in the standard way of converting CFGs to PDA, keeping in mind the following. When a_l (resp. a_r, a_{alph}) is the top of the stack, the PDA has a loop over symbols from $\alpha_l(A)$ (resp. $\alpha_r(A)$, $\alpha(L(G,A))$), in whichever state it is, at that time. $\alpha_l(A)$ (resp. $\alpha_r(A)$) is a set of symbols appearing in $\alpha(L(G,m'))$ (resp $\alpha(L(G,m''))$) for all rules $C \to m'Bm''$ where $C =_1 A =_1 B$. The PDA stays in state q when the top of stack contains a non-terminal A. This non-terminal is popped and the right hand side m of its production rule $A \to m$ is pushed on the stack. If the top of the stack is i) a terminal from T which also happens to be the next input symbol, then we just pop it ii) a terminal of the form a_{alph}, a_l or a_r, we move to state $q_{A_{\text{alph}}}$, q_{A_l} or q_{A_r} respectively and loop over $\alpha(L(G,A))$, $\alpha_l(A)$ or $\alpha_r(A)$ respectively and come back to q non-deterministically. The PDA accepts in q when the stack becomes empty. A_l, A_r and A_{alph} do not produce any further non-terminals while A_m produces non-terminals B where $B <_1 A$ and $B \neq_1 A$. Thus, the stack size is $\leq \mathcal{O}(|N|)$. ◀

We conclude this section with the decidability of PosPTL separability of HOPDA. It is enough to establish the decidability of emptiness of $\mathcal{L}(A) \cap \mathcal{L}(B){\downarrow}$ for HOPDA A and B. It is known[15] that $\mathcal{L}(B){\downarrow}$ is computable, and is a regular language. Hence, checking the emptiness of $\mathcal{L}(A) \cap \mathcal{L}(B){\downarrow}$ is decidable due to decidability of emptiness of HOPDA [18]. Corollary 17 follows from Lemma 23.

▶ **Theorem 16.** PosPTL *separability of languages accepted by HOPDA is decidable.*

▶ **Corollary 17.** PosPTL *separability of languages accepted by OMPA is decidable.*

4.2 PosPTT Separability of 2NFT

For ease of presentation in this section, we work with 2NFTs where the reading head has only directions +1, -1. The direction 0 can be replaced by two successive transitions, one which moves +1 and the other -1. We define here the notion of *crossing sequences* [5], which will be used in the proof of Lemma 18. Let $w{=}\vdash a_1 \dots a_n \dashv$ be an input word and let ρ be a run of the 2NFT on w. A state is associated to each position and a direction (+1 or -1) in the run. For example, at position i, in state q, the run could proceed to position $i+1$, or to position $i-1$. Thus, the state q is associated to the position i and direction +1, or to the state q and direction -1. Say that the transducer is in state q at $\rho(i,j)$ if it is in state q when visiting position i for the $(j+1)$st time. For a run ρ, the crossing sequence at a position x is a tuple $(q_0, q_1, \dots, q_h)$ where the q_y's are all states at $\rho(x,y)$. On each successful run ρ, the crossing sequence has an odd length since the whole word is read in an accepting run, and hence the last transition on each position is from left to right.

Consider *normalized* runs, where the transducer never visits the same position, in the same state s.t. both visits are a result of a right to left transition, or both visits are a result of a left to right transition. Any accepting run ρ can be normalized : a run which is not normalized will have two visits of the same position i of the word in the same state as a result of say, a left to right transition. Then, after the first visit of position i from position $i-1$ in state q, the transducer has explored some positions till its second visit to position i from position $i-1$ in state q. We can delete this explored part in between, obtaining again,

an accepting run. In every normalized accepting run, the crossing sequences have a length $\leq 2|Q|-1$, and hence the number of crossing sequences is exponential in $|Q|$.

Given a 2NFT $\mathcal{T}$, $\mathsf{min}(\llbracket\mathcal{T}\rrbracket)$ represents set of minimal pairs (u,v) in $\mathcal{T}$. That is, $\mathsf{min}(\llbracket\mathcal{T}\rrbracket) = \{(u,v) \mid \neg\exists(u',v') \prec (u,v) \wedge (u',v') \in \llbracket\mathcal{T}\rrbracket\}$, with $\prec$ applied component wise : $(u',v') \prec (u,v)$ iff $u' \prec u$ or $(u' = u$ and $v \prec v')$.

▶ **Lemma 18.** *Let $\mathcal{T}$ be a 2NFT. If $(u,v) \in \mathsf{min}(\llbracket\mathcal{T}\rrbracket)$, then $|u| \leq \mathsf{in}_{\max} = |Q|^{2|Q|-1}$ and $|v| \leq \mathsf{out}_{\max} = (2|Q|-1)\cdot\gamma_{\max}\cdot|Q|^{2|Q|-1}$, where $\gamma_{\max}$ represents the maximum length of an output on any transition in $\mathcal{T}$.*

Proof. Assume $(u,v) \in \mathsf{min}(\llbracket\mathcal{T}\rrbracket)$. If $|u| > \mathsf{in}_{\max}$, then there exists a crossing sequence which is repeated in the accepting run of u. By deleting the factor between two occurrences of this crossing sequence, we obtain an accepting run over a word u', which is a strict subword of u, and whose output v' is also a subword (may not be strict) of v, a contradiction to $(u,v) \in \mathsf{min}(\llbracket\mathcal{T}\rrbracket)$. If $u \in \mathsf{min}(\mathsf{dom}(\mathcal{T}))$ and $|v| > \mathsf{out}_{\max}$ for $v = \llbracket\mathcal{T}\rrbracket(u)$, then, since $|u| \leq \mathsf{in}_{\max}$, and at each position of u, at most $(2|Q|-1)\gamma_{\max}$ symbols can be produced. This way $|v| \leq (2|Q|-1).\gamma_{\max}\cdot|u|$, a contradiction to $|v| > \mathsf{out}_{\max}$. ◀

▶ **Theorem 19.** PosPTT *separability of 2NFT is decidable.*

Proof. We first show that Theorem 8 can be extended to sets of relations i.e. given two sets of relations R_1 and R_2, they are PosPTT separable iff $R_1{\uparrow} \cap R_2 = \emptyset$ (see Lemma 24 in Appendix D for details). Thanks to Lemma 24, our problem of deciding PosPTT separability of 2NFTs T_1 and T_2 reduces to deciding emptiness of $T_1{\uparrow} \cap T_2$. As proved in Lemma 18, the length of u,v in $(u,v) \in \mathsf{min}(\llbracket T_1\rrbracket)$ and $\mathsf{min}(\llbracket T_2\rrbracket)$ are exponentially bounded above by the sizes of T_1, T_2. Hence $T_1{\uparrow}$ can be represented as $\bigcup_{(u_i,v_i)\in\mathsf{min}(T_1)} u_i{\uparrow} \times v_i{\uparrow}$. The emptiness of $T_1{\uparrow} \cap T_2$ now reduces to $\bigcup_i (L_i \times L_i') \cap T_2$ where L_i and L_i' are regular languages. Let A_i, B_i respectively be DFA s.t. $\mathcal{L}(A_i) = L_i$ and $\mathcal{L}(B_i) = L_i'$. For each i in the union, we construct a two way automaton as follows and check its emptiness. To test the emptiness of $(A_1 \times B_1) \cap T_2$, the two way automaton first simulates A_1 in the input, and if accepted, goes back to the beginning of the input, and simulates T_2 and B_1 in parallel. In other words, first we test whether the input word is in A_1, and then we check whether it is accepted by T_2 (in $\mathsf{dom}(T_2)$) and if its output wrt T_2 is accepted in B_1. Since the emptiness of two way automaton is decidable, the emptiness of $T_1{\uparrow} \cap T_2$ is also decidable. Hence, the separability of 2NFT by PosPTT is decidable. ◀

We conclude by illustrating an example wrt PTT and PosPTT separability. Consider transformations $T_1 : a^i \to b^{i+1}c$ which replaces all as with b and adds an extra bc in the output. Let transformation T_2 be $a^i \to c^{i+1}b$. Let $\Sigma = \{a\}$, $\Gamma = \{b,c\}$ and $i \geq 0$ in both T_1, T_2. T_1 and T_2 are PosPTT separable since $T_1{\uparrow} \cap T_2 = \emptyset$ (Lemma 24). The PosPTT separator for T_1, T_2 is $T_1{\uparrow} = \Sigma^* \times \Gamma^* b \Gamma^* c \Gamma^*$. This also implies that T_1 and T_2 are PTT separable. Note however that, PTT separable transformations given in example 2 are not PosPTT separable, since $T_1{\uparrow} = \Sigma^* \times \Gamma^* b \Gamma^*$ and $T_1{\uparrow} \cap T_2 \neq \emptyset$.

We sign off with a brief remark on PosPTT separability. As seen from the above example, it is easy to check the emptiness of $T_1{\uparrow} \cap T_2$ obtaining a PosPTT separator whenever one such exists, while deciding PTT separability is much more involved : we encode relations of T_1, T_2 as OMPL and check their PTL separability. As mentioned in [13], an "effective algorithm" to check this (and hence PTT separability) will involve invoking in parallel, positive and negative semi-procedures for PTL separability.

References

1 Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004. URL: https://doi.org/10.1023/B:FORM.0000033962.51898.1a, doi:10.1023/B:FORM.0000033962.51898.1a.

2 R. Alur and P. Černý. Expressiveness of streaming string transducers. In *FSTTCS*, volume 8, pages 1–12, 2010.

3 R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL*, pages 599–610, 2011.

4 Mohamed Faouzi Atig. Model-checking of ordered multi-pushdown automata. *Logical Methods in Computer Science*, 8(3), 2012. URL: https://doi.org/10.2168/LMCS-8(3:20)2012, doi:10.2168/LMCS-8(3:20)2012.

5 Félix Baschenis. *Minimizing resources for regular word transductions. (Gestion de ressources des transductions régulières sur les mots)*. PhD thesis, University of Bordeaux, France, 2017. URL: https://tel.archives-ouvertes.fr/tel-01680357.

6 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Minimizing resources of sweeping and streaming string transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 114:1–114:14, 2016. URL: https://doi.org/10.4230/LIPIcs.ICALP.2016.114, doi:10.4230/LIPIcs.ICALP.2016.114.

7 Olivier Carton and Luc Dartois. Aperiodic two-way transducers and fo-transductions. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, pages 160–174, 2015. URL: https://doi.org/10.4230/LIPIcs.CSL.2015.160, doi:10.4230/LIPIcs.CSL.2015.160.

8 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Regular separability of parikh automata. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 117:1–117:13, 2017. URL: https://doi.org/10.4230/LIPIcs.ICALP.2017.117, doi:10.4230/LIPIcs.ICALP.2017.117.

9 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.

10 Wojciech Czerwinski and Slawomir Lasota. Regular separability of one counter automata. *Logical Methods in Computer Science*, 15(2), 2019. URL: https://lmcs.episciences.org/5563.

11 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, pages 35:1–35:18, 2018. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2018.35, doi:10.4230/LIPIcs.CONCUR.2018.35.

12 Wojciech Czerwinski, Wim Martens, and Tomás Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 150–161, 2013. URL: https://doi.org/10.1007/978-3-642-39212-2_16, doi:10.1007/978-3-642-39212-2_16.

13 Wojciech Czerwinski, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017. URL: http://dmtcs.episciences.org/4131.

14 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001. URL: https://doi.org/10.1145/371316.371512, doi:10.1145/371316.371512.

15 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *Proceedings of the 43rd Annual*

ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016, pages 151–163, 2016. URL: https://doi.org/10.1145/2837614.2837627, doi:10.1145/2837614.2837627.

16 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

17 Anca Muscholl and Gabriele Puppis. The many facets of string transducers (invited talk). In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 2:1–2:21, 2019. URL: https://doi.org/10.4230/LIPIcs.STACS.2019.2, doi:10.4230/LIPIcs.STACS.2019.2.

18 Luke Ong. Higher-order model checking: An overview. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 1–15, 2015. URL: https://doi.org/10.1109/LICS.2015.9, doi:10.1109/LICS.2015.9.

19 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, pages 729–740, 2013. URL: https://doi.org/10.1007/978-3-642-40313-2_64, doi:10.1007/978-3-642-40313-2_64.

20 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 75:1–75:10, 2014. URL: https://doi.org/10.1145/2603088.2603098, doi:10.1145/2603088.2603098.

21 Thomas Place and Marc Zeitoun. Separation for dot-depth two. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. URL: https://doi.org/10.1109/LICS.2017.8005070, doi:10.1109/LICS.2017.8005070.

