On Bisimilarity of Higher-Order Pushdown Automata: Undecidability at Order Two

Christopher Broadbent*[1] and Stefan Göller†[2]

1 Université Paris Diderot-Paris 7 and CNRS
Paris, France
christopher.broadbent@univ-paris-diderot.fr

2 Université Paris Diderot-Paris 7 and CNRS / University of Bremen
Paris, France / Bremen, Germany
goeller@informatik.uni-bremen.de

Abstract

We show that bisimulation equivalence of order-two pushdown automata is undecidable. Moreover, we study the *lower order problem* of higher-order pushdown automata, which asks, given an order-k pushdown automaton and some $k' < k$, to determine if there exists a reachable configuration that is bisimilar to some order-k' pushdown automaton. We show that the lower order problem is undecidable for each $k \geq 2$ even when the input k-PDA is deterministic and real-time.

Keywords and phrases Bisimulation equivalence, Higher-order pushdown automata

1 Introduction

Among the various notions of equivalence in concurrency theory [29] bisimulation equivalence (bisimilarity for short) is undoubtedly the central one in formal verification. For instance, elegant characterizations of the bisimulation-invariant fragments of well-known logics like first-order logic, monadic second-order logic or monadic path logic have been obtained in terms of modal logic [28], the modal μ-calculus [9], and CTL* [17], respectively.

The resulting decision problem, given two transition systems and a state of each of them, to decide whether the two states are bisimilar, is well-known to be complete for deterministic polynomial time on finite systems [2]. The status of decidability and of computational complexity of bisimilarity on infinite state systems is significantly less clear. A prominent such example is the class of systems described by pushdown automata (pushdown systems): Decidability was proven by Sénizergues [21], extending his famous decidability result on language equivalence of deterministic pushdown automata [20] (see [25] for a primitive recursive upper bound and [11] for a recent proof); however, the best known lower bound is EXPTIME [15]. Unfortunately, there are only few classes of infinite state systems, where bisimilarity is decidable and the precise complexity is known [10, 3].

Worse still, bisimilarity is not known to be decidable on PA-processes, ground tree rewrite systems, nor on higher-order pushdown systems, whereas its weaker variant (weak bisimilarity) is not known to be decidable on BPP nor on BPA, just to mention some prominent examples. We refer to [23] for an up-to-date record on the decidability and complexity status of bisimilarity of infinite state systems in Mayr's Process Rewrite Systems hierarchy.

* The author was supported by La Fondation Sciences Mathématiques de Paris. This work was also supported by by AMIS (ANR 2010 JCJC 0203 01 AMIS), FREC (ANR 2010 BLAN 0202 02 FREC) and VAPF (Région IdF)

† The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454.

© Christopher Broadbent and Stefan Göller;
licensed under Creative Commons License NC-ND

A milestone technique for proving lower bounds for bisimilarity on infinite state systems entitled "Defender's Forcing" has been introduced by Jančar and Srba [12]. Bisimulation equivalence can be seen as a game between "Attacker" and "Defender", where Defender wins the game if and only if the pair of states are bisimilar. When aiming at showing hardness of bisimilarity via reduction from a hard problem, the simple but powerful idea of "Defender's Forcing" is to construct, whenever necessary, a subgame that allows Defender to make a choice. This is important since, in a certain sense, Attacker has much more freedom in the bisimulation game, as in each round he is the one who chooses an outgoing transition of one of the two states, whereas Defender has to respond with a matching transition in the other system. With this technique lower bounds on various classes of infinite systems have been proven, for instance Σ_1^1-completeness of bisimilarity on prefix-recognisable graphs [12].

A further natural question is to decide whether a given infinite system is bisimilar to a finite one, the so-called *regularity* problem. It seems that even less is known about this problem, for instance, decidability is open for any class of systems that lies between pushdown systems (resp. PA-processes) and Mayr's class of Process Rewrite Systems [23]. However for deterministic pushdown automata decidability follows from [27, 24].

Higher-order pushdown automata were introduced by Maslov [16] and independently by Damm and Goerdt [7]. They generate the same class of trees as *safe* higher order recursion schemes [14], which have applications to software verification for higher-order programs.

To the best of the authors' knowledge the decidability status of bisimilarity on higher-order pushdown automata is open so far, but not explicitly stated as such in the literature. In our first result we show that bisimilarity of higher-order pushdown automata is already undecidable at order two. Inspired by [12] we show undecidability via a reduction from the infinitary version of the Modified Post's Correspondence Problem (MPCP). We must, however, restrict ourselves to instances of the infinitary MPCP for which all homomorphisms are non-erasing for our proofs to work (in contrast to [12]). To obtain this result, we use the above-mentioned technique "Defender's Forcing". Our undecidability result confirms that to some extent the decidability of bisimilarity of equational graphs with finite out-degree [21] cannot be significantly improved. It is worth mentioning that transition graphs of higher-order pushdown automata have finite out-degree and decidable monadic second-order theories [4, 5]. Deciding equivalence of deterministic order-k pushdown automata is an interesting open problem, although some progress has been made on this by Stirling [26].

In the second part of the paper, we study the *lower order* problem, which asks, given some configuration c of some order-k pushdown automaton and some $k' < k$, whether c can reach a configuration that is bisimilar to some configuration of some order-k' pushdown automaton. When $k' = k-1$ this question can be seen to ask whether the order-k pushdown automaton always exhibits behaviour that is 'inherently order-k'. When $k' = 0$, it is a weaker variant of the regularity problem, which asks whether a k-PDA is bisimilar to some finite state system. The property queried by the latter implies the property queried by the 'lower order problem' with $k' = 0$, but not *vice versa*.

We show that the lower order problem is undecidable whenever $k \geq 2$ even when the input order-k pushdown automaton is deterministic and real-time (i.e. free of ε-transitions).

2 Preliminaries

Transition systems and bisimulation equivalence. By $\mathbb{N} \stackrel{\text{def}}{=} \{1, 2, \ldots\}$ we denote the *naturals*. For each $n \in \mathbb{N}$ we define $[1, n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Let us fix a countable set of *actions* $\mathbb{A}$. A *labeled transition system* is a tuple $\mathcal{T} = (S, \mathsf{A}, \{\stackrel{a}{\longrightarrow} \mid a \in \mathsf{A}\})$, where S is a set of *states*,

$\mathsf{A} \subseteq \mathbb{A}$ is a finite set of *actions*, and $\xrightarrow{a} \subseteq S \times S$ is a binary *transition relation* for each $a \in \mathsf{A}$. A state $s \in S$ is called *deterministic* if for each $a \in \Sigma$ we have $|\{s' \in S \mid s \xrightarrow{a} s'\}] \leq 1$. We say $\mathcal{T}$ is *deterministic* if every state of $\mathcal{T}$ is deterministic. We naturally extend the transition relation $\xrightarrow{a}$ inductively to words, as follows: $\xrightarrow{\varepsilon} \stackrel{\text{def}}{=} \{(s,s) \mid s \in S\}$ and $\xrightarrow{wa} \stackrel{\text{def}}{=} \{(s,t) \mid \exists u \in S : s \xrightarrow{w} u \text{ and } u \xrightarrow{a} t\}$. We write $s \xrightarrow{w}$ in case $s \xrightarrow{w} s'$ for some state s', for each word $w \in \Sigma^*$. A binary relation $R \subseteq S \times S$ is called *bisimulation* if for each $(s_1, s_2) \in R$, for each $a \in \mathsf{A}$ and for each $i \in \{1,2\}$ we have that if $s_i \xrightarrow{a} s_i'$ for some $s_i' \in S$, then $s_{3-i} \xrightarrow{a} s_{3-i}'$ for some $s_{3-i}' \in S$ such that $(s_1', s_2') \in R$.

