# The Limits of Decidability for First Order Logic on CPDA Graphs

**Christopher H. Broadbent**

**Department of Computer Science, Oxford University**
**Wolfson Building, Parks Road, Oxford, OX1 3QD, UK**
`christopher.broadbent@keble.oxon.org`

## Abstract

Higher-order pushdown automata (HOPDA) are abstract machines equipped with a nested 'stack of stacks of stacks'. Collapsible pushdown automata (CPDA) extend these devices by adding 'links' to the stack and are equi-expressive for tree generation with simply typed $\lambda Y$ terms. Whilst the configuration graphs of HOPDA are well understood, relatively little is known about the CPDA graphs. The order-2 CPDA graphs already have undecidable MSO theories but it was only recently shown by Kartzow (in his 2010 STACS paper) that first-order logic is decidable at the second level. In this paper we show the surprising result that first-order logic ceases to be decidable at order-3 and above. We delimit, in terms of quantifer alternation and the orders of CPDA links used, the fragments of the decision problem to which our undecidability result applies. Additionally we exhibit a natural sub-hierarchy to which limited decidability applies.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.4.1 Mathematical Logic

**Keywords and phrases** Collapsible Pushdown Automata, First Order Logic, Logical Reflection

## 1 Introduction

Higher-order pushdown automata generalise traditional pushdown automata by allowing the stack to contain other stacks rather than just atomic elements. These devices are closely related to *recursion schemes*, which are essentially simply typed $\lambda Y$ terms that generate a single infinite tree. Enjoying decidable $\mu$-calculus theories, the class of trees generated by recursion schemes shows a lot of promise as a model for verifying higher-order functional programs [12, 13]. Unfortunately $n$-PDA are believed to expressively coincide with order-$n$ recursion schemes only when the latter satisfy a property called *safety* [10]. It is conjectured that unsafe recursion schemes are strictly more expressive and this is known for level 2 [14]. Hence a more powerful automaton is needed, which motivates order-$n$ *collapsible pushdown automata* ($n$-CPDA) [7]. Inspired by *panic automata* [11], which can be viewed as the special case at order-2, atomic elements in collapsible stacks eminate 'links' that target a component of the stack further below. Their expressive power coincides precisely with unrestricted order-$n$ recursion schemes.

We concern ourselves here with configuration *graphs* of these automata, with states of memory as nodes and transitions as edges. It is particularly fruitful to consider the '$\epsilon$-closures' of such graphs, which allow to construct a graph whose edges consist of an unbounded number of transitions rather than just single steps. The $\epsilon$-closures of HOPDA graphs form precisely the *Caucal Hierarchy* [6, 3, 5], which is defined independently in terms of graph transformations. This deep result has as a consequence that every $n$-PDA graph has decidable MSO theory.

So how does the addition of links affect this? Unfortunately there is even a 2-CPDA graph that has *un*decidable MSO theory [7]. Nevertheless the local nature of first-order logic meant it was widely assumed the first-order theories would still enjoy decidability. However, the problem remained open for a few years until Kartzow saw that the $\epsilon$-closures of 2-CPDA graphs are *tree automatic* [9] and so do indeed have decidable first-order theory.