22 Kari-Jouko Räihä and Esko Ukkonen. The shortest common supersequence problem over binary alphabet is np-complete. *Theor. Comput. Sci.*, 16:187–198, 1981. URL: https://doi.org/10.1016/0304-3975(81)90075-X, doi:10.1016/0304-3975(81)90075-X.

23 J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, April 1959. doi:10.1147/rd.32.0198.

Appendix

A Proofs from Section 3

A.1 PTT separability of n-Rat, $n > 1$

The idea behind this proof is similar to that for the case of Rat. We construct using Lemma 20 ordered multi pushdown automata $\mathsf{OMPA}(T_1)$ and $\mathsf{OMPA}(T_2)$ corresponding to n-Rat transducers T_1, T_2 s.t. $\mathcal{L}(\mathsf{OMPA}(T_i)) = \{u\#v_1\#v_2\#\dots\#v_n \mid (u, v_1, \dots, v_n) \in [\![T_i]\!]\}$. The PTT separability problem of n-Rat then reduces to the PTL separability of ordered multi pushdown languages.

▶ **Lemma 20.** *Let T be n-dimensional rational transducer. We can construct a ordered multi pushdown automaton $\mathsf{OMPA}(T)$ which accepts the language $\{w\#w_1\#\dots\#w_n \mid (w, w_1, \dots, w_n) \in [\![T]\!]\}$.*

Proof. $\mathsf{OMPA}(T)$ is constructed from T as follows. $\mathsf{OMPA}(T)$ has $2n$ stacks for computation, two stacks for each output tape. The proof proceeds by a simulation of T exactly as in Lemma 4. Each transition of T is simulated by $\mathsf{OMPA}(T)$: whenever T outputs $(\gamma_1, \dots, \gamma_n)$, $\mathsf{OMPA}(T)$ pushes γ_i onto the i^{th} stack. When T (and $\mathsf{OMPA}(T)$) reaches a final state, $\mathsf{OMPA}(T)$ either moves to a new state q_{temp} or continues mimicking T. If it enters q_{temp}, it only reads $(\#\Gamma^*)^n$. It pops symbols from i^{th} stack and pushes the same symbols in $2i^{th}$ stack. This way the content of the ith output tape is in the correct order on stack $2i$. This is done for all stacks starting from $i = 1$, to maintain the ordering of the pop from the first non-empty stack. While looping over state q_{temp}, $\mathsf{OMPA}(T)$ does not read any symbols. When the first i stacks become empty in $\mathsf{OMPA}(T)$, it enters a state q_{f1} and reads symbol $\#$. Now on $\#$, it reads symbols from Γ and pops the same symbol from stack $n+1$; when stack $n+1$ becomes empty, it moves to state q_{f2} while reading $\#$, and pops the symbols from stack $n+2$ and read the same symbols. This continues for all stacks from $n+1$ to $2n$. Clearly $(w, w_1, \dots, w_n) \in [\![\mathcal{T}]\!] \Leftrightarrow (w\#w_1\#\dots\#w_n) \in \mathcal{L}(\mathsf{OMPA}(T))$.

◀

▶ **Theorem 21.** PTT *separability of n-Rat is decidable.*

Proof. Let T_1 and T_2 be n-dimensional rational transducers. We can construct OMPA A_1 and A_2 such that $\mathcal{L}(A_i) = \{u\#v_1\#v_2\#\dots\#v_n \mid (u, v_1, \dots, v_n) \in [\![T_i]\!]\}$ using lemma 20.

We claim that $\mathsf{OMPA}(T_1)$ and $\mathsf{OMPA}(T_2)$ are separable by an n-dimensional PTT T_3 iff the languages $\mathsf{OMPA}(T_1), \mathsf{OMPA}(T_2)$ are separable by a PTL. An n-dimensional PTT is an n-rational transducer containing simple relations $\Sigma^* \times (\Gamma^*)^n \cdot (a_1, (b_{11}, \dots, b_{n1})) \dots \Sigma^* \times (\Gamma^*)^n \cdot (a_k, (b_{k1}, \dots, b_{kn})) \cdot \Sigma^* \times (\Gamma^*)^n$ where all a_i are from Σ and all $b_{ij} \in \Gamma$, as well as finite boolean combinations of simple relations.

Assume that T_3 is a PTT separating T_1, T_2. Then we have $[\![T_1]\!] \subseteq [\![T_3]\!]$. Then $(w, w_1, \dots, w_n) \in [\![T_1]\!]$ implies $(w, w_1, \dots, w_n) \in [\![T_3]\!]$. By construction, $w\#w_1\#\dots\#w_n \in \mathcal{L}(\mathsf{OMPA}(T_1))$, $w\#w_1\#\dots\#w_n \in \mathcal{L}(\mathsf{OMPA}(T_3))$. It can be easily seen that $\mathcal{L}(\mathsf{OMPA}(T_1)) \subseteq \mathcal{L}(\mathsf{OMPA}(T_3))$.

We know that $[\![T_2]\!] \cap [\![T_3]\!] = \emptyset$, whenever $(w, w_1, \dots, w_n) \in [\![T_2]\!]$, $(w, w_1, \dots, w_n) \notin [\![T_3]\!]$, and conversely. By construction, this gives $w\#w_1\#\dots\#w_n \in \mathcal{L}(\mathsf{OMPA}(T_2))$ iff $w\#w_1\#\dots\#w_n \notin \mathcal{L}(\mathsf{OMPA}(T_3))$. This shows that $\mathsf{OMPA}(T_1), \mathsf{OMPA}(T_2)$ are separable by $\mathsf{OMPA}(T_3)$. By an easy extension of Lemma 5, it can be seen that $\mathsf{OMPA}(T_3)$ is a PTL.

For the converse, assume that $\mathsf{OMPA}(T_1)$ and $\mathsf{OMPA}(T_2)$ are separable by PTL L_3. Then $\mathcal{L}(\mathsf{OMPA}(T_1)) \subseteq L_3$ and $\mathcal{L}(\mathsf{OMPA}(T_2)) \cap L_3 = \emptyset$. L_3 is a PTL so, there exists a first order

logic formula φ such that $\mathcal{L}(\varphi) = L_3$ and φ is a boolean combination of existential formulas. Consider the formula

$$\varphi' = \varphi \wedge \exists x_1 \dots x_n \bigwedge_{i=1}^{n} \#(x_i) \wedge \bigwedge_{i,j=1 \; i \neq j}^{n} (x_i \neq x_j) \wedge \neg \exists x_1 \dots x_{n+1} \left(\bigwedge_{i=1}^{n} \#(x_i) \wedge \bigwedge_{i,j=1 \; i \neq j}^{n+1} (x_i \neq x_j) \right)$$

φ' restricts the number of #s to be exactly n in all accepted words. Notice that the PTL generated by φ' is also a separator of $\mathsf{OMPA}(T_1)$ and $\mathsf{OMPA}(T_2)$.

To construct the PTT separating T_1, T_2, we proceed as in Lemma 6. Consider the DFA $\mathcal{A}$ accepting the language $\mathcal{L}(\varphi')$ over $\Sigma \cup \Gamma \cup \{\#\}$. We can construct a PTT T_3, whose space space is $n+1$ copies of state space of $\mathcal{A}$, after seeing i number of #s, we move to copy $i+1$. T_3 mimics the transitions of $\mathcal{A}$ by reading the same input symbol as $\mathcal{A}$ and producing the empty word as output until #, which it treats like $\dashv$. When $\mathcal{A}$ reads #, T_3 outputs ϵ, and stays in the same state, same position, but in copy 2. From this state, again it mimics the transitions of $\mathcal{A}$ but the input symbol of $\mathcal{A}$ (from Γ) becomes the output symbol for the first dimension for T_3, the output for all other dimensions is ϵ, and the input symbol for T_3 is ϵ. While producing output, it continues being in states with flag 1. When reading symbols of Γ between the ith and $i+1$st #, T_3 outputs the symbol read for the ith dimension, and outputs ϵ for all other dimensions. This process is repeated for each # in the input. Final states are same as $\mathcal{A}$ with flag $n+1$.

The decidability follows from the decidability of the piecewise testable separability of multi pushdown automata, shown in theorem 7.

◀

A.2 PTT separability of n-Sweep

Next we show the separability result for sweeping transducers. These are extensions of rational transducers such that the input can be read in both ways, but reversal happens at ends of the input tape only. We consider unambiguous sweeping transducer where the number of reversals are bounded.

▶ **Theorem 22.** PTT *separability of* n-Sweep *is decidable.*

Proof. Given two sweeping transducers T_1 and T_2, we encode the relation defined by them in OMPA P_1 and P_2 respectively such that $(u, v) \in [\![T_i]\!] \Leftrightarrow u\#v \in \mathcal{L}(P_i)$.

As a first step, we convert the k-sweeping transducer into a streaming string transducer (SST) with k registers.

SST

Streaming string transducers [2, 3] are one-way finite-state transducers that manipulates a finite set of string variables to compute its output. Instead of appending symbols to the output tape, SSTs concurrently update all string variables using a concatenation of string variables and output symbols. The transformation of a string is then defined using an output (partial) function F that associates states with a concatenation of string variables, s.t. if the state q is reached after reading the string and $F(q)$=xy, then the output string is the final valuation of x concatenated with that of y.

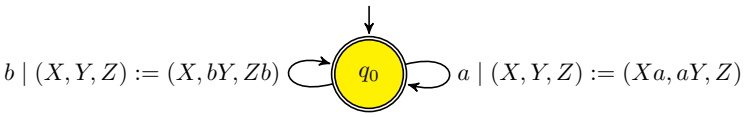
Let $\mathcal{X}$ be a finite set of variables and Γ be a finite alphabet. A substitution σ is defined as a mapping $\sigma : \mathcal{X} \to (\Gamma \cup \mathcal{X})^*$. A valuation is defined as a substitution $\sigma : \mathcal{X} \to \Gamma^*$. Let $\mathcal{S}_{\mathcal{X},\Gamma}$ be the set of all substitutions $[\mathcal{X} \to (\Gamma \cup \mathcal{X})^*]$. Any substitution σ can be

extended to $\hat{\sigma} : (\Gamma \cup \mathcal{X})^* \to (\Gamma \cup \mathcal{X})^*$ in a straightforward manner. The composition $\sigma_1\sigma_2$ of two substitutions σ_1 and σ_2 is defined as the standard function composition $\hat{\sigma_1}\sigma_2$, i.e. $\hat{\sigma_1}\sigma_2(X) = \hat{\sigma_1}(\sigma_2(X))$ for all $X \in \mathcal{X}$. We now introduce streaming string transducers. A streaming string transducer is a tuple $(\Sigma, \Gamma, Q, q_0, Q_f, \delta, \mathcal{X}, \rho, F)$ where: (1) Σ and Γ are finite sets of input and output alphabets; (2) Q is a finite set of states with initial state q_0; (3) $\delta : Q \times \Sigma \to Q$ is a transition function; (4) $\mathcal{X}$ is a finite set of variables; (5) $\rho : \delta \to \mathcal{S}_{\mathcal{X},\Gamma}$ is a variable update function; (6) Q_f is a subset of final states; and (7) $F : Q_f \rightharpoonup \mathcal{X}^*$ is an output function.

The concept of a run of an SST is defined in an analogous manner to that of a finite state automaton. The sequence $\langle \sigma_{r,i} \rangle_{0 \le i \le |r|}$ of substitutions induced by a run $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots q_{n-1} \xrightarrow{a_n} q_n$ is defined inductively as the following: $\sigma_{r,i} = \sigma_{r,i-1}\rho(q_{i-1}, a_i)$ for $1 < i \le |r|$ and $\sigma_{r,1} = \rho(q_0, a_1)$. We denote $\sigma_{r,|r|}$ by σ_r. If the run r is final, i.e. $q_n \in Q_f$, we can extend the output function F to the run r by $F(r) = \sigma_\epsilon \sigma_r F(q_n)$, where σ_ϵ substitutes all variables by their initial value ϵ. For all strings $s \in \Sigma^*$, the output of s by T is defined only if there exists an accepting run r of T on s, and in that case the output is denoted by $T(s) = F(r)$. The transformation $[\![T]\!]$ defined by an SST T is the function $\{(s, T(s)) : T(s) \text{ is defined}\}$.

Let us consider the SST T_2 with one state q_0 and three variables X, Y, and Z, shown below implementing the transformation f_1 defined as follows. Let $\Sigma = \{a, b\}$. For all strings $s \in \Sigma^*$, we denote by $\overline{s}$ its mirror image, and for all $\sigma \in \Sigma$, by $s \backslash \sigma$ the string obtained by removing all symbols σ from s. The transformation $f_1 : \Sigma^* \rightharpoonup \Sigma^*$ maps any string $s \in \Sigma^*$ to the output string $(s \backslash b)\overline{s}(s \backslash a)$. For example, $f_1(abaa) = aaa.aaba.b$.