We write $s_1 \sim s_2$ if there is some bisimulation R with $(s_1, s_2) \in R$. It is also sometimes useful to think of bisimulation equivalence as a game played between *Attacker* and *Defender*. Starting from a pair of states (s_1, s_2) the game proceeds in rounds in which Attacker makes the first move by choosing some label a and some transition $s_i \xrightarrow{a} s_i'$ for some $i \in \{1,2\}$. Defender has to respond with some transition $s_{3-i} \xrightarrow{a} s_{3-i}'$ and the new game position is (s_1', s_2'). If one of the players cannot make an appropriate move, then the other player wins, and moreover Defender wins every infinite game.

Higher-order pushdown automata. Let us inductively define the set of *k-stacks*, for each $k \geq 1$, over some finite stack alphabet Γ with $[,] \notin \Gamma$ and where $\bot \notin \Gamma$ is a special *bottom-of-stack symbol*:

- A *1-stack* is an element of $\Gamma^*\bot$.
- A *$(k+1)$-stack* is a finite sequence $[\alpha_1][\alpha_2]\cdots[\alpha_n]$, where $n \geq 1$ and α_i is a k-stack for each $i \in [1,n]$.

Let us denote by $\mathsf{Stacks}_k(\Gamma)$ the set of all *k-stacks* over Γ. The *empty order k-stack* $\bot_k$ is inductively defined as $\bot_1 \stackrel{\text{def}}{=} \bot$ and $\bot_{k+1} \stackrel{\text{def}}{=} [\bot_k]$ for each $k \in \mathbb{N}$.

Over each 1-stack α we define the (partial) operation swap_w for each $w \in \Gamma^* \cup \Gamma^*\bot$ as

$$\mathsf{swap}_w(\alpha) \stackrel{\text{def}}{=} \begin{cases} wa_2\cdots a_n & \text{if } w \in \Gamma^*,\ \alpha = a_1\cdots a_n\bot,\ n \geq 1 \text{ and } a_i \in \Gamma \text{ for each } i \in [1,n], \\ w & \text{if } w \in \Gamma^*\bot \text{ and } \alpha = \bot, \text{ and} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\text{and} \quad \mathsf{top}_1(\alpha) \stackrel{\text{def}}{=} \begin{cases} a_1 & \text{if } \alpha = a_1\cdots a_n\bot \text{ with } n \geq 1 \text{ and } a_i \in \Gamma \text{ for each } i \in [1,n] \text{ and} \\ \bot & \text{otherwise.} \end{cases}$$

Let us define the partial operation $\mathsf{pop}_1(\alpha) \stackrel{\text{def}}{=} \mathsf{swap}_\varepsilon(\alpha)$ and for each k-stack $\alpha = [\alpha_1][\alpha_2]\cdots[\alpha_n]$ with $k \geq 2$ let us define:

$$\begin{aligned}
\mathsf{swap}_w(\alpha) &\stackrel{\text{def}}{=} [\mathsf{swap}_w(\alpha_1)][\alpha_2]\cdots[\alpha_n] \\
\mathsf{push}_k(\alpha) &\stackrel{\text{def}}{=} [\alpha_1][\alpha_1][\alpha_2]\cdots[\alpha_n] \\
\mathsf{push}_\ell(\alpha) &\stackrel{\text{def}}{=} [\mathsf{push}_\ell(\alpha_1)][\alpha_2]\cdots[\alpha_n] \quad \text{for each } 2 \leq \ell < k \\
\mathsf{pop}_k(\alpha) &\stackrel{\text{def}}{=} \begin{cases} [\alpha_2]\cdots[\alpha_n] & \text{if } n \geq 2 \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathsf{pop}_\ell(\alpha) &\stackrel{\text{def}}{=} [\mathsf{pop}_\ell(\alpha_1)][\alpha_2]\cdots[\alpha_n] \quad \text{for each } 2 \leq \ell < k \\
\mathsf{top}_k(\alpha) &\stackrel{\text{def}}{=} \alpha_1 \\
\mathsf{top}_\ell(\alpha) &\stackrel{\text{def}}{=} \mathsf{top}_\ell(\alpha_1) \quad \text{for each } 1 \leq \ell < k
\end{aligned}$$

Let $\mathsf{Op}_k \stackrel{\text{def}}{=} \{\mathsf{swap}_w \mid w \in \Gamma^* \cup \Gamma^*\bot\} \cup \{\mathsf{pop}_\ell \mid \ell \in [1,k]\} \cup \{\mathsf{push}_\ell \mid \ell \in [2,k]\}$ denote the set of *k-operations*. Note $\alpha \in \mathsf{Stacks}_k$ and $\mathsf{op} \in \mathsf{Op}_k$ implies $\mathsf{op}(\alpha) \in \mathsf{Stacks}_k$ if $\mathsf{op}(\alpha)$ is defined.

For each $k \geq 1$, an *order-k pushdown automaton (k-PDA)* is a tuple $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$, where Q is a finite set of *control locations*, $\mathsf{A} \subseteq \mathbb{A}$ is a finite set of actions, Γ is a finite *stack alphabet*, and where $\Delta \subseteq Q \times (\Gamma \cup \{\bot\}) \times \mathsf{A} \times Q \times \mathsf{Op}_k$ is a finite set of *stack rewrite rules*, where each $(q, x, a, q, \mathsf{op}) \in \Delta$ satisfies (i) $x = \bot$ and $\mathsf{op} = \mathsf{swap}_w$ implies $w \in \Gamma^*\bot$ and (ii) $x \in \Gamma$ and $\mathsf{op} = \mathsf{swap}_w$ implies $w \in \Gamma^*$. We abbreviate $(q, x, a, q', \mathsf{op}) \in \Delta$ by $qx \overset{a}{\hookrightarrow}_{\mathcal{P}} q'\mathsf{op}$.