| | 2-CPDA /w $\epsilon$-clos. | $3_2$-CPDA | $3_2$-CPDA /w $\epsilon$-clos. | $n_n$-CPDA ($n \geq 3$) | $n_n$-CPDA /w $\epsilon$-clos. ($n \geq 3$) | $n_{n,(n-1)}$-CPDA $n \geq 4$ | $n_{n,(n-1)}$-CPDA /w $\epsilon$-clos. (all $n$) | $n_m$-CPDA ($n \geq 4$, $m \leq n-2$) | $n$-PDA /w $\epsilon$-clos. (all $n$) |
|---|---|---|---|---|---|---|---|---|---|
| $\Sigma_1$ | Dec [9] | Dec [1] | **Dec** | **Dec** | **Dec** | ? | ? | **Und** | Dec [5] |
| $\Pi_2$ | Dec [9] | Dec [1] | **Und** | **Und** | **Und** | ? | ? | **Und** | Dec [5] |
| ***FO*** | Dec [9] | Dec [1] | **Und** | **Und** | **Und** | ? | ? | **Und** | Dec [5] |
| MSO | Und [7] | Und [7] | Und [7] | Und [7] | Und [7] | Und [7] | Und [7] | Und [7] | Dec [5] |
| $\mu$-calculus | Dec [11] | Dec [7] | Dec [7] | Dec [7] | Dec [7] | Dec [7] | Dec[7] | Dec [7] | Dec [10] |

**Table 1** Summary of (un)decidability results known to date. Those in bold are from this paper. The notation '$n_{m,m'}$-CPDA' means an $n$-CPDA that only uses links of orders $m$ and $m'$.

Our first contribution is to show that Kartzow's result cannot be extended to higher-orders in full generality. At order-3 we get undecidability when we consider $\Pi_2$-sentences, namely those with quantifier alternation of the form $\forall\exists$. If we allow order-3 links we do not even need $\epsilon$-closure for this. This result is surprising in itself, but we also gain some insight into what links 'mean' in terms of 3-CPDA *graphs*. On the one hand links can act as 'place holders' that allow first-order logic to compare internal components of a single order-3 stack rather than just two order-3 stacks in their entirety. This is the core of the undecidability result, which goes via a reduction from Post's Correspondence Problem [15]. Additionally, order-3 links provide edges in the graph that are 'non-local' in nature, allowing $\epsilon$-closure to be eliminated as a requirement for undecidability. At order-4 we get that even the $\Sigma_1$-theory is undecidable—*viz.* the theory consisting of sentence *without any* quantifier alternation, a particularly expressively weak fragment of the logic.

Our second contribution introduces a technique to tackle the $\Sigma_1$-theories of CPDA graphs. Making use of *logical reflection* [2], which enables CPDA to 'know' which $\mu$-calculus sentences they satisfy at any given point, we define a notion of *monotonic CPDA* that can construct all of its reachable configurations in a manner that does not every destroy an order-$(n-1)$ stack. This has some parallels to Carayol's work on canonical sequences of operations witnessing the constructibility of HOPDA stacks [4]. Provided that an $n$-CPDA only has 'order-$n$ links', monotonicity allows us to eliminate them, thereby producing an $n$-PDA with decidable MSO theory, and leading to the decidability of the $\Sigma_1$ theory of the original $n$-CPDA. This can be viewed as a graph analogue of Aehlig *et al.'s* work [8] in which it is shown that links can be eliminated from 2-CPDA graphs to produce an equivalent word-generating 2-PDA at the expense of introducing non-determinism. This decidability result is the first that applies a fragment of first-order logic to $n$-CPDA of all orders and establishes that restricting links can recover some decidability.

# 2 Preliminaries

## 2.1 Higher-Order Stacks

Let us fix a stack-alphabet $\Gamma$. For higher-order automata this alphabet must be finite, but it is convenient for definitions to allow it to be infinite. An *order*-1 stack over $\Gamma$ is just a string of the form $[\ \gamma\ ]$ where $\gamma \in \Gamma^*$. Let us refer to the set of order-1 stacks over $\Gamma$ as $stack_1(\Gamma)$. For $n \in \mathbb{N}$ the set of *order*-$(n+1)$-stacks is: $stack_{n+1}(\Gamma) := stack_1(stack_n(\Gamma))$. In a (higher-order) stack $[\ s_1 s_2 \cdots s_m\ ]$ we refer to $s_i$ as the *ith component of the stack*. We allow the following operations on an order-1 stack $s$ for every $a \in \Gamma$: $push_1^a([\ a_1 \cdots a_m\ ]) := [\ a_1 \cdots a_m a\ ]$, $pop_1([\ a_1 \cdots a_m a_{m+1}\ ]) := [\ a_1 \cdots a_m\ ]$, $nop(s) := s$. We allow the following operations on an order-$(n+1)$ stack $s$, where $\theta$ is any operation that may be performed on an order-$n$ stack: $push_{n+1}([\ s_1 \cdots s_m\ ]) := [\ s_1 \cdots s_m s_m\ ]$, $pop_{n+1}([\ s_1 \cdots s_m s_{m+1}\ ]) := [\ s_1 \cdots s_m\ ]$,