The variable update is shown in the figure and the output function is s.t. $F(q_0) = XYZ$.



Unambiguous $k-$sweeping transducer to k-register Concatenation Free SST

An SST is concatenation free when the right hand sides of variable updates never concatenate registers. That is, there will never be an update of the form $x := \gamma_1 x_1 \gamma_2 x_2 \gamma_3$ where $x, x_1, x_2 \in \mathcal{X}$ and $\gamma_1, \gamma_2, \gamma_3 \in \Gamma^*$.

Now we show that given an unambiguous $k-$pass sweeping transducer $\mathcal{T}$, we can construct a k-register concatenation free streaming string transducer(SST) $\mathcal{T}'$ such that $[\![\mathcal{T}]\!] = [\![\mathcal{T}']\!]$, and there are $\lceil k/2 \rceil$ *append-registers* and $\lceil k/2 \rceil$ *prepend-registers*. If x is a prepend-register then, $x = \gamma.x$ are the only updates allowed in the SST, for $\gamma \in \Gamma^*$. Similarly, if x is an append register the updates allowed are of the form $x = x.\gamma$ where $\gamma \in \Gamma^*$. The following proof is an adaptation of the equivalence proof of concatenation free SSTs and unambiguous k-sweeping transducers [6].

The main idea is to construct a one way automaton $\mathcal{A}$ from the underlying two-way automaton of $\mathcal{T}$ by applying the classical Shepherdson's algorithm [23] using crossing sequences (crossing sequences are defined in Section 4.2). States of $\mathcal{A}$ are crossing sequences of length/height k, and are thus tuples of the form $(q_1, \dots, q_k)$, where q_i is a state of $\mathcal{T}$ which visited the position during the ith sweep. If a word $u \in \mathsf{dom}(\mathcal{T})$, then it admits a unique accepting run in $\mathcal{A}$. We define the register updates for every transition of $\mathcal{A}$ as follows.

1. We maintain prepend registers for all even indexed passes (sweeps 2, 4 ...) and append registers for all odd indexed passes, (sweeps 1, 3, ...), obtaining a total of k registers.

2. A transition $(q_1, q_2, \ldots, q_k) \rightarrow^a (p_1, p_2, \ldots, p_k)$ from state $(q_1, q_2, \ldots, q_k)$ to a state $(p_1, p_2, \ldots, p_k)$ is allowed in $\mathcal{A}$ when we have the transition $q_i \rightarrow^a (p_i, +1)$ for each odd i and the transition $p_i \rightarrow^a (q_i, -1)$ for each even i in $\mathcal{T}$.
3. For all odd i, the register x_i is updated as $x_i = x_i.w_i$ where w_i is the output produced during $q_i \rightarrow^a (p_i, +1)$ transition in $\mathcal{T}$. This accumulates the output in a left to right sweep.
4. Similarly for all even i, register x_i is updated as $x_i = w_i^R.x_i$ where w_i^R is the reverse of the output w_i produced during $p_i \rightarrow^a (q_i, -1)$ transition in $\mathcal{T}$. This accumulates the output in a right to left sweep. The prepend gives the output of the ith right to left sweep in the correct order in which it was produced in $\mathcal{T}$.
5. On the accepting state of $\mathcal{A}$, the output function is defined as $x_1 x_2 \ldots x_n$. The output is computed by concatenating the contents of the registers in the same order as the sweeps of $\mathcal{T}$.

Concatenation free SST $\mathcal{T}$ to OMPA $\mathcal{A}$ s.t. $(u, v) \in \mathcal{T} \Leftrightarrow u\#v \in \mathcal{L}(\mathcal{A})$

Start with the SST $\mathcal{B}$ constructed as above using only prepend and append registers. Let the set of registers in $\mathcal{B}$ be $X = \{x_1, \ldots, x_m\}$ (wlog let m be even). Recall that x_1 is an append register, x_2 is a prepend register, x_3 is an append register and so on alternatively. We construct an $m + \lceil m/2 \rceil$-ordered multi pushdown automaton $\mathcal{A}$ from the SST $\mathcal{B}$ as follows. $\mathcal{A}$ simulates $\mathcal{B}$ on the input, and, in addition we have m temporary states.

- For every transition $q \rightarrow^a q'$ with S_σ the register update function in $\mathcal{B}$ which updates x_1 as $x_1 w_1$, x_2 as $w_2 x_2$ and so on, we have following transition in $\mathcal{A}$:
  $< q, \epsilon, \epsilon, \epsilon, \ldots, \epsilon > \rightarrow^a < q', w_1, \epsilon, w_2, w_3, \epsilon, w_4, w_5, \epsilon, \ldots, w_{m-1}, \epsilon, w_m >$

where w_i is the output word added to register x_i on left in S_σ for even i, and w_i is the output word added to register x_i on right in S_σ for odd i. When the whole word is read and we reach some accepting state q_f in $\mathcal{B}$, we have the value of registers x_i in stack $i + \lceil i/2 \rceil$. If i is even, value of x_i is stored in the corresponding stack in correct order i.e. the value is same as the word we get by reading stack top to bottom. For odd i, the value of x_i is stored in the stack in the reverse order. From q_f (final state of $\mathcal{B}$), $\mathcal{A}$ can either continue simulating transition of $\mathcal{B}$ or read # and switch the control to state temp_1. Once temp_1 is reached, $\mathcal{A}$ reads symbol from Γ only. First we correct the value of x_1 i.e. reverse the content of stack 1 by pushing its elements to stack 2. After this operation, stack 1 is empty and value of x_1 is in correct order in stack 2. Hence we pop the symbols from stack 2 while reading the same symbol in the input.

Once stack 2 becomes empty, control changes to state temp_2. Similarly $\mathcal{A}$ has to read the content of register x_2 next, which is stored in stack 3 in correct order. So it will read symbols from Γ and pop the same symbol from stack 3, while being in temp_2. When the stack 3 becomes empty, control changes to state temp_3. The above procedure is repeated for all stacks, following the corresponding operations for even and odd i from temp_i. It is clear that, when all stacks are empty we have read the input word, followed by #, followed by its output in $\mathcal{A}$.

We have shown that the relations defined by concatenation free SSTs can be encoded in the language of OMPA. Finally we can prove that the separability of $n-$Sweep by PTT is decidable iff separability of the OMPLs (constructed above) by PTL is decidable similar to lemma 6, which completes this proof.

◀

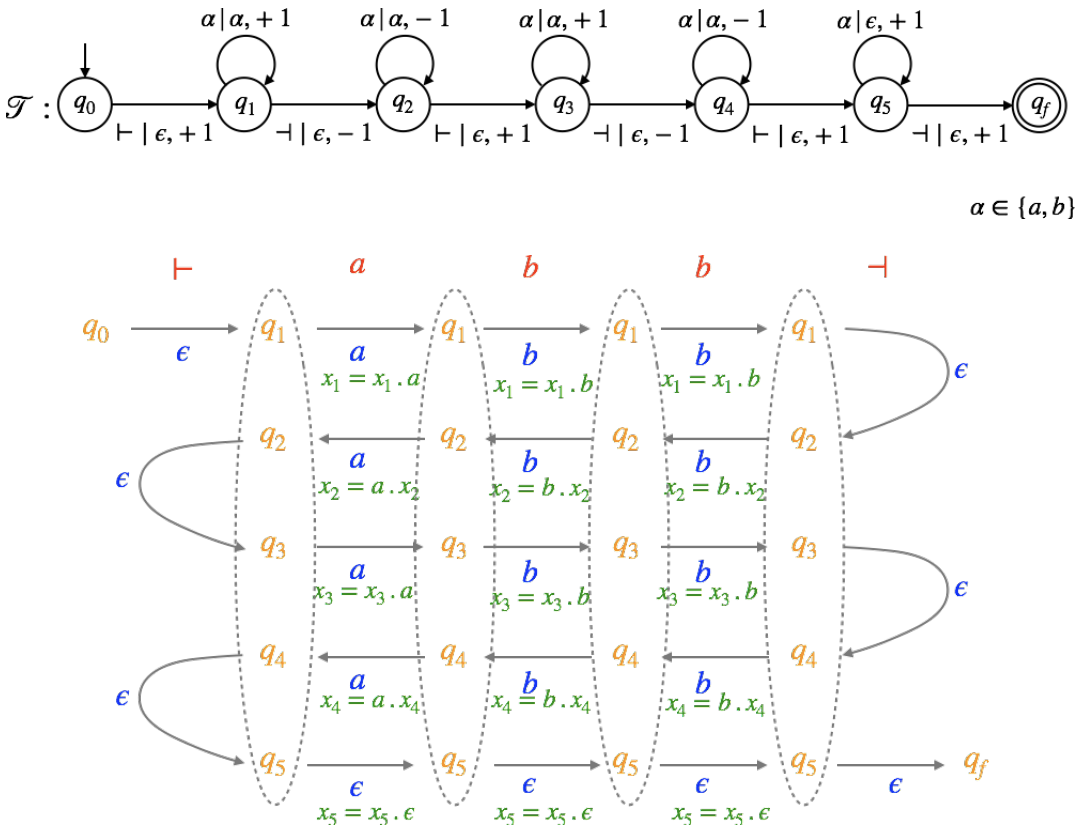


Figure 1 An example illustrating the conversion from 5-sweeping transducer to concatenation free SST given in section A.2. The sweeping transducer $\mathcal{T}$ defines a function $f(w) = ww^R ww^R$ for any $w \in \{a, b\}^*$. Notice that it needs 4 sweeps to produce the output and last sweep is to reach the end of the input to accept the word. An accepting run of $w = \vdash abb \dashv$ on $\mathcal{T}$ is shown in the figure, blue colored symbols represent the output produced in the sweeping transducer. After applying Shepherdson's algorithm we get states of one way automaton, which is shown as dotted ovals in the figure. Corresponding to each sweep we have a register. During left to right sweep (from $\vdash$ to $\dashv$), we update the register by appending the output produced by transitions. This way x_1, x_3, x_5 store the correct value of sweep 1, 3 and 5. During right to left sweep (from $\dashv$ to $\vdash$), the output produced is prepended to the corresponding register to get the output of that pass in correct order in CFSST. In the final state of CFSST i.e. right most circled state we will have the output of w in the concatenation $x_1 x_2 \ldots x_5$.

B PTL Separability of OMPL

The main contribution of this section is the simulation of ordered multipushdown automata having n stacks using an n-order HOPDA. In [13], it has been shown that for a language class $\mathcal{C}$ satisfying the conditions of a full trio, that is, closure under morphism, inverse morphism and intersection with regular languages, PTL separability is decidable if and only if (i) downward closure is computable for class $\mathcal{C}$ iff (ii) diagonal problem is decidable for class $\mathcal{C}$. This result of [13] shows the decidability of PTL separability for the class of HOPDA.

Lemma 23 shows that we can simulate the language of an n-ordered multi pushdown automaton by an n-order higher order pushdown automaton.

B.1 n HOPDA

HOPDA are a generalization of PDA where we can have nested stacks: order-0 stack is a symbol from Γ, order-1 stack is a stack of order-0 stacks same as PDA, order-2 stack is a stack of order-1 stacks and so on. An order n push operation pushes a new order-$(n-1)$ stack onto the order-n stack, which is a copy of the existing topmost order-$(n-1)$ stack. A rewrite operation changes the character which is at the top of the topmost stack.

Order-n Stacks. The set of order-n stacks over the stack alphabet Γ is defined inductively as follows [15].

$$\mathcal{S}_0^\Gamma = \Gamma$$
$$\mathcal{S}_{k+1}^\Gamma = \{[s_1 \dots s_m]_{k+1} \mid \forall i, s_i \in \mathcal{S}_k^\Gamma\}$$

Stacks are written with their tops to the left. A push will move the existing top of stack to the right. We define several operations. Like PDA, we can read the top most stack element. On each transition, we can have a stack operation of the form push_i, pop_i, rew_γ or nop for $i \in [1, n]$. push_i copies the topmost order $(i-1)$ stack in top most order i stack, and adds the copy as the new top in the order-i stack. pop_i pops the top of the topmost order $(i-1)$ stack in the order-i stack. These are defined formally below.