The transition system of $\mathcal{P}$ is $\mathcal{T}(\mathcal{P}) \stackrel{\text{def}}{=} (Q \times \mathsf{Stacks}_k(\Gamma), \mathsf{A}, \{\xrightarrow{a}_{\mathcal{P}} \mid a \in \mathsf{A}\})$, where $(q,\alpha) \xrightarrow{a}_{\mathcal{P}} (q',\alpha')$ if there is $qx \overset{a}{\hookrightarrow}_{\mathcal{P}} q'\mathsf{op}$ in Δ such that $\mathsf{top}_1(\alpha) = x$ and $\alpha' = \mathsf{op}(\alpha)$ for each $q, q' \in Q$, each $a \in \mathsf{A}$ and each $\alpha, \alpha' \in \mathsf{Stacks}_k$. Thus, states of $\mathcal{T}(\mathcal{P})$ are elements of $Q \times \mathsf{Stacks}_k(\Gamma)$ that we also denote as *configurations of $\mathcal{P}$*. We call a configuration (q_0, α_0) of $\mathcal{P}$ *normed* if there exists some control location $q_f \in Q$ with $(q_f, \bot_k) \not\xrightarrow{a}$ (emits no a-transition) for each $a \in \Sigma$, and such that every configuration (q,α) with $(q_0,\alpha_0) \longrightarrow^* (q,\alpha)$ we have $(q,\alpha) \longrightarrow^* (q_f, \bot_k)$, where $\longrightarrow^*$ is the reflexive transitive closure of $\bigcup_{a\in\Sigma} \xrightarrow{a}$.
We can now state the decision problem which we study in the next section.

k-PDA-Bisimilarity

INPUT: A k-PDA $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$ and two configurations $(q,\alpha), (q',\alpha') \in Q \times \mathsf{Stacks}_k(\Gamma)$.

QUESTION: Does $(q,\alpha) \sim (q',\alpha')$ hold in $\mathcal{T}(\mathcal{P})$?

The following proposition is folklore and essentially follows from the fact that every configuration of a k-PDA has only finitely many successors.

▶ Proposition 1. The problem k-PDA-Bisimilarity is in Π^0_1 for each $k \geq 1$.

Post's Correspondence Problem and Variants of it. For two words u, v over some finite alphabet Σ we write $u \preceq v$ if $uw = v$ for some $w \in \Sigma^*$, that is if u is a prefix of v. For a word $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ for each $i \in [1,n]$ we denote its *reverse* by $w^R \stackrel{\text{def}}{=} a_n \cdots a_1$. For a finite (resp. infinite) sequence of finite words $u_1, \ldots, u_n$ (resp. $u_1, u_2, \ldots$) we write $\prod_{i\in[1,n]} u_i \stackrel{\text{def}}{=} u_1 u_2 \cdots u_n$ (resp. $\prod_{i\geq 1} u_i \stackrel{\text{def}}{=} u_1 u_2 \cdots$) to denote their concatentation.

An *instance* of *Post's Correspondence Problem* is given by a tuple $\mathcal{X} = ([1,n], \Sigma, h_1, h_2)$, where $n \in \mathbb{N}$, Σ is a finite *word alphabet*, and where $h_1, h_2 : [1,n]^* \to \Sigma^*$ are homomorphisms. We call $\mathcal{X}$ *increasing* if $|h_1(j)| \leq |h_2(j)|$ for each $j \in [1,n]$. We call $\mathcal{X}$ *non-erasing* if $h_1(j), h_2(j) \neq \varepsilon$ for each $j \in [1,n]$. A *solution* to $\mathcal{X}$ is a mapping $s : [1,m] \to [1,n]$ (equivalently a word $w \in [1,n]^*$), where $m \geq 1$ such that $s(1) = 1$ and moreover $h_1(s(1)\cdots s(m)) = h_2(s(1)\cdots s(m))$. An *$\omega$-solution* to $\mathcal{X}$ is a mapping $s : \mathbb{N} \to [1,n]$ with $s(1) = 1$ such that the following equality over ω-words holds: $\prod_{i\geq 1} h_1(s(i)) = \prod_{i\geq 1} h_2(s(i))$.

▶ Remark. When $\mathcal{X}$ is non-erasing and increasing, the following two statements are equivalent for each $s : \mathbb{N} \to [1,n]$:

- The mapping s is an ω-solution to $\mathcal{X}$.
- $s(1) = 1$ and $h_1(s(1)\cdots s(\ell)) \preceq h_2(s(1)\cdots s(\ell))$ for every $\ell \in \mathbb{N}$.

The classical problem MPCP asks, given an instance $\mathcal{X}$, whether $\mathcal{X}$ has a solution. The infinitary variant ω-MPCP asks, given an instance $\mathcal{X}$, whether $\mathcal{X}$ has an ω-solution.

It was shown in [19] that ω-MPCP is Π^0_1-complete. As already observed in [12], Sipser's Σ^0_1-hardness reduction [22] from the halting problem of Turing machines to MPCP can be transferred to a Π^0_1-hardness reduction to ω-MPCP even when restricting instances to be increasing (by only using Steps 1 to 5 and avoiding Steps 6 and 7 in Section 5.2 of [22]). In fact, by inspecting the homomorphisms constructed by Sipser, one can additionally assume that the instances are non-erasing; the latter was not necessary in the undecidability proofs from [12], but is essential in our hardness proofs. This leads us to the following:

ω-NonErasing-Increasing-MPCP

INPUT: An instance $\mathcal{X} = ([1,n], \Sigma, h_1, h_2)$ that is non-erasing and increasing, i.e. such that $h_1(j), h_2(j) \neq \varepsilon$ and $|h_1(j)| \leq |h_2(j)|$ for each $j \in [1,n]$.

QUESTION: Does $\mathcal{X}$ have an ω-solution?

▶ **Theorem 1** ([22]). *The problem* ω-NonErasing-Increasing-MPCP *is* Π_1^0*-complete.*

3 Undecidability of 2-PDA-Bisimilarity

We prove Π_1^0-hardness of 2-PDA-Bisimilarity by giving a many-to-one reduction from ω-NonErasing-Increasing-MPCP. For this, let us fix an instance $\mathcal{X} = (J, \Sigma, h_1, h_2)$ of ω-NonErasing-Increasing-MPCP. We will construct a 2-PDA $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$ and two configurations $(q, [1\bot])$ and $(q', [1\bot])$ such that $\mathcal{X}$ has an ω-solution if and only if $(q, [1\bot]) \sim (q', [1\bot])$ holds in $\mathcal{T}(\mathcal{P})$.

Overview of the Construction.

We start from the pair of configurations $(q, [1\bot])$ (the initial left configuration) and $(q', [1\bot])$ (the initial right configuration), thus both initial configurations consist of just one order-1 stack. We partition the bisimulation game into *three phases*.

Defining $j_1 \stackrel{\text{def}}{=} 1$, in **phase 1** we repeatedly push indices $j_2, j_3, \ldots \in [1,n]$ onto the order-1-stack of *both* configurations and we let Defender choose them by using the technique of "Defender's Forcing". The idea is that Defender's job is to push an infinite sequence of indices that is an ω-solution to $\mathcal{X}$ onto both order-1 stacks ad infinitum. At any situation in the game of the form $(q, [j_\ell \cdots j_1 \bot])$ and $(q', [j_\ell \cdots j_1 \bot])$ Attacker can play the action f to challenge Defender by claiming that $h_1(j_1 \cdots j_\ell)$ is not a prefix of $h_2(j_1 \cdots j_\ell)$.