$\theta([\ s_1 \cdots s_m\ ]) := [\ s_1 \cdots \theta(s_m)\ ]$. Operations that we may perform on an order-$n$ stack are collectively referred to as *order-$n$ operations*. Where $s$ is an $(n+1)$-stack. We also use the notation $top_{n+1}(s)$ to denote the top-most order-$n$ stack and $top_1(s)$ as the top atomic element.

▶ **Definition 1.** Let $s := [s_1 s_2 \cdots s_m]$ be an $n$-stack. Then we define the *$n$-height* $|s|_n$ of $s$ by $|s|_n := m$. If $1 \leq k < n$, then the *$k$-height* $|s|_k$ of $s$ is recursively defined by: $|s|_k := \sum_{i=1}^{m} |s_i|_k$

▶ **Definition 2.** Let $s := [s_1 s_2 \cdots s_m]$ and $t := [t_1 t_2 \cdots t_{m'}]$ be two $n$-stacks. We say that $s$ is *an $n$-prefix of $t$* written $s \sqsubseteq_n t$ just in case $m \leq m'$ and $s_i = t_i$ for $1 \leq i \leq m$. We recursively say that $s \sqsubseteq_k t$ for $k < n$ just in case $m \leq m'$ and $s_i = t_i$ for $1 \leq i < m$ and $s_m \sqsubseteq_k t_m$. We write $s \sqsubset_k t$ to mean that $s \sqsubseteq_k t$ and $s \neq t$.

▶ **Example 3.** Consider 2-stacks $s := [[ababa][bababa]]$ and $t := [[ababa][bab]]$. Then we have $s \sqsubseteq_1 t$ but we do not have $s \sqsubseteq_2 t$. We also have $|s|_2 = |t|_2 = 2$ but $|s|_1 = 11$ and $|t|_1 = 8$.

▶ **Definition 4.** Let $s = [s_1 \cdots s_m]$ be a higher-order stack. Then $s_{\leq s_i} := s = [s_1 \cdots s_i]$ for $1 \leq i \leq m$. If $t$ is an occurrence of a stack in $s_i$, then $s_{\leq t} := [s_1 \cdots s_{i \leq t}]$. We also have a strict version where $s_{<s_i} := s = [s_1 \cdots s_{i-1}]$ and $s_{<t} := s = [s_1 \cdots t_{<s_i}]$.

## 2.2 Collapsible Pushdown Stacks

We offer fine control over the orders of links that a collapsible stack may contain—an order-$(n+1)$ link is one that targets an order-$n$ stack within an order-$(n+1)$ stack. We include the orders of links as a subscript, so an order-$n_S$ stack is an order-$n$ stack equipped with order-$i$ links for each $i \in S$. The *$S$-collapsible pushdown alphabet* (for $S \subseteq \mathbb{N}$) $\Gamma^{[S]}$ induced by an alphabet $\Gamma$ is the set $\Gamma \times S \times \mathbb{N}$. The set of order-$n_S$ *open* collapsible stacks $stack^{\mathcal{C}}_{n_S}(\Gamma)$ is defined by: $stack^{\mathcal{C}}_{n_S}(\Gamma) := stack_n(\Gamma^{[S]})$. The order of a link of an atomic element $(a, l, p) \in