- $\text{top}_k([s_1 s_2 \dots s_m]_k) = s_1$
- $\text{top}_k([s_1 s_2 \dots s_m]_n) = \text{top}_k(s_1)$ if $n > k$
- $\text{rew}_\gamma([\gamma_1 \gamma_2 \dots \gamma_m]_1) = [\gamma \gamma_2 \dots \gamma_m]_1$
- $\text{rew}_\gamma([s_1 s_2 \dots s_m]_n) = [\text{rew}_\gamma(s_1) s_2 \dots s_m]_n$ for $n > 1$
- $\text{push}_k([s_1 s_2 \dots s_m]_k) = [s_1 s_1 s_2 \dots s_m]_k$
- $\text{push}_k([s_1 s_2 \dots s_m]_n) = [\text{push}_k(s_1) s_2 \dots s_m]_n$ for $k < n$
- $\text{pop}_k([s_1 s_2 \dots s_m]_k) = [s_2 \dots s_m]_k$
- $\text{pop}_k([s_1 s_2 \dots s_m]_n) = [\text{pop}_k(s_1) s_2 \dots s_m]_n$ for $k < n$

A transition in a n-HOPDA is s.t. from some state p, on reading an input symbol (or ϵ), it performs an operation of the above kind and changes state. In the initial configuration, the n-HOPDA is in its initial state with stack contents $[[\dots[[\bot]_1]_2 \dots]_{n-1}]_n$.

▶ **Lemma 23.** *Given an ordered multi pushdown automaton $\mathcal{A}$ with n stacks, one can construct a higher order multi pushdown automaton $\mathcal{B}$ of order n such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

Proof. Given an n-OMPA $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \gamma_0, F)$ in normal form (we follow the notations from [4]), we can construct order-n HOPDA $\mathcal{B} = (Q \cup \{q_{\text{in}}\}, \Sigma, \Gamma, \Delta', q_{\text{in}}, \gamma_0, F)$ as follows. In HOPDA, we allow sequences of operations $o1; ...; o_m$ in the rules instead of single operations. When using sequences we allow a test operation $\gamma?$ that only allows the sequence to proceed

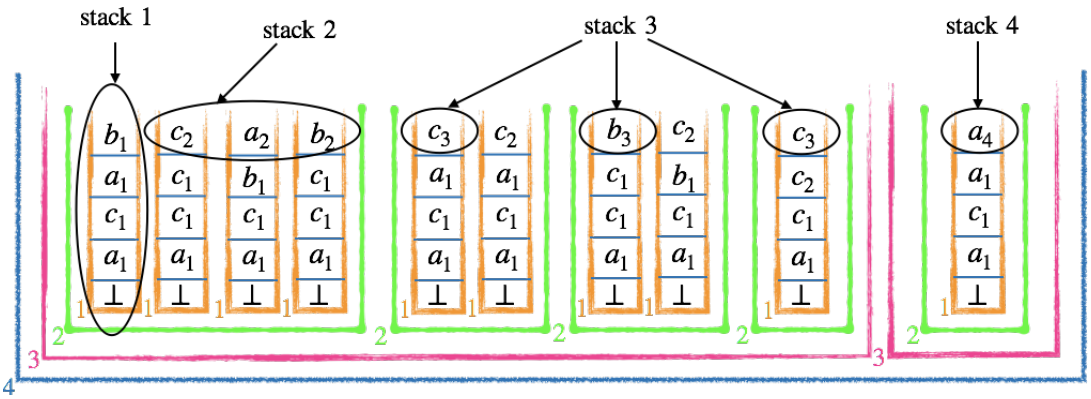


Figure 2 figure explaining the content of stacks stored in hopda

if the top_1 character of the stack is γ. This extension can be encoded by introducing intermediate control states.

The idea is to store the content of stack i of the OMPA into the top most stack of order $i+1$. In other words, top order-1 stack represents the content of stack 1, top elements of the stacks at order 2 except the first one, represents the content of stack 2 and so on. A mapping between the jth element of stack i in the OMPA to the respective element in the HOPDA is presented in Theorem 7. It is also represented graphically in the figure 2. The formal construction of order n HOPDA is shown below. We assume the OMPA is in the normal form [4]. Below, we explain wrt the transitions in the normal form of OMPA, the corresponding transitions in the HOPDA.

- Δ' :
  - First, we have special transition to initialize:

$$< q_{\text{in}}, \bot > \rightarrow < q_0,\ \text{push}_n;\ \text{push}_{n-1}; \ldots;\ \text{push}_2 >$$

    From the initial stack configuration $[[\ldots[[\bot]_1]_2\ldots]_{n-1}]_n$, this first gives
    - $[[\ldots[[\bot]_1]_2\ldots]_{n-1}\quad[\ldots[[\bot]_1]_2\ldots]_{n-1}]_n$, then
    - $[\ [[\ldots[[\bot]_1]_2\ldots]_{n-2}\quad[\ldots[[\bot]_1]_2\ldots]_{n-2}\ ]_{n-1}\quad[\ldots[[\bot]_1]_2\ldots]_{n-1}]_n$ and so on.
  - Now consider the transition in normal form which pops $\gamma \in \Gamma$ from stack 1 of the OMPA and pushes atmost 2 symbols in stack 1, and atmost 1 symbol in stacks 2 to n. Such a transition in the OMPA has the form $< q, \gamma, \epsilon, \ldots, \epsilon > \ \rightarrow^a_A\ < q', \alpha_1, \ldots, \alpha_n >$ such that $\gamma \in \Gamma$, and $\alpha_1 \in \Gamma^*$, $|\alpha_1| \leq 2$ and $\alpha_j \in \Gamma_\epsilon \setminus \{\bot\}$ for all $j \in [2,n]$. The ϵs represent that the top of stacks 2 onwards cannot be accessed. If we assume $\alpha_1 = cd \in \Gamma^2$ we have the corresponding transition in B:
    $(q,\gamma) \rightarrow^a_B (q', \text{rew}_c; \text{push}_1; \text{rew}_d; [\text{push}_1; \text{rew}_{\alpha_j}; \text{push}_j; \text{pop}_1]_{j=2}^n)$ where the last four operations are repeated for $j \in [2,n]$ in the sequence for which $\alpha_j \neq \epsilon$. First operation rew_c replaces top symbol γ with c. The next two operations $\text{push}_1; \text{rew}_d$ pushes d on the top most order 1 stack. Next three operations $\text{push}_1; \text{rew}_{\alpha_j}; \text{push}_j$ first pushes α_j on top most order 1 stack, then makes the copy of first top most order j stack, this way we have α_j in the desired place. At last, it pops the symbol α_j from the top stack to maintain correct stack 1 content in top order 1 stack. The last four operations are repeated for all $j \in [2,n]$ to push α_j in the required place in the HOPDA.

- The next kind of transition in the normal form is one where the top of the first non-empty stack (say i) is popped, and at the same time, we push a symbol of Γ to stack 1. For every transition $< q, \bot, \ldots, \bot, \gamma, \epsilon, \ldots, \epsilon > \ \rightarrow^a_A \ < q', \gamma'\bot, \bot, \ldots, \bot, \epsilon, \ldots, \epsilon >$ where $\gamma, \gamma' \in \Gamma \setminus \{\bot\}$ and first non-empty stack is $i > 1$, we have the following transition in B:

$$\begin{aligned}(q, \bot) \rightarrow^a (q', \mathrm{pop}_2, \bot?\mathrm{pop}_3; \ldots \bot?\mathrm{pop}_{i-1}; \gamma?\mathrm{pop}_i;\\ \mathrm{push}_i; (\mathrm{pop}_{i-1})^*; (\mathrm{pop}_{i-2})^*; \ldots; (\mathrm{pop}_1)^*;\\ \mathrm{push}_2; \mathrm{push}_3; \ldots \mathrm{push}_{i-1}; \mathrm{push}_1; \mathrm{rew}_{\gamma'})\end{aligned}$$

  Operations on first line checks whether the first stack in the OMPA is empty, if yes it pops the top order-1 stack from the order-2 stack. Then it checks if second stack of OMPA is empty, if yes, it pops the topmost order-2 stack from the order-3 stack. The top of the current topmost order-2 stack in the order-3 stack represents the top of stack 3 in the OMPA. If that is $\bot$, then stack 3 is empty in the OMPA. This order-3 stack (the topmost in order-4) is popped from the HOPDA. The top of the current topmost order-3 stack in the order-4 stack represents the top of stack 4 in the OMPA and so on. When this checks reaches the i^{th} stack of the OMPA and if its top most element is γ, then it pops the top of the order i stack. During this process, we have lost the initial marker $\bot$ for the stacks 1 to $i-1$. Operations on the second line takes care of this. It first copies the order i stack and then empties it. After execution of these operations the top stack in the order i stack will look like $[[\ldots[\bot]_1]_{i-2}]_{i-1}$. This way end marker of stack 1 is restored, we have to do it for all stacks. The third line is used to restore $\bot$ in stacks 2 to $i-1$. At the end of this, we push γ' on stack 1 i.e. push_1 and rewriting it with γ' in B.

◀

With the help of Lemma 23 and the decidability of separability of HOPDA by PTL, we can infer the following:

B.2 Proof of Theorem 7

Separability of languages of ordered multi pushdown automata by PTL is decidable.

To prove the correctness of the construction given in Lemma 23, we provide an encoding function which maps stack contents of HOPDA to the respective element in the OMPA. Consider a function f from configurations of n-order HOPDA to the configurations of n-OMPA defined as following:

$f([s_1 s_2 \ldots s_m]_n) = \ < s'_1, s'_2, \ldots, s_n >$ where s'_i represents the content of i^{th} stack of the OMPA. $s'_i[j]$ represents the j^{th} element in i^{th} stack from the top in the OMPA. Let $([s_1 \ldots s_m]_k)[i] = s_i$ for any order k.

- $s'_1[1] = \mathrm{top}_1([s_1 s_2 \ldots s_m]_n)$
  top_1 gives the top of the topmost order-1 stack. This is clearly the top of stack 1 ($s'_1[1]$) in the OMPA.
- $s'_1[i] = (\mathrm{top}_2([s_1 s_2 \ldots s_m]_n))[i]$
  To obtain the ith element from the top of stack 1 in the OMPA, first consider $top_2(s_1)$, where s_1 is the leftmost order-$(n-1)$ stack in the order-n stack. This recursively applies top_2 to the topmost order-$(n-2)$ stack, and so on until we reach the topmost order-2 stack. top_2 applied to this gives the topmost order-1 stack. Applying $[i]$ to this order 1 stack gives its ith element.

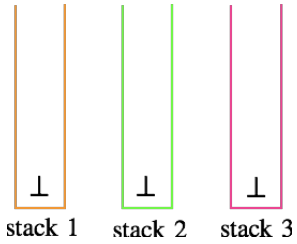
- $s'_2[i] = \text{top}_1((\text{top}_3([s_1 s_2 \dots s_m]_n))[i+1])$
  To access the ith element in stack 2 of the OMPA, first use top_3 to obtain the top of the topmost stack of order-3. This is an order-2 stack. Applying $[i+1]$ to this gives the $i+1$st order-1 stack in this. The top of this is the ith element of stack 2.
- $s'_3[i] = \text{top}_1((\text{top}_4([s_1 s_2 \dots s_m]_n))[i+1])$
  Use top_4 to obtain the the top of the topmost stack of order-4. This is an order-3 stack. Applying $[i+1]$ to this gives the $i+1$st order-2 stack in this. top_1 applied to this gives the top of this stack, which is the ith element of stack 3 in the OMPA.
  $s'_j[i] = \text{top}_1((\text{top}_{j+1}([s_1 s_2 \dots s_m]_n))[i+1])$ for all $j \in [2, n-1]$
- $s'_n[i] = \text{top}_1(s_{i+1})$

For the initial configuration, we have a special mapping: $f([\dots[\bot]_1 \dots]_n) = \ < \bot, \bot, \dots, \bot >$

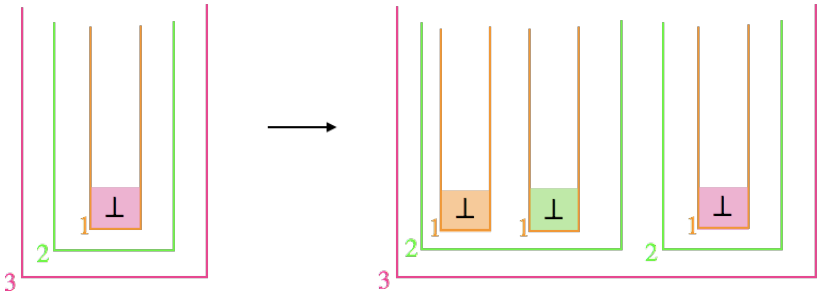
Lemma 23 is proved in such a way that when a transition t is executed in the OMPA A, if the configuration is c', and if a corresponding transition is applied in the HOPDA, and the resultant configuration is c, then $f(c) = c'$.

The construction of the HOPDA given in Lemma 23 is explained below by an example. Simulation of sample run of A in B as is shown in the figures below.