This leads us to **phase 2** in which Defender wishs to prove $h_1(j_1 \cdots j_\ell) \preceq h_2(j_1 \cdots j_\ell)$. Let $w = j_\ell \cdots j_1 \bot$. We let the game get to the pair of configurations $(t, [w][w][w])$ and $(t', [w][w][w])$. From this position, by again using the "Defender's Forcing" technique and popping on the top-most order-1 stack, we allow Defender to choose a situation of the form $(x, [u^R j_{k-1} \cdots j_1 \bot][w][w])$ and $(x', [u^R j_{k-1} \cdots j_1 \bot][w][w])$, where $1 \leq k \leq \ell$, where u is a prefix of $h_2(j_k)$, and moreover $h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_{k-1})u$.

From the latter pair of configurations, **phase 3** deterministically prints from the left configuration essentially (plus some intermediate symbols) the string $h_1(j_1 \cdots j_\ell)^R$ by first performing a pop_2, and from the right configuration essentially (plus some intermediate symbols) the string $u^R h_2(j_1 \cdots j_{k-1})^R$ by continuing with the current top order-1-stack.

Since we had three copies of w at the end of phase two, we can now perform a pop_2 followed by a single 'wait' on the left configuration, and two pop_2s on the right configuration, so that both then have stack $[w]$. This allows them both to empty their stacks using the same number of pop_1 operations, allowing our 2-PDA to be *normed*. Thus our suggested generalisation of normedness standard for 1-PDA does not help recover decidability.

When describing the rules in detail, we list the rewrite rules of $\mathcal{P}$ in reverse order, i.e. first for **phase 3**, then for **phase 2** and finally for **phase 1**. Adapting the notation from [12], the rewrite rules that are presented in a $\square$ represent the moves added to implement "Defender's Forcing". These moves allow Defender to render the two configurations equal, and hence trivially bisimilar, if Attacker does not allow Defender to "decide the stack operations".

The Details.

Let $\Gamma \stackrel{\text{def}}{=} [1,n] \cup \Sigma$. The set of states Q, the set of actions A and the transitions Δ of $\mathcal{P}$ are implicitly given by the following rewrite rules. We describe the rules for **phase 3** first.

$x\ \gamma \overset{f}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{pop}_2$	and	$x'\ \gamma \overset{f}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{swap}_\gamma$	for each $\gamma \in \Gamma \cup \{\bot\}$
$y\ a \overset{a}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{pop}_1$	and	$y'\ a \overset{a}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{pop}_1$	for each $a \in \Sigma$
$y\ j \overset{a}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{swap}_{v^R}$			for each $j \in [1,n]$, where $h_1(j) = va$
		$y'\ j \overset{a}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{swap}_{v^R}$	for each $j \in [1,n]$, where $h_2(j) = va$
$y \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} z_1\ \mathsf{swap}_\bot$	and	$y' \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} z_1'\ \mathsf{pop}_2$	
$z_1 \bot \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_2$	and	$z_1'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_2$	for each $j \in [1,n]$
$z\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_1$			for each $j \in [1,n]$

For the following lemma, observe that from both the initial configurations in the lemma there is a unique maximal (w.r.t. $\preceq$) word that can be traced.

▶ **Lemma 2.** *Assume $j_1, \cdots, j_\ell \in [1,n]$ with $\ell \geq 1$ and let $0 \leq k \leq \ell$. Assume some 2-stack $\alpha = [u^R j_k \cdots j_1 \bot][j_\ell \cdots j_1 \bot][j_\ell \cdots j_1 \bot]$, where $u \in \Sigma^*$. Then we have*

$$(x, \alpha) \sim (x', \alpha) \quad \textit{if and only if} \quad h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_k)u.$$

Let us add the following rules to Δ in order to implement **phase 2**.

$r_1\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} r_2\ \mathsf{push}_2$	and	$r_1'\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} r_2'\ \mathsf{push}_2$	for each $j \in [1,n]$
$r_2\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} t\ \mathsf{push}_2$	and	$r_2'\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} t'\ \mathsf{push}_2$	for each $j \in [1,n]$
$t \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_\bot$	and	$t' \bot \overset{\bot}{\hookrightarrow}_{\mathcal{P}} x'\ \mathsf{swap}_\bot$	
$\boxed{t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_{\downarrow}\ \mathsf{swap}_j}$	and	$t'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_{\downarrow}\ \mathsf{swap}_j$	for each $j \in [1,n]$
$t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} \bar{t}\ \mathsf{swap}_j$			for each $j \in [1,n]$
$\boxed{t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_j(w)\ \mathsf{swap}_j}$	and	$t'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_j(w)\ \mathsf{swap}_j$	for each $j \in [1,n]$ and each prefix w of $h_2(j)$
$\bar{t}\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}}\ t\ \mathsf{pop}_1$	and	$t'_{\downarrow}\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}}\ t'\ \mathsf{pop}_1$	for each $j \in [1,n]$
		$\boxed{t'_j(w)\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}} t\ \mathsf{pop}_1}$	for each $j \in [1,n]$ and each prefix w of $h_2(j)$
$\bar{t}\ j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{w^R}$	and	$t'_j(w) j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x'\ \mathsf{swap}_{w^R}$	for each $j \in [1,n]$ and each prefix w of $h_2(j)$
	and	$\boxed{t'_{\downarrow}\ j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{w^R}}$	
		$\boxed{t'_j(w)\ j \overset{\langle j,v\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{v^R}}$	for each $j \in [1,n]$ and all prefixes v, w of $h_2(j)$ s.t. $v \neq w$

▶ **Lemma 3.** *Let $\alpha = j_1 \cdots j_\ell \in [1,n]^\ell$ with $\ell \geq 1$. Then we have*

$$(r_1, [\alpha^R \bot]) \sim (r_1', [\alpha^R \bot]) \quad \textit{if and only if} \quad h_1(\alpha) \preceq h_2(\alpha).$$

Proof. When inspecting the first two pairs of rules from the previous block, it is clear that $(r_1, [\alpha^R \bot]) \sim (r_1', [\alpha^R \bot])$ if and only if $(t, [\alpha^R \bot][\alpha^R \bot][\alpha^R \bot]) \sim (t', [\alpha^R \bot][\alpha^R \bot]\alpha^R \bot])$. Define the configuration $\alpha_k \overset{\text{def}}{=} [j_k \cdots j_1 \bot][\alpha^R \bot][\alpha^R \bot]$ for each $0 \leq k \leq \ell$.