The initial configuration of A looks like the following:



Corresponding to this we have transition in $B : < q_{\text{in}}, \bot > \rightarrow < q_0,\ \text{push}_3;\ \text{push}_2 >$



Suppose following are the transitions executed in OMPA and their configuration, corresponding transitions in HOPDA and their configurations are shown right below it:

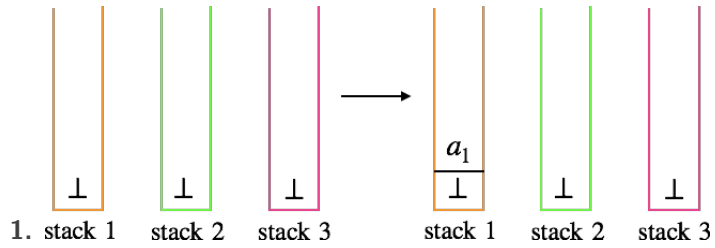


Figure 3 $< q_0, \bot, \epsilon, \epsilon > \rightarrow_A^a < q_1, \bot a_1, \epsilon, \epsilon >$

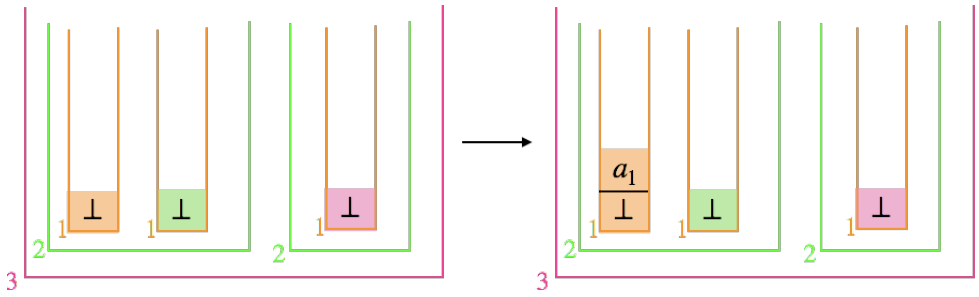


Figure 4 $< q_{\text{in}}, \bot > \rightarrow_B^b < q_1,\ \text{rew}_\bot;\ \text{push}_1;\ \text{rew}_{a_1} >$

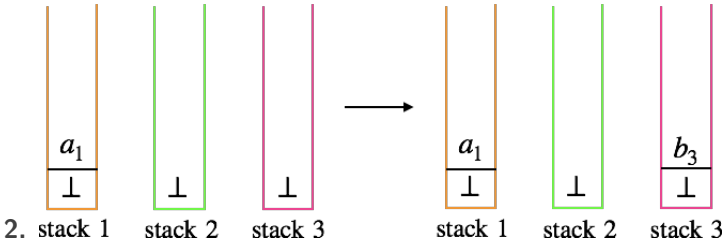


Figure 5 $< q_1, a_1, \epsilon, \epsilon > \rightarrow_A^a < q_2, a_1, \epsilon, b_3 >$

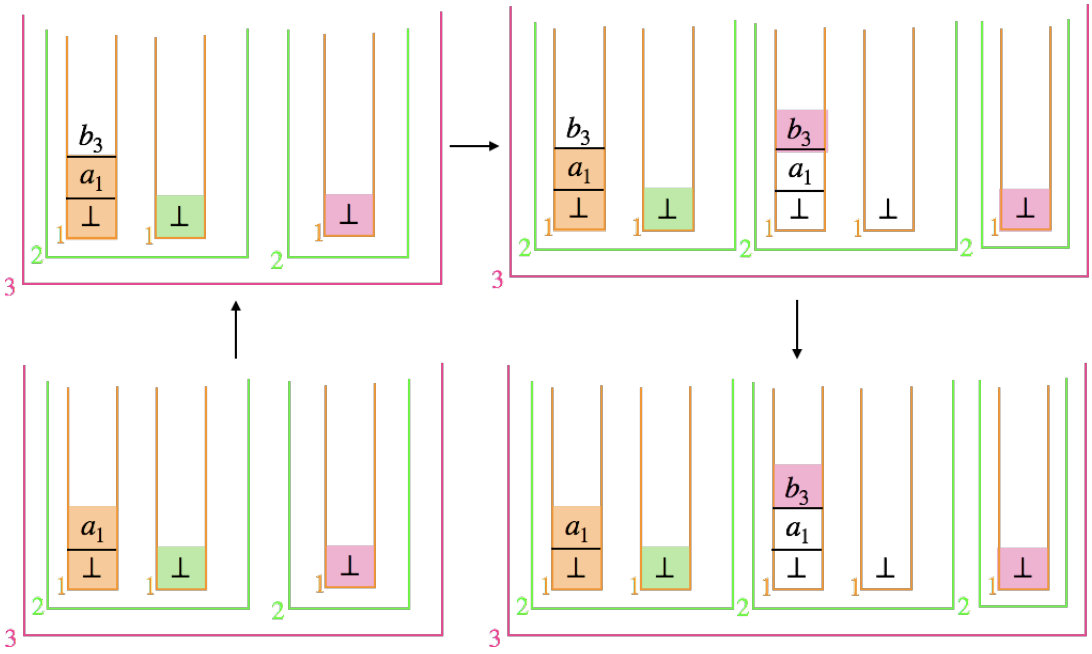


Figure 6 $< q_1, a_1 > \rightarrow_B^a < q_2,\ \mathrm{rew}_{a_1};\ \mathrm{push}_1;\ \mathrm{rew}_{b_3};\ \mathrm{push}_3;\ \mathrm{pop}_1 >$

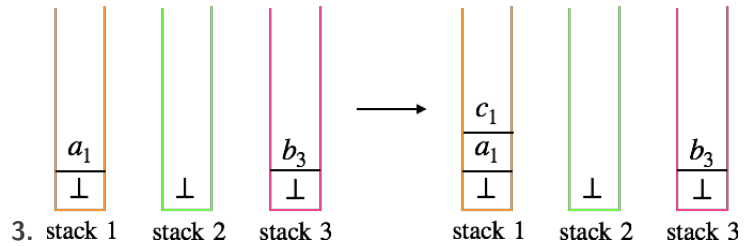


Figure 7 $< q_2, a_1, \epsilon, \epsilon > \rightarrow_A^a < q_3, a_1 c_1, \epsilon, \epsilon >$

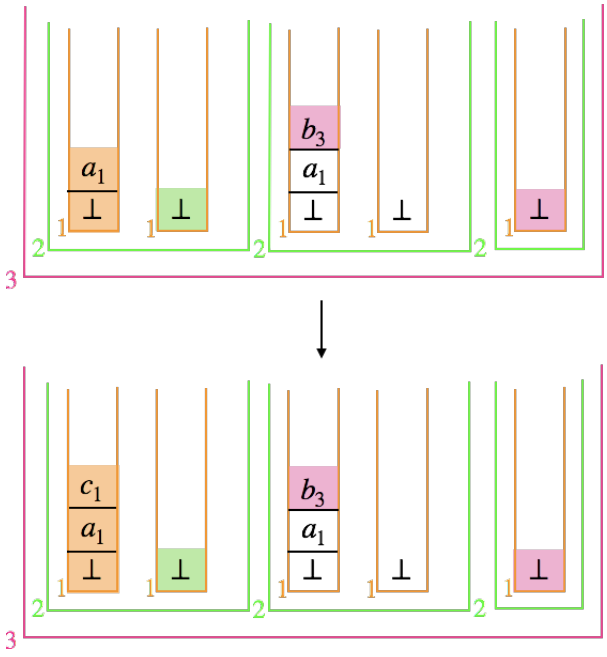


Figure 8 $< q_2, a_1 > \rightarrow_B^a < q_3,\ \mathrm{rew}_{a_1};\ \mathrm{push}_1;\ \mathrm{rew}_{c_1} >$

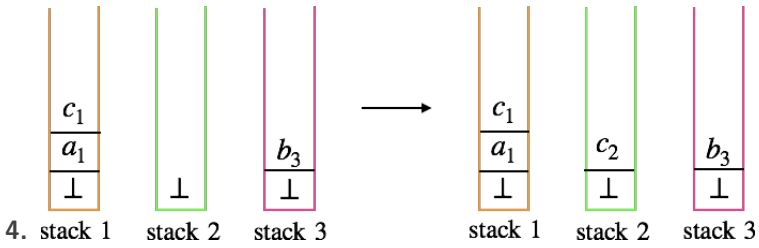


Figure 9 $< q_3, c_1, \epsilon, \epsilon > \rightarrow_A^a < q_4, c_1, c_2, \epsilon >$

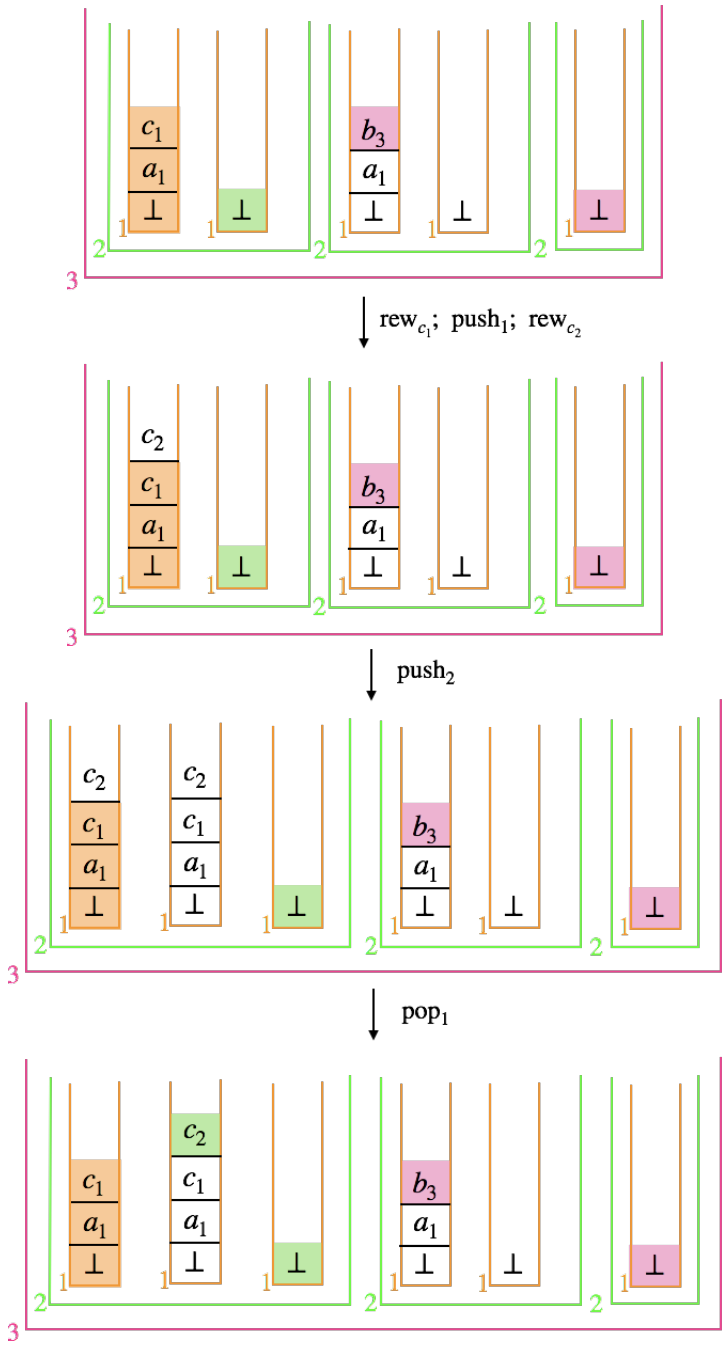


Figure 10 $< q_3, c_1 > \rightarrow_B^a < q_4,\ \text{rew}_{c_1};\ \text{push}_1;\ \text{rew}_{c_2};\ \text{push}_2;\ \text{pop}_1 >$

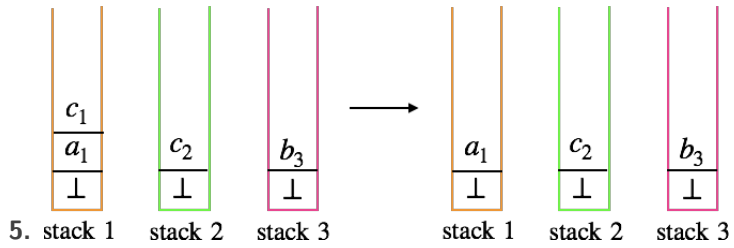


Figure 11 $< q_4, c_1, \epsilon, \epsilon > \rightarrow_A^a < q_5, \epsilon, \epsilon, \epsilon >$

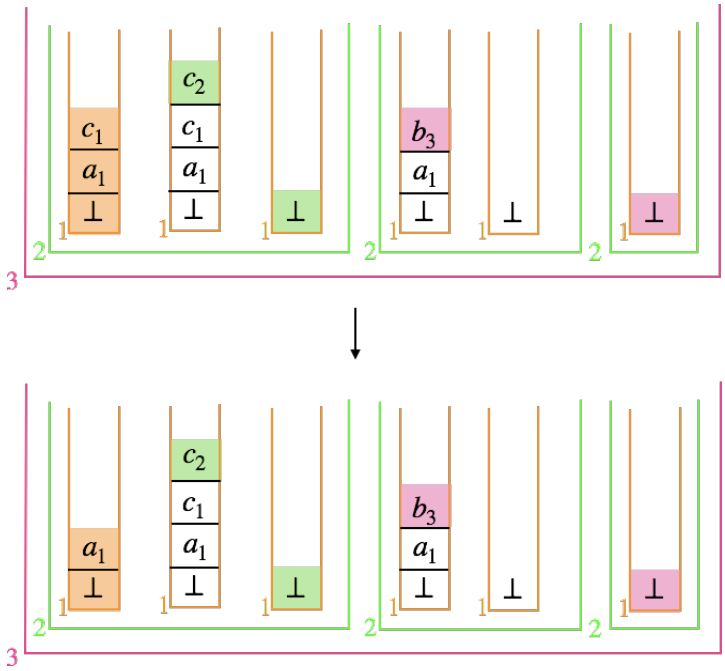


Figure 12 $< q_4, c_1 > \rightarrow_B^a < q_5, \ \mathrm{pop}_1 >$

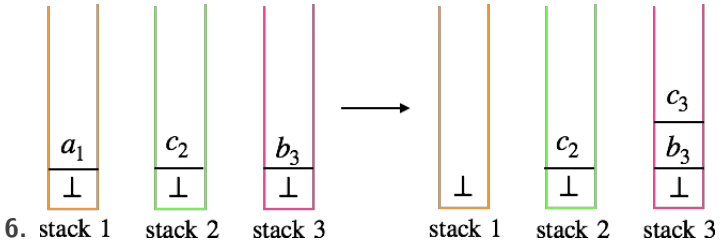


Figure 13 $< q_5, a_1, \epsilon, \epsilon > \rightarrow_A^a < q_6, \epsilon, \epsilon, c_3 >$

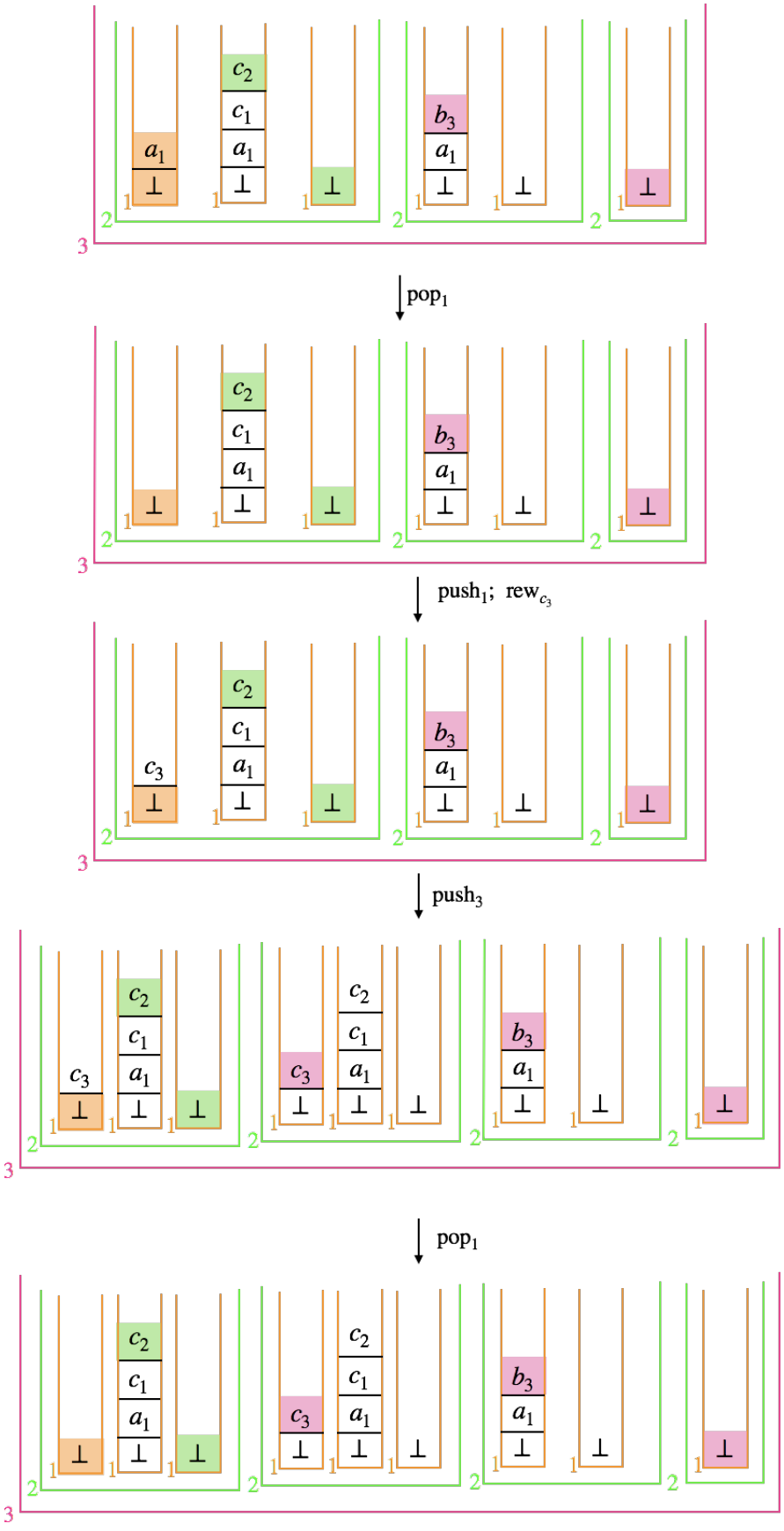


Figure 14 $< q_5, a_1 > \rightarrow_B^a < q_6,\ \mathrm{pop}_1;\ \mathrm{push}_1;\ \mathrm{rew}_{c_3};\ \mathrm{push}_3;\ \mathrm{pop}_1 >$

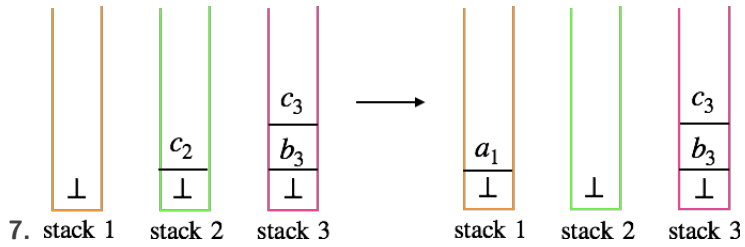


Figure 15 $< q_6, \perp, c_2, \epsilon > \rightarrow_A^a < q_7, a_1, \epsilon, \epsilon >$

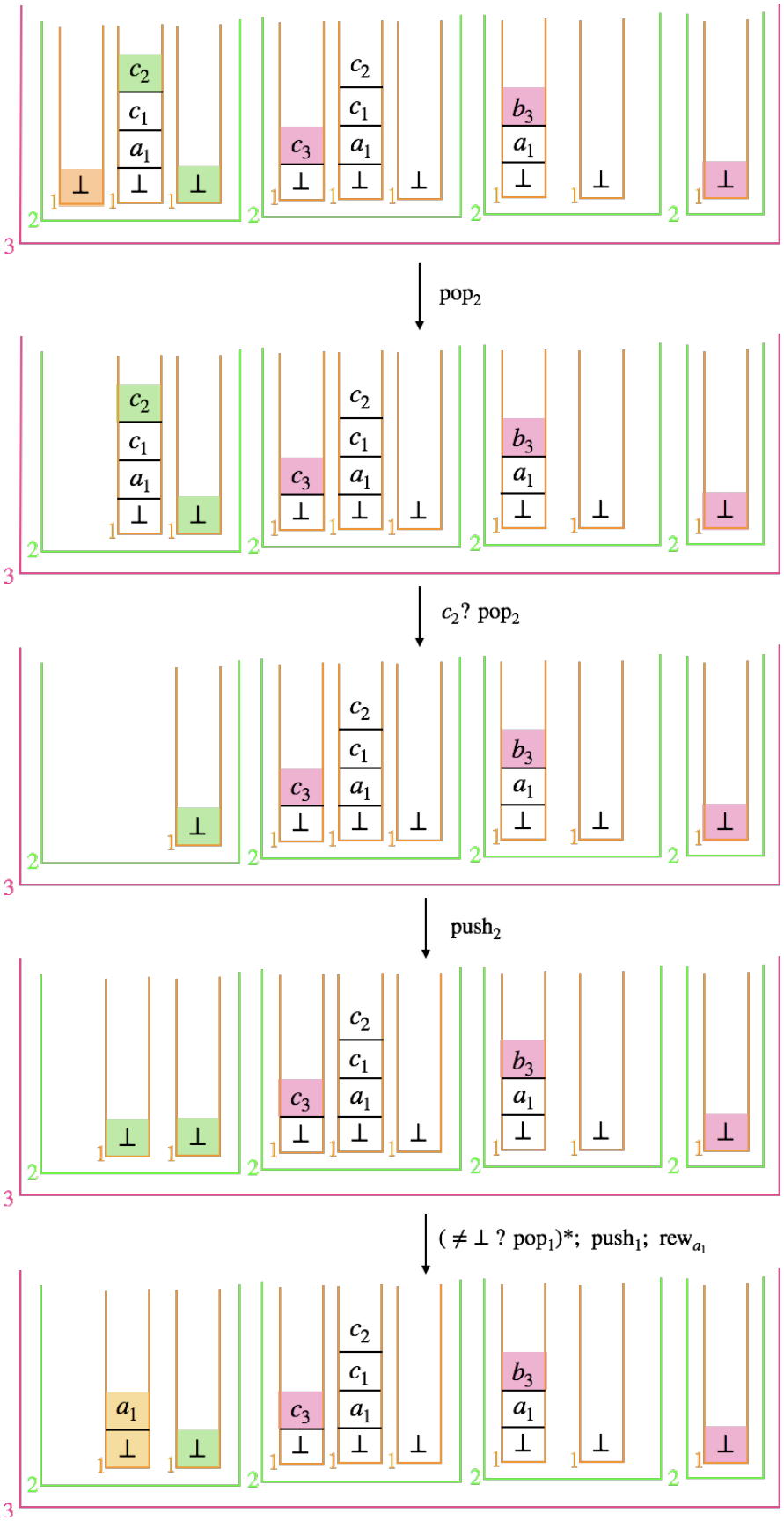


Figure 16 $< q_6, \perp > \rightarrow^{a}_{B} < q_7,\ \mathrm{pop}_2; c_2?;\ \mathrm{pop}_2;\ \mathrm{push}_2;\ \mathrm{push}_1; (\neq \perp?\ \mathrm{pop}_1)^*;\ \mathrm{push}_1;\ \mathrm{rew}_{a_1} >$

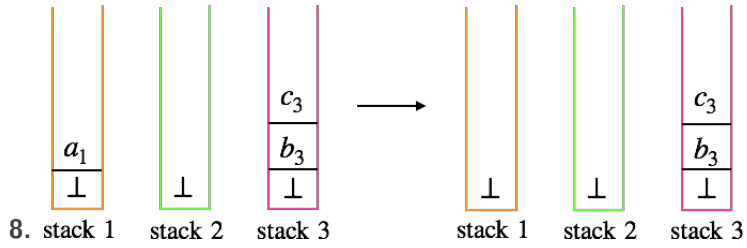


Figure 17 $< q_7, a_1, \epsilon, \epsilon > \rightarrow_A^a < q_8, \epsilon, \epsilon, \epsilon >$