"If": Assume $h_1(\alpha) \preceq h_2(\alpha)$. By assumption we have $h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_{k_0 - 1})u$, where $1 \leq k_0 \leq \ell$ and where $u \in \Sigma^*$ is some non-empty prefix of $h_2(j_{k_0})$. First, we have that

there exists some bisimulation relation R that relates $(x, [u^R j_{k_0-1} \cdots j_1 \bot][\alpha^R \bot][\alpha^R \bot])$ and $(x', [u^R j_{k_0-1} \cdots j_1 \bot][\alpha^R \bot][\alpha^R \bot])$ by Lemma 2. For each configuration $\zeta \in Q \times \mathsf{Stacks}_2(\Gamma)$, let us define $\mathsf{cl}(\zeta) \stackrel{\text{def}}{=} \{(\zeta', \zeta') \mid \exists v \in \Sigma^* : \zeta \xrightarrow{v} \zeta'\}$. The reader verifies that the symmetric closure of the following relation is indeed a bisimulation that relates $(t, [\alpha^R \bot][\alpha^R \bot][\alpha^R \bot]) = (t, \alpha_\ell)$ and $(t', [\alpha^R \bot][\alpha^R \bot]\alpha^R \bot]) = (t', \alpha_\ell)$:

$$
\begin{array}{llll}
 & \{(\langle t, \alpha_k\rangle, \langle t', \alpha_k\rangle) \mid k_0 \le k \le \ell\} & \cup & \{(\langle \overline{t}, \alpha_k\rangle, \langle t'_\downarrow, \alpha_k\rangle) \mid k_0 < k \le \ell\} \\
\cup & \bigcup_{k_0 < k \le \ell, w \preceq h_2(k)} \mathsf{cl}(\langle t'_k(w), \alpha_k\rangle) & \cup & \bigcup_{w \preceq h_2(k_0), w \neq u} \mathsf{cl}(\langle t'_{k_0}(w), \alpha_{k_0}\rangle) \\
\cup & \{(\langle \overline{t}, \alpha_{k_0}\rangle, \langle t'_{k_0}(u), \alpha_{k_0}\rangle)\} & & \\
\cup & \mathsf{cl}(\langle t, \alpha_{k_0-1}\rangle) & \cup & R
\end{array}
$$

"Only if": Assume $h_1(j_1 \cdots j_\ell) \not\preceq h_2(j_1 \cdots j_\ell)$. First, recall that we have

$$(x, [u^R j_{k-1} \cdots j_1 \bot][\alpha^R \bot][\alpha^R \bot]) \not\sim (x', [u^R j_{k-1} \cdots j_1 \bot][\alpha^R \bot][\alpha^R \bot]) \tag{1}$$

for every $k \in [1, \ell]$ and each prefix u of $h_2(j_k)$ by Lemma 2.

For proving $(t, [\alpha\bot][\alpha\bot][\alpha\bot]) \not\sim (t', [\alpha\bot][\alpha\bot][\alpha\bot])$ we will show $(t, \alpha_k) \not\sim (t', \alpha_k)$ for each $0 \le k \le \ell$ by induction on k.

Induction base. We have to prove $(t, [\bot][\alpha\bot][\alpha\bot]) \not\sim (t', [\bot][\alpha\bot][\alpha\bot])$. When performing action $\bot$ on both configurations we obtain the pair $(x, [\bot][\alpha\bot][\alpha\bot])$ and $(x', [\bot][\alpha\bot][\alpha\bot])$ which is not bisimilar by Lemma 2 since $|\alpha| \ge 1$ and h_2 is non-erasing by assumption.
Induction step. Let $0 \le k < \ell$. Consider the bisimulation game starting from the pair of configurations (t, α_{k+1}) and (t', α_{k+1}). We will describe a winning strategy for Attacker. Attacker plays $(t, \alpha_{k+1}) \xrightarrow{p} (\overline{t}, \alpha_{k+1})$. We make a case distinction on the answers of Defender. Firstly, assume Defender responds $(t', \alpha_{k+1}) \xrightarrow{p} (t'_\downarrow, \alpha_{k+1})$. In the next round, Attacker plays $(t'_\downarrow, \alpha_{k+1}) \xrightarrow{\downarrow} (t', \alpha_k)$ and Defender can only respond with $(\overline{t}, \alpha_{k+1}) \xrightarrow{\downarrow} (t, \alpha_k)$. For the resulting pair of configurations it holds $(t, \alpha_k) \not\sim (t', \alpha_k)$ by induction hypothesis. Secondly, assume Defender responds $(t', \alpha_{k+1}) \xrightarrow{p} (t'_{j_{k+1}}(w), \alpha_{k+1})$ for some $w \preceq h_2(j_{k+1})$. Then Attacker can play $(t'_{j_{k+1}}(w), \alpha_{k+1}) \xrightarrow{\langle j_{k+1}, w\rangle} (x', [w^R j_k \cdots j_1][\alpha\bot][\alpha\bot])$ and Defender can only respond with $(\overline{t}, \alpha_{k+1}) \xrightarrow{\langle j_{k+1}, w\rangle} (x, [w^R j_k \cdots j_1][\alpha\bot][\alpha\bot])$ and the resulting pair of configurations is not bisimilar by (1). ◀

Finally, let us add the following rules to Δ for implementing **phase 1**.

$$
\begin{array}{llll}
\boxed{q\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} q'_j\ \mathsf{swap}_k} \quad \text{and} \quad q'\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} q'_j\ \mathsf{swap}_k & & & \text{for each } j, k \in [1, n] \\
q\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} \overline{q}\ \mathsf{swap}_k & & & \text{for each } k \in [1, n] \\
\overline{q}\ k \overset{j}{\hookrightarrow}_{\mathcal{P}} q\ \mathsf{swap}_{jk} & & & \text{for each } k, j \in [1, n] \\
 & q'_j\ k \overset{j}{\hookrightarrow}_{\mathcal{P}} q'\ \mathsf{swap}_{jk} & & \text{for each } k, j \in [1, n] \\
 & \boxed{q'_j\ k \overset{j'}{\hookrightarrow}_{\mathcal{P}} q\ \mathsf{swap}_{j'k}} & & \text{for each } k, j, j' \in [1, n] \text{ with } j' \neq j \\
q\ k \overset{f}{\hookrightarrow}_{\mathcal{P}} r_1\ \mathsf{swap}_k \quad \text{and} \quad q'k \overset{f}{\hookrightarrow}_{\mathcal{P}} r'_1\ \mathsf{swap}_k & & & \text{for each } k \in [1, n]
\end{array}
$$

▶ **Lemma 4.** *We have* $(q, [1\bot]) \sim (q', [1\bot])$ *if and only if* $\mathcal{X}$ *has an* ω*-solution.*

One can easily verify that both configurations $(q, [1\bot])$ and $(q', [1\bot])$ are normed. For the following theorem, the lower bound follows from Theorem 1 and and Lemma 4, whereas the upper bound follows from Proposition 1.

▶ **Theorem 5.** *The problem* 2-PDA-Bisimilarity *is* Π^0_1*-complete, even when both input configurations are normed.*

4 The Lower Order Problem

We consider the following class of *lower order* problems for $k \geq 2$ and $k' \in [0, k-1]$ (where a 0-PDA is a finite automaton):

LO$_{k,k'}$

INPUT: A *deterministic* k-PDA $\mathcal{P}$ and a configuration (q, α) of $\mathcal{P}$.

QUESTION: Does there exist a configuration (r, β) of $\mathcal{P}$ with $(q, \alpha) \longrightarrow^* (r, \beta)$ such that $(r, \beta) \sim (r', \beta')$, where (r', β') is a configuration of some k'-PDA $\mathcal{P}'$?

This problem is related to determining whether a program employing order-k functions can ever reach a state from which it could continue using code only constructed with order-k' functions. The case when $k' = 0$ is thus related to the problem of determining whether a (higher-order) recursive program is equivalent to one using only constant memory.

The following holds whether or not $\mathcal{P}'$ is restricted to being deterministic:

▶ **Theorem 6.** **LO$_{k,k'}$** *is undecidable for every* $k \geq 2$ *and* $k' \in [0, k-1]$.

To our knowledge the decidability of **LO$_{1,0}$** and the variant of **LO$_{k,k'}$** for which we require $(r, \beta) = (q, \alpha)$ remain open when $k' \geq 1$ (and $k' \geq 0$ if $\mathcal{P}$ is allowed to be non-deterministic). These stronger problems would be even more pertinent to practical applications.

4.1 Language Recognition

Our construction requires us to use k-PDA as *language recognisers*. In order to do this we extend the model further to: $\mathcal{R} = (Q, q_0, F, \mathsf{A}, \Gamma, \Delta)$ where $q_0 \in Q$ is an *initial control location* and $F \subseteq Q$ is a set of *accepting control locations*. The *language recognised by* $\mathcal{R}$ is:

$$\mathcal{L}(\mathcal{R}) \stackrel{\text{def}}{=} \{\, w \in \mathsf{A}^* \mid (q_0, \bot_k) \xrightarrow{w} (q, \alpha) \text{ for some } q \in F \text{ and } \alpha \in \mathsf{Stacks}(\Gamma) \,\}$$

For simplicity, we allow $\varepsilon \in \mathsf{A}$ and for language recognisers view this as a silent transition. A *reachable* configuration (q, α) is one such that $(q_0, \bot_k) \xrightarrow{w} (q, \alpha)$ for some $w \in \Sigma^*$.

We say that the language $\mathcal{L} \subseteq \Sigma^*$ is k*-complete* if it is recognised by a *deterministic* k-PDA but not by any $(k-1)$-PDA (whether deterministic or non-deterministic). A k-complete language exists for every $k \geq 1$ [6, 7, 16].

In the spirit of [1], we say that a language is *visibly* k*-complete* for $k \geq 2$ if it is k-complete and we can partition A into $\mathsf{A} = \mathsf{A}^{\uparrow} \uplus \mathsf{A}^{<} \uplus \mathsf{A}^{\downarrow}$ such that it is recognised by a *visible* k-PDA $\mathcal{R}$. A k-PDA $\mathcal{R}$ is *visible* if it has a transition of the form $q\gamma \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{op}$ only if $a \in \mathsf{A}^{\uparrow}$ implies $\mathsf{op} = \mathsf{push}_k$, $a \in \mathsf{A}^{\downarrow}$ implies $\mathsf{op} = \mathsf{pop}_k$ and $a \in \mathsf{A}^{<}$ implies $\mathsf{op} \in \mathsf{Op}_{k-1}$. Thus the language is 'marked' with the order-k pushes and pops performed by some k-PDA recognising it. It is clear that the existence of k-complete languages implies the existence of visibly k-complete languages (pick a k-complete language and some k-PDA recognising it, then convert the k-PDA to recognise a visibly k-complete language by marking the symbols labelling push_k and pop_k transitions—any $(k-1)$-PDA generating the annotated language could also generate the original by removing the annotations).

Let us further say that a deterministic k-PDA $\mathcal{R}$ is *total* if $\mathcal{R}$ is real-time (i.e. without ε-transitions) and for every reachable configuration (q, α) and every $a \in \mathsf{A}$, there exists a transition of the form $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{op}$ such that $\mathsf{op}(\alpha)$ is defined.

We say that a deterministic visible k-PDA $\mathcal{R}$ recognising a visibly k-complete language is *visibly-total* if $\mathcal{R}$ is real-time and for every reachable configuration (q, α) and every $a \in \mathsf{A}^{<}$ there exists a transition of the form $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{op}$ such that $\mathsf{op}(\alpha)$ is defined, for every $a \in \mathsf{A}^{\uparrow}$ a transition $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{push}_k$ and for every $a \in \mathsf{A}^{\downarrow}$ there is a transition of the form $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{pop}_k$ (whether or not $\mathsf{pop}_k(\alpha)$ is defined).

▶ **Lemma 7.** *For every $k \geq 1$ there exists a (visibly) k-complete language that is recognised by a deterministic (visibly)-total k-PDA.*

4.2 Outline of the undecidability of $\mathsf{LO}_{k,k'}$ with $k \geq 2$

In all cases we work by a reduction from (the finitary variant of) MPCP. We fix an MPCP instance $\mathcal{X} = ([1,n], \Sigma, h_1, h_2)$ and for each $\mathsf{LO}_{k,k'}$ we construct a k-PDA $\mathcal{P}_{\mathcal{X}}$ such that there is a solution to the lower order problem if and only if there is a solution to $\mathcal{X}$. We have two separate constructions, one for the case $k - k' \geq 2$ and one for the case $k' = k-1$. First, let us assume $k - k' \geq 2$. The idea is that for each potential solution w to $\mathcal{X}$ there is a configuration c_w that is reachable from a configuration $(q_0, \bot_k)$ whose k-stack is of the form $[\alpha][\mathsf{swap}_w(\bot_{k-1})]$ for some $(k-1)$-stack α. From c_w the deterministic k-PDA $\mathcal{P}_{\mathcal{X}}$ can simulate on its stack α some deterministic $(k-1)$-PDA recognizing some $(k-1)$-complete language. However, at any point $\mathcal{P}_{\mathcal{X}}$ threatens to pop $[\alpha]$ and then deterministically output $h_1(w)^R$ in case the current control location of the simulation is in some accepting control location or else to deterministically output $h_2(w)^R$ in case the current control location of the simualation is in some non-accepting control location. In case $h_1(w) = h_2(w)$ the distinction between accepting and rejecting control locations cannot be made by observing this behaviour and so there will be a finite automaton bisimilar to $\mathcal{P}_{\mathcal{X}}$.

Let us now assume $k' = k - 1$. We follow a similar idea, but this time simulate an automaton for a k-complete language. However this simulation may lead to k-stacks of the form $[\alpha_m] \cdots [\alpha_1][\mathsf{swap}_w(\bot_{k-1})]$ for arbitrarily large m. This means that to carry out the threat of printing $h_1(w)^R$ or $h_2(w)^R$ it is necessary to make an unbounded number of pop_k operations. If w is a solution to $\mathcal{X}$ so that $h_1(w) = h_2(w)$ then a bisimilar $\mathcal{P}'$ will need to 'know' m in order to synchronise with this preparation to print $h_1(w)$ or $h_2(w)$. It thus needs to be equipped with a counter, but since $k - 1 \geq 1$, this is possible. Concerning the proof, there is a simple modification exploiting visibly total automata recognising k-complete visible languages, thus we have decidided to relegate this case to the long version.