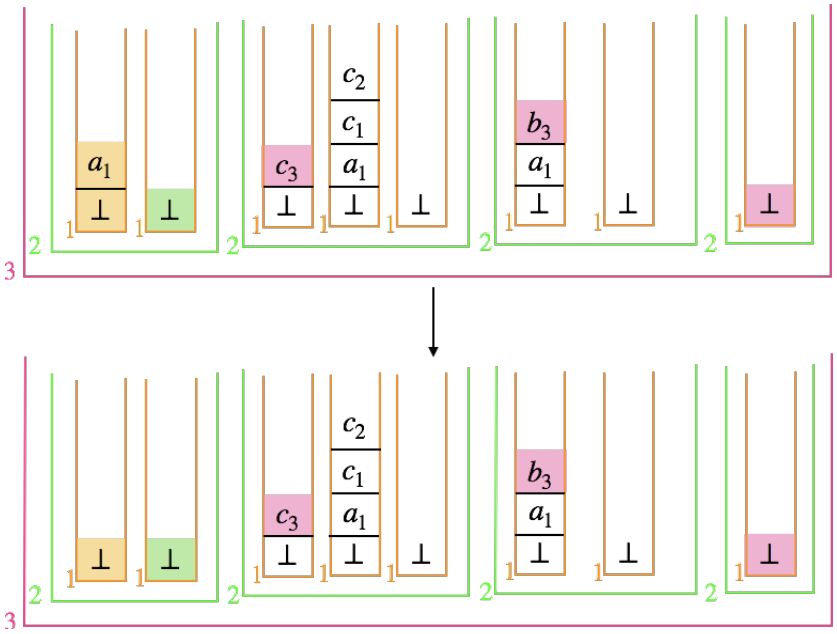


Figure 18 $< q_7, a_1 > \rightarrow_B^a < q_8, \ \mathrm{pop}_1 >$

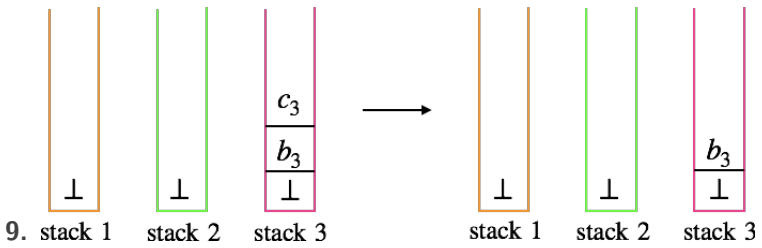
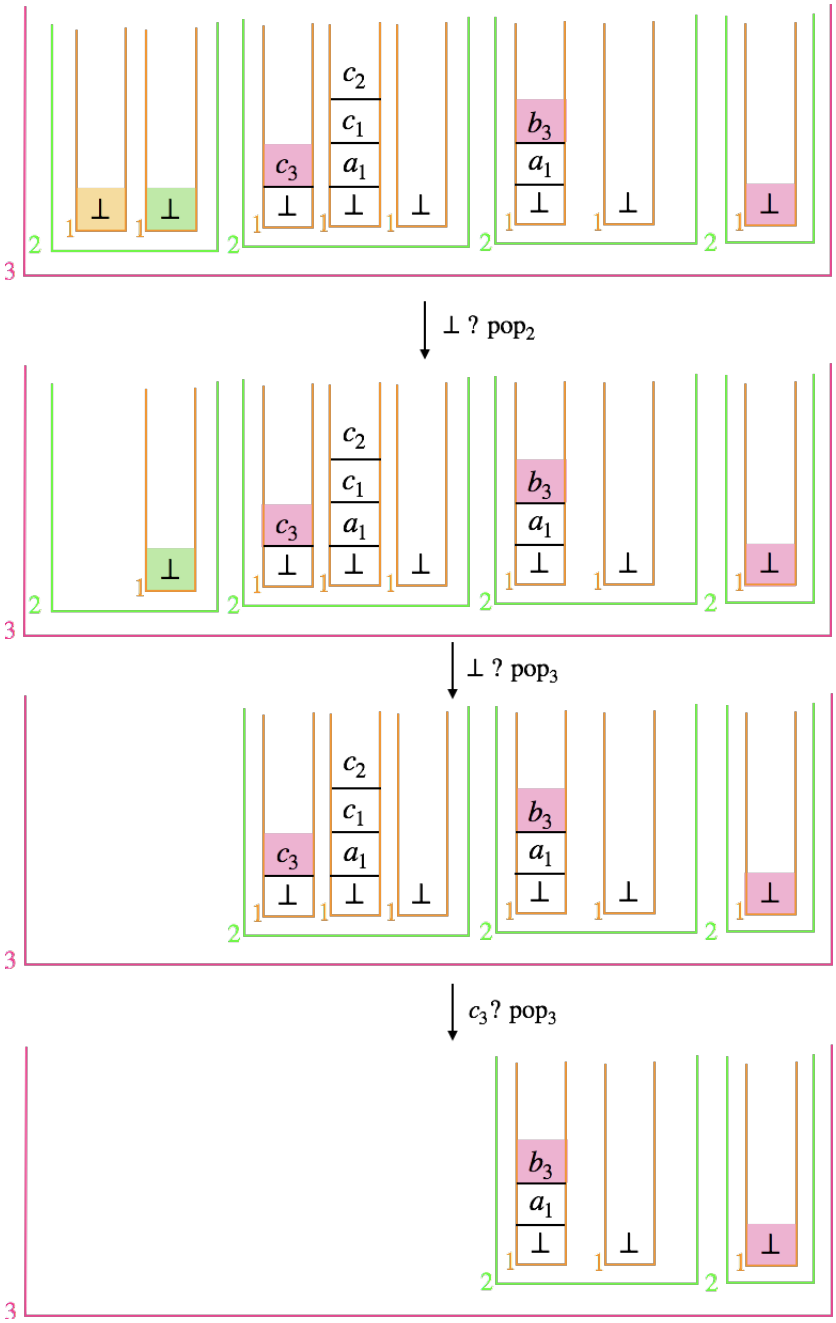


Figure 19 $< q_8, \perp, \perp, c_3 > \rightarrow_A^a < q_9, a_1, \epsilon, \epsilon >$


c_2
c_1
a_1
c_3
b_3
a_1
⊥
1
2
3
⊥ ? pop$_2$
⊥ ? pop$_3$
c_3? pop$_3$

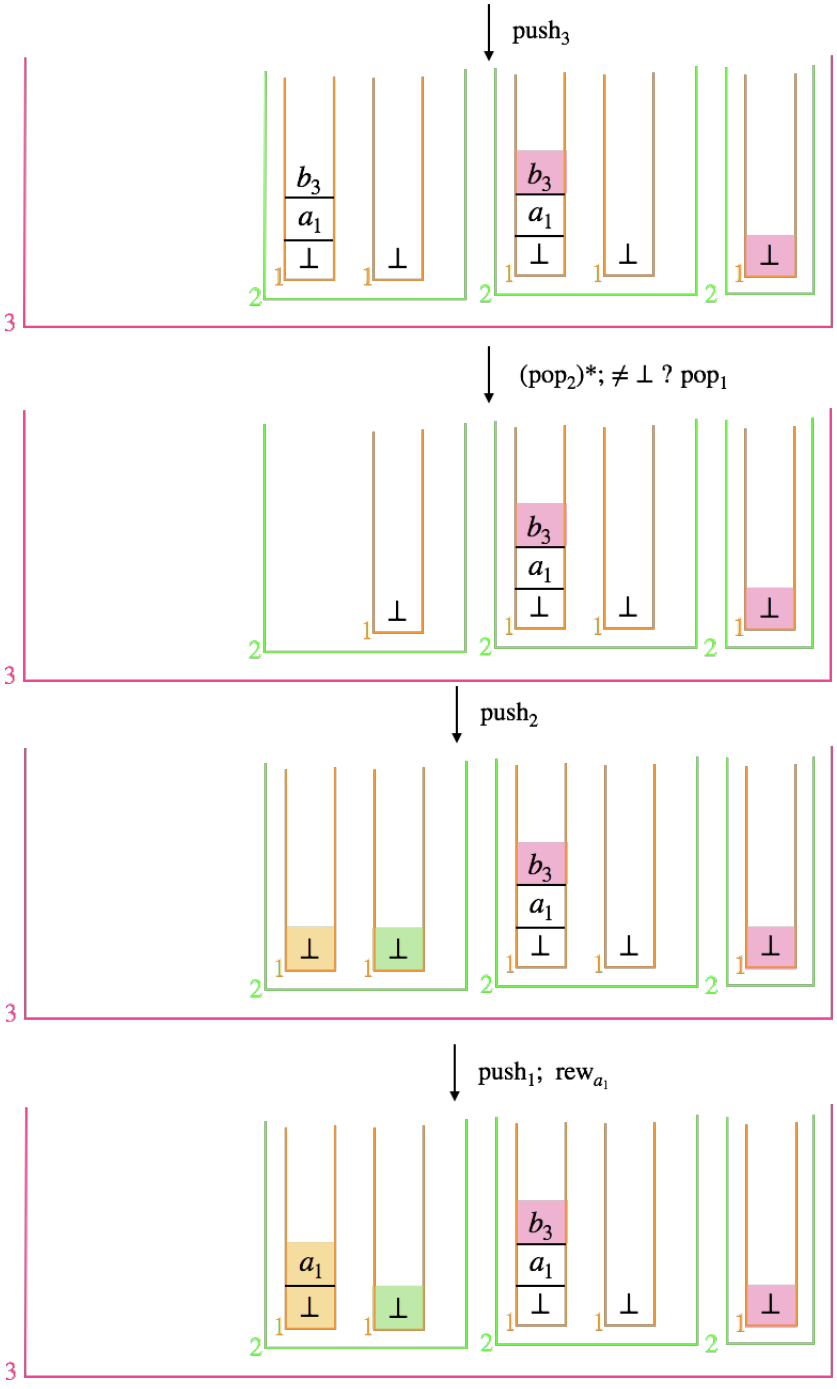


Figure 20 $< q_8, a_1 > \rightarrow_B^a < q_9,$ $\bot?$ $\text{pop}_2;$ $\bot?$ $\text{pop}_3;$ $c_3?$ $\text{pop}_3;$ $\text{push}_3;$ $(\text{pop}_2)^*; (\neq \bot?\ \text{pop}_1; \text{push}_2;\ \text{push}_1;\ \text{rew}_{a_1} >$

C Details from Section 4

C.1 PSPACE-hardness in Theorem 11

Our construction is inspired from [16, Lemma 8.6].

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a linear bounded TM (LBTM). Σ is the input alphabet, Γ is the tape alphabet (contains Σ and B and some extra symbols) and B stands for the blank symbol. M uses at most $p(n)$ space for the input of size n, where $p(n)$ is a polynomial in n. The maximum number of distinct configurations of M can be $max(|Q|, |\Gamma|)^{p(|w|)+1}$, say $k'_{\max}$, assuming a configuration is represented as $w_1 q w_2$ where $w_1 w_2$ is a word over Γ that covers the entire tape (thus, we also include the spaces occupied by the blank symbols).

Consider constants $k_{\max} = 2^{\lceil \log k'_{\max} \rceil}$ and $c_{\max} = \log k_{\max}$ to give precise bounds. Without loss of generality we assume that all halting computations of M on w take an even number of steps.

Given the LBTM M, we give the construction of PDAs P_1 and P_2 as follows. The input alphabet of P_1, P_2 is $\Gamma \cup Q \cup \{\#, \$\}$, where $\#, \$$ are new symbols. The stack alphabet for P_1, P_2 contain these symbols as well as symbols $0, 1, \#_1$. 0,1 are used to encode binary numbers to keep track of the number of configurations seen so far on any input. The idea is to restrict this number upto $k_{\max}$.

P_1 and P_2 both generate words of the form $c_1 \# c_2^R \# c_3 \# c_4^R \# \dots \# c_k^R \# \$^{c_{max}} \dots \# \$^{c_{max}} \#$ where number of #s $= k_{\max}$, satisfying the following conditions.

(1) each c_i is a valid configuration of M and of size exactly $c_{\max}$,
(2) In P_2, c_1 is an initial configuration of M on w i.e. $q_0 w B^{c_{\max}-(|w|+1)}$,
(3) In P_1, the last configuration before \$ is an accepting configuration,
(4) c_{i+1} is a valid successor configuration of c_i in P_1 for all odd i,
(5) c_{i+1} is a valid successor configuration of c_i in P_2 for all even i.

This way, correctness of consecutive odd and even configurations are guaranteed by P_2 and P_1 respectively and $\mathcal{L}(P_1) \cap \mathcal{L}(P_2)$ is the set of accepting computations of M on w padded with copies of word ($\$^{c_{\max}}\#$) at the right end. The length of the words accepted by P_1 and P_2 is k_{max} (number of #s) + $c_{max}.k_{max}$ (the size of each c_i or c_i^R is c_{max}, the padding with \$ also has length c_{max}. There are k_{max} of these c_i or padded \$.)

Construction of P_1.

(a) Initially we push $\log k_{\max}$ zeros (where $k_{\max}$ is the maximum possible number of configurations) and a unique symbol $\#_1$ in the stack to keep track of number of #s read so far. $\#_1$ is used as a separator in the stack between the counter values (encoded using 0,1) and the encoded configurations over Γ, Q. Thus, we initialize the value of the binary counter to $\log k_{\max}$ zeroes, with the separator $\#_1$ on top.

(b) Let L be $\{c_1 \# c_2^R \# \mid c_2$ is the successor configuration of c_1 in $M\}$. To realize L, we do the following. We read c_1 and check whether it is of the form $w_1 q w'_1$ where $w_1, w'_1 \in \Gamma^*$ and $q \in Q$. While reading c_1 upto q, we push w_1 on its stack. As soon as we find $q \in Q$ in c_1, we store q in the finite control state and read the next input symbol, say X.
- If X is #, then there is no successor configuration, and we reject the word denoting that we have reached the end of the tape.
- If $\delta(q, X) = (p, Y, R)$, then we push Yp onto the stack.
- If $\delta(q, X) = (p, Y, L)$, let Z be on top of stack, then we replace Z by pZY.

Again while reading w'_1, we push the symbols read onto the stack. Now the stack content will be (from bottom to top) the encoding of the counter, $\#_1$, and c_2. After reading the # after c_1, we compare each input symbol with the top stack symbol (now we are reading c_2^R in the input). If they differ, there is no next move. If they are equal, we pop the

top stack symbol. When the top of stack is $\#_1$, we read $\#$ after c_2^R and accept. While reading c_i (or c_i^R) we must also verify that the length of c_i is exactly $c_{\max}$. We can do this by adding polynomial size counter in the finite control state.

(c) We also need to increment the counter by 2 for the number of $\#$ (configurations) seen so far. We have read two configurations now in the above step. This can be done by popping the stack content starting from $\#_1$ until we find 0. During this popping, we store in the finite control, the information of how many 1s we have popped. As soon as we find 0 on the top of the stack, replace it with 1 and push back as many number of 1s as we popped previously (this info is stored in the finite control), followed by the separator $\#_1$. We need to do this step again to increment the counter since we saw two configurations (the two $\#$s represent this) in L.

To construct the PDA P_1, we follow steps of [(a)] i.e., initializing the counter. Next it iterates over steps of [(b)] followed by [(c)], thus reading two consecutive $c_i\#c_{i+1}^R\#$ and incrementing the counter by 2 until we find the configuration of the form $\Gamma^* F \Gamma^*$ or the counter becomes full. If we stop the iteration due to accepting configuration and counter is not full yet (counter is full when we have all 1s in the stack), we pad $\$^{c_{max}}\#$ until it becomes full. Otherwise if we stop the iteration due to the counter reaching $k_{\max}$ value before getting accepting configuration, we stop and reject this sequence of configurations.

Construction of P_2. Similarly, we can construct PDA P_2: The difference here is that it starts with initial configuration i.e. $q_0 w B^{c_{\max}-|w|}$ followed by $\#$ (here also we increment the counter by 1), then it will iterate over $\{c_1^R\#c_2\# \mid c_2 \text{ is the successor configuration of } c_1 \text{ in } M\}$ and increment the counter by 2 after each iteration. Then it will read accepting configuration non-deterministically (of course, followed by $\#$ and increment in the binary counter) and pad the string with $\$^{c_{max}}\#$ until counter becomes full.

By padding the words accepted with extra $\$^{c_{max}}\#$s we are enforcing that P_1 and P_2 accepts words of same length. Having the blank symbols B as part of the LBTM configuration is also motivated by this. Indeed, $w \in \mathcal{L}(P_1) \cap \mathcal{L}(P_2)$ iff the LBTM M accepts w. Since the words accepted by P_1, P_2 have same constant length, $\mathcal{L}(P_1) \cap \mathcal{L}(P_2) \neq \emptyset$ iff $\mathcal{L}(P_1)\uparrow \cap \mathcal{L}(P_2)\downarrow \neq \emptyset$.

C.2 Proof of Claim 14

Suppose $w \in L(G,A)\downarrow$, then w is labelled by symbols from $\alpha(L(G,A))$. Clearly $w \in (\alpha(L(G,A)))^*$.

Conversely, assume $w \in (\alpha(L(G,A)))^*$. We prove by induction on the length of w that $w \in L(G,A)\downarrow$. If the length of w is 1, then trivially one can generate any word in $L(G,A)$, which contains w as a subword. Assume the length of w to be $k+1$. By induction hypothesis, there exists a word $w' \in L(G,A)$ such that $w[:k] \preceq w'$ and w' is generated from A. We construct $w'' \in L(G,A)$ such that $w \preceq w''$. The derivation of w'' from A proceeds as follows. Start with the rule $A \Rightarrow^+ mAm'Am''$, and substitute the first A with w'. For the second A, substitute any word w''' s.t. $A \Rightarrow_G^+ w'''$ s.t. the $k+1$st symbol of w, $w[k+1]$ is contained in w'''. Then we obtain $w \preceq w''$, and hence $w \in L(G,A)\downarrow$.

C.3 Construction of G' and $A_{G\downarrow}$

The PDA $A_{G\downarrow}$ from G, G'. Now we construct the PDA $A_{G\downarrow} = (Q, T, N', \delta, \{q\}, S\downarrow, \{q\})$ from G' and G. This works in the standard way of converting CFGs to PDA, keeping in mind the following. When a_l (resp. a_r, a_{alph}) is the top of the stack, the PDA has a loop over symbols from $\alpha_l(A)$ (resp. $\alpha_r(A)$, $\alpha(L(G,A))$), in whichever state it is, at that time. The PDA stays in state q when the top of stack contains a non-terminal A. This non-terminal is

foreach *non-terminal* $A \in N$ **do**
  $\alpha_l(A) := \emptyset, \alpha_r(A) := \emptyset$;
end
foreach *non-terminal* $A \in N$ **do**
  if $A <_2 A$ **then**
    $A\downarrow \to A_{\text{alph}}$
  else
    foreach $B \in N$ *s.t.* $B =_1 A$ **do**
      foreach $p : B \to m$ **do**
        if $m = A_1A_2$ *for* $A_1 \neq_1 A$ *and* $A_2 \neq_1 A$ **then**
          $A_m \to A_1\downarrow A_2\downarrow$
        end
        if $m = a$ *for* $a \in T$ **then**
          $A_m \to a \mid \epsilon$
        end
        if $m = m'Cm''$ *for unique* $C =_1 A$ **then**
          $\alpha_l(A) := \alpha(L(G_2, m')) \cup \alpha_l(A), \quad \alpha_r(A) := \alpha(L(G_2, m'')) \cup \alpha_r(A)$
        end
      end
    end
    $A\downarrow \to A_l\ A_m\ A_r, \quad A_l \to a_l, \quad A_r \to a_r, \quad A_{\text{alph}} \to a_{\text{alph}}$.
  end
end

Algorithm 1: Construction of G'

popped the right hand side m of its production rule $A \to m$ is pushed on the stack. If the top of the stack is i) a terminal from T which also happens to be the next input symbol, then we just pop it ii) a terminal of the form a_{alph}, a_l or a_r, we move to state $q_{A_{\text{alph}}}$, q_{A_l} or q_{A_r} respectively and loop over $\alpha(L(G, A))$, $\alpha_l(A)$ or $\alpha_r(A)$ respectively and come back to q non-deterministically. The PDA accepts in q when the stack becomes empty.

Complexity. We analyze the complexity of constructing G' as well as $A_{G\downarrow}$.

- Construction of G':
  - for every $A, B \in N$, checking $A <_2 B$ reduces to checking membership of a_1a_2 in $L(G_1, B)$ where G_1 is obtained from G by replacing productions $C \to a$ of G with $C \to \epsilon$, $C \to a_1 \mid a_2$. G_1 is constructed in polynomial time from G. Similarly, we can check if $A <_1 B$ and $a \in \alpha(L(G, A))$.
  - for every $A \in N$, five non-terminals and three terminals are introduced. Hence, G' has $5|N|$ non-terminals and $|T| + 3|N|$ terminals, which is again polynomial in the input G'.
  - each step of the algorithm 1 can be computed in at most polynomial time and the algorithm has loops whose sizes range over terminals, non-terminals and production rules of G.
- PDA $A_{G\downarrow}$: Each non-terminal A has corresponding non-terminals A_l, A_r, A_{alph} or A_m in G'. Out of these, A_l, A_r and A_{alph} do not produce any further non-terminals while A_m produces non-terminals B where $B <_1 A$ and $B \neq_1 A$. Thus, the stack size is bounded above by $\mathcal{O}(|N|)$.

C.4 Proof of Lemma 12

Let $G = (N, T, R, S)$, we construct CFG G' which accepts subset of $\mathcal{L}(G)$ and includes $\mathsf{min}(\mathcal{L}(G))$. Clearly $\mathcal{L}(G)\uparrow = \mathcal{L}(G')\uparrow$. Let G' be $(N \times \{0, \ldots, |N|\}, T, R', (S, |N|))$ where production rules are defined as following:

- $(A, 0) \to a$ if $A \to a$
- $(A, i) \to (B, i-1)(C, i-1)$ for all $i \in [1, |N|]$ if $A \to BC$
- $(A, i) \to (A, i-1)$ for all $i \in [1, |N|]$

We can construct a PDA P for $\mathcal{L}(G')$ in polynomial time. It is easy to see that the PDA uses at most $|N| + 1$ height of the stack. It accepts all words of $\mathcal{L}(G)$ having derivation with at most $|N| + 1$ steps. We claim that $\mathcal{L}(G')$ contains $\mathsf{min}(\mathcal{L}(G))$. If not, then there exists $w \in \mathsf{min}(\mathcal{L}(G))$ with at least $|N| + 2$ derivation steps i.e. some non-terminal A appears at least two times in derivation tree say at level i and $j > i$. If we replace the subtree rooted at level j by subtree at level i, we get a proper subword of w (wlog we assume G is in CNF and it does not have useless symbols and $\epsilon-$productions), which is also in $\mathcal{L}(G)$, contradicting our assumption that $w \in \mathsf{min}(\mathcal{L}(G))$. Finally we can obtain a PDA for $\mathcal{L}(G)\uparrow$ by adding self loops on each state of P for each symbol $a \in \Sigma$ as $\mathcal{L}(G)\uparrow = \mathcal{L}(G')\uparrow$.

D Details for Section 4.2

Here we give details of missing proofs.

▶ **Lemma 24.** *Let R_1 and R_2 be sets of relations. R_1 and R_2 are* PosPTT *separable iff $R_1\uparrow \cap R_2 \neq \emptyset$.*

Proof. For the forward direction, assume R_1 and R_2 can be separated by PosPTT R_3 i.e. $R_1 \subseteq R_3$ and $R_2 \cap R_3 = \emptyset$. It is shown below that the set of relations defined by PosPTT is an upward closed set. The upward closure of R_3 is $R_3\uparrow = \{(u\uparrow, v\uparrow) \mid (u, v) \in R_3\}$. We first prove that any set of relations R defined by a PosPTT can be written as the union of a product of PosPTL i.e. $R = \bigcup_i (L_{1i} \times L_{2i})$, where L_{1i}, L_{2i} are PosPTLs. This can be proved inductively: simple relations are a product of PosPTL languages. Consider $R = R' \cap R''$. By inductive hypothesis, $R' = \bigcup_i (L_{1i} \times L_{2i})$ and $R'' = \bigcup_j (L'_{1j} \times L'_{2j})$, where L_{1i}, L_{2i}, L'_{1j} and L'_{2j} are all PosPTLs. Then $R = \bigcup_{i,j} (L_{1i} \cap L'_{1j}) \times (L_{2i} \cap L'_{2j})$. R is a PosPTT since $L_{1i} \cap L'_{1j}$ and $L_{2i} \cap L'_{2j}$ are PosPTLs. It is easy to see that the union of a product of PosPTL languages is upward closed, and hence R_3 is upward closed. Since R_3 is upward closed, $R_1\uparrow \subseteq R_3$ and thus $R_2 \cap R_1\uparrow = \emptyset$. For the reverse direction, we show that when $R_1\uparrow \cap R_2 = \emptyset$, $R_1\uparrow$ is a separator for R_1, R_2. $R_1\uparrow$ is definable by a PosPTT: $\mathsf{min}(R_1\uparrow)$ is a finite set since the subword ordering extended component wise is a wqo on $\Sigma^* \times \Gamma^*$. Let $\mathsf{min}(R_1\uparrow) = \{(u_1, v_1), \ldots, (u_m, v_m)\}$. Then $R_1\uparrow = \bigcup_{i=1}^{m} \{u_i\}\uparrow \times \{v_i\}\uparrow$. For a singleton set $\{u\}$, its upward closure is simply $\Sigma^* u_1 \Sigma^* \ldots u_k \Sigma^*$ where $u = u_1 \ldots u_k \in \Sigma^*$. Thus $R_1\uparrow$ is a finite union of the product of PosPTLs and therefore is a PosPTT. ◀