4.3 The undecidability of $\mathsf{LO}_{k,k'}$ for $k \geq 2$ and $k - 2 \geq k' \geq 0$.

We fix a deterministic $(k-1)$-PDA $\mathcal{R} = (Q_{\mathcal{R}}, q_{0\mathcal{R}}, F_{\mathcal{R}}, \mathsf{A}_{\mathcal{R}}, \Gamma_{\mathcal{R}}, \Delta_{\mathcal{R}})$ that recognises a $(k-1)$-complete language. Due to Lemma 7 we may assume that it is *total-deterministic*. We do not require any visibility assumption in this section. We construct a k-PDA $\mathcal{P}_{\mathcal{X}} = (Q, \mathsf{A}, \Gamma, \Delta)$ and take a configuration $(q_0, \bot_k)$ as input to the problem. We will progressively introduce Δ (and thereby Q, A, and Γ).

First we have rules responsible for 'guessing' a solution to MPCP and spawning $\mathcal{R}$:

$$
\begin{array}{llll}
q_0 \bot \overset{\#_1}{\hookrightarrow}_{\mathcal{P}_{\mathcal{X}}} q_0\ \mathsf{swap}_{1\bot} & \text{and} & q_0\ j \overset{\#_{j'}}{\hookrightarrow}_{\mathcal{P}_{\mathcal{X}}} q_0\ \mathsf{swap}_{j'j} & \text{for each } j, j' \in [1,n] \\
q_0\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_{\mathcal{X}}} q_0'\ \mathsf{swap}_j & \text{and} & q_0'\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_{\mathcal{X}}} q_0''\ \mathsf{push}_k & \text{for each } j \in [1,n] \\
q_0''\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_{\mathcal{X}}} q_0''\ \mathsf{pop}_1 & \text{and} & q_0'' \bot \overset{\#}{\hookrightarrow}_{\mathcal{P}_{\mathcal{X}}} q_{0\mathcal{R}}\ \mathsf{swap}_{\bot} & \text{for each } j \in [1,n]
\end{array}
$$

We then add every rule in $\Delta_{\mathcal{R}}$ to Δ. Finally we have rules responsible for printing out the image of the alleged solution under either h_1 or h_2:

$q_f\ \gamma \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1^-\ \mathsf{pop}_k$	and	$q\ \gamma \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2^-\ \mathsf{pop}_k$	for each $\gamma \in \Gamma_{\mathcal{R}}$, $q_f \in F_{\mathcal{R}}$, $q \in Q_{\mathcal{R}} - F_{\mathcal{R}}$
$y_1^-\, j \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1\ \mathsf{push}_k$	and	$y_2^-\, j \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2\ \mathsf{push}_k$	for each $j \in [1,n]$
$y_1\ a \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1\ \mathsf{pop}_1$	and	$y_2\ a \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2\ \mathsf{pop}_1$	for each $a \in \Sigma$
$y_1\ j \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1\ \mathsf{swap}_{v_1^R}$	and	$y_2\ j \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2\ \mathsf{swap}_{v_2^R}$	f. e. $j \in [1,n]$: $h_1(j) = v_1 a$, $h_2(j) = v_2 a$
$y_1 \perp \overset{\mathsf{reset}}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0'\ \mathsf{pop}_k$	and	$y_2 \perp \overset{\mathsf{reset}}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0'\ \mathsf{pop}_k$	

We have borrowed the rules for y_1 and y_2 from the previous construction, and so from the proof of Lemma 2 we get:

▶ **Lemma 8.** *Suppose that we have a stack* $\alpha = [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]\ [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *where* $j_1, \ldots, j_\ell \in [1,n]$. *Then we have* $(y_i, \alpha) \xrightarrow{u\mathsf{reset}} (q_0', \mathsf{pop}_k(\alpha))$ *if and only if* $u = h_i(j_1 \cdots j_\ell)^R$, *for each* $i \in \{1,2\}$.

We have the following characterisation of reachable configurations of $\mathcal{P}_\mathcal{X}$:

▶ **Lemma 9.** *Let* (p,α) *be a configuration of* $\mathcal{P}_\mathcal{X}$ *with* $(q_0, \perp_k) \longrightarrow^* (p,\alpha)$. *Then one of the following holds:*

- $p \in \{q_0, q_0', y_1^-, y_2^-\}$ *and* $\alpha = [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1, \ldots, j_\ell \in [1,n]$ *with* $j_1 = 1$.
- $p = q_0''$ *and* $\alpha = [\mathsf{swap}_{j_{\ell'} \cdots j_1 \perp}(\perp_{k-1})]\ [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1, \ldots, j_\ell \in [1,n]$ *with* $j_1 = 1$ *and* $0 \le \ell' \le \ell$.
- $p \in \{y_1, y_2\}$ *and* $\alpha = [\mathsf{swap}_{w j_{\ell'} \cdots j_1 \perp}(\perp_{k-1})]\ [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1, \ldots, j_\ell \in [1,n]$ *with* $j_1 = 1$ *and* $0 \le \ell' \le \ell$ *and* $w \in \Sigma^*$.
- $p \in Q_{\mathcal{R}}$ *and* $\alpha = [\beta]\ [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1, \ldots, j_\ell \in [1,n]$ *with* $j_1 = 1$ *and* $0 \le \ell' \le \ell$ *where* (p,β) *is a reachable configuration of* $\mathcal{R}$.

Proof. By induction on the length of a run from $(q_0, \perp_k)$ witnessing reachability. ◀

Thanks to the **reset** transitions we have the following:

▶ **Lemma 10.** *Assume* $(q_0, \perp_k) \longrightarrow^* c$ *for some configuration* c *of* $\mathcal{P}$ *such that* $c \sim c'$ *for some configuration* c' *of some* k'*-PDA with* $k - k' \ge 2$. *Then* $(q_0, \perp_k) \longrightarrow^* c_0$ *where* $c_0 = (q_0', [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})])$, *for some* $j_1, \ldots, j_\ell \in [1,n]$ *and* $j_1 = 1$. *Moreover, there is some configuration* c_0' *of* $\mathcal{P}'$ *with* $c' \longrightarrow^* c_0'$ *and* $c_0 \sim c_0'$.

Proof. Since $c = (p,\alpha)$ must satisfy one of the conditions Lemma 9, by considering each case in turn it can easily be verified that $c \xrightarrow{u} c_0$ for c_0 of the requisite form with $u \in \mathsf{A}^*$. Since $c \sim c'$ we must have $c' \xrightarrow{u} c_0'$ with $c_0 \sim c_0'$ for some configuration c_0' of $\mathcal{P}'$. ◀

Let us first consider the case when $\mathcal{X}$ has no solution. Due to Lemma 10 it is sufficient to consider $\mathcal{P}$-configurations c of the form (q_0', α). Since the image of the sequence of indices in α under h_1 and h_2 must differ, it can be shown that any automaton with configuration bisimilar to c could be converted to one recognising $\mathcal{L}(\mathcal{R})$. Completeness then gives:

▶ **Lemma 11.** *If* $\mathcal{X}$ *has no solution, then there is no reachable configuration* c *of* $\mathcal{P}$ *and* k'*-PDA* $\mathcal{P}'$ *with configuration* c' *such that* $c \sim c'$.

Now we prove the converse.

▶ **Lemma 12.** *If* $\mathcal{X}$ *has a solution, then* $(q_0, \perp_k) \longrightarrow^* c$ *for some configuration* c *of* $\mathcal{P}$ *and a deterministic real-time* k'*-PDA* $\mathcal{P}'$ *with configuration* c' *such that* $c \sim c'$.

Proof. Let $j_1 \cdots j_\ell$ be a solution to $\mathcal{X}$. Let $c \stackrel{\text{def}}{=} (q_0', [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})])$ with $h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_\ell) = a_1 \cdots a_m$ with $a_i \in \Sigma$ for each $i \in [1,m]$. We then take $\mathcal{F} = (S, \mathsf{A}, \{\xrightarrow{a}_{\mathcal{F}} \mid a \in \mathsf{A}\})$ to be the following deterministic finite transition system, which in particular is the transition system of a k'-PDA and set $c' \stackrel{\text{def}}{=} s$. We set $S \stackrel{\text{def}}{=} \{s,t,u\} \cup \{x_i \mid 0 \leq i \leq \ell\} \cup \{y_i \mid 0 \leq i \leq m\}$ and define and $\xrightarrow{a}_{\mathcal{F}}$ for each $a \in \mathsf{A}$ as follows:

$$
\begin{array}{llll}
s \xrightarrow{\#}_{\mathcal{F}} x_\ell & \text{and} & x_\ell \xrightarrow{\#}_{\mathcal{F}} x_{i-1} & \text{for each } i \in [1,\ell] \\
x_0 \xrightarrow{\#}_{\mathcal{F}} t & \text{and} & t \xrightarrow{a}_{\mathcal{F}} t & \text{for each } a \in \mathsf{A}_{\mathcal{R}} \\
t \xrightarrow{\#}_{\mathcal{F}} u & \text{and} & u \xrightarrow{\#}_{\mathcal{F}} \bar{y_m} & \\
\bar{y_0} \xrightarrow{\mathsf{reset}} s & \text{and} & \bar{y_i} \xrightarrow{a_i}_{\mathcal{F}} \bar{y_{i-1}} & \text{for each } 1 \leq i \leq m
\end{array}
$$

Recalling that $\mathcal{R}$ is total-deterministic and real-time and so from every configuration may make an a-transition for any $a \in \Sigma_{\mathcal{R}}$, the reader can verify that the symmetric closure of the following relation is a bisimulation:

$$
\begin{aligned}
& \{(c,s)\} \cup \{((q_{\mathcal{R}},\beta),t) \mid c \longrightarrow^* (q_{\mathcal{R}},\beta) \text{ and } q_{\mathcal{R}} \in Q_{\mathcal{R}}\} \\
\cup\ & \{((\bar{y_j},\alpha),u) \mid c \longrightarrow^* (\bar{y_j},\alpha)\} \\
\cup\ & \{((q_0'',\alpha_{\ell'}),x_{\ell'}) \mid \alpha_{\ell'} = [\mathsf{swap}_{j_{\ell'} \cdots j_1 \perp}(\perp_{k-1})]\,[\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]) \quad 0 \leq \ell' \leq \ell\} \\
\cup\ & \{((y_j,\alpha_j^i),\bar{y_i}) \mid (y_j,[\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]\,[\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]) \xrightarrow{a_m \cdots a_{i+1}} (y_j,\alpha_j^i), \\
& \qquad 1 \leq j \leq 2,\ 0 \leq i \leq m\}
\end{aligned}
$$

◀

5 Further Directions

We believe some limited generalisation of undecidability for $\mathbf{LO}_{k,k'}$ to the case when $\mathcal{P}'$ may also range over *collapsible pushdown automata* [8] is possible. We expect it to be possible to adapt the construction to use the recent hierarchy theorem by Kartzow and Parys for deterministic CPDA [13]. Indeed in the light of [18] one might expect to get a version where $\mathcal{P}$ is a 2-CPDA and $\mathcal{P}'$ can range over deterministic PDA of *any order*. One obstacle is when $(k - k') = 1$ and $\mathcal{P}'$ needs to be able to keep track of the height of the k-stack of $\mathcal{P}$, meaning that a simple extension to the CPDA case would require one to prohibit 'k-links'.

References

1 Rajeev Alur and P Madhusudan. Visibly Pushdown Languages. In *STOC*, pages 202–211. ACM, 2004.

2 J. L. Balcázar, J. Gabarró, and M. Santha. Deciding Bisimilarity is P-Complete. *Formal Asp. Comput.*, 4(6A):638–648, 1992.

3 S. Böhm, S. Göller, and P. Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In *Proc. of CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2010.

4 T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proceedings of ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer, 2003.

5 A. Carayol and S. Wöhrle. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

6 W. Damm. An algebraic extension of the Chomsky Hierarchy. In *MFCS*, 1979.

7 W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71(1-2):1–32, October 1986.

8 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible Pushdown Automata and Recursion Schemes. In *LICS*. IEEE Computer Society, 2008.

9 D. Janin and I. Walukiewicz. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In *Proc. of CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

10 P. Jančar. Strong Bisimilarity on Basic Parallel Processes is PSPACE-complete. In *Proc. of LICS*, pages 218–227. IEEE Computer Society, 2003.

11 P. Jančar. Decidability of dpda language equivalence via first-order grammars. In LICS, 2012.

12 P. Jančar and Jirí Srba. Undecidability of bisimilarity by defender's forcing. *J. ACM*, 55(1), 2008.

13 A. Kartzow and P. Parys. Strictness of the Collapsible Pushdown Hierarchy. In *MFCS*, 2012.

14 T.r Knapik, D. Niwinski, and P. Urzyczyn. Higher-Order Pushdown Trees Are Easy. In *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

15 A. Kucera. Efficient Verification Algorithms for One-Counter Processes. In *ICALP*, pages 317–328, 2000.

16 A N Maslov. Multilevel Stack Automata. *Problems of Information Transmission*, (12):38–43, 1976.

17 F. Moller and A. M. Rabinovich. Counting on CTL^*: on the expressive power of monadic path logic. *Inf. Comput.*, 184(1):147–159, 2003.

18 P. Parys. On the Significance of the Collapse Operation. In *LICS*, volume 1, 2012.

19 K. Ruohonen. Reversible machines and post's correspondence problem for biprefix morphisms. *Elektronische Informationsver- arbeitung und Kybernetik*, 21(12):579–595, 1985.

20 G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.

21 G. Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.*, 34(5):1025–1106, 2005.

22 M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

23 J. Srba. *Roadmap of Infinite results*, volume Vol 2: Formal Models and Semantics. World Scientific Publishing Co., 2004. http://www.brics.dk/~srba/roadmap.

24 R.E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967.

25 C. Stirling. Deciding DPDA Equivalence Is Primitive Recursive. In *Proc. of ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer, 2002.

26 C. Stirling. Second-order simple grammars. In *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 2006.

27 Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *J. ACM*, 22(1):1–10, January 1975.

28 J. van Benthem. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, 1976.

29 R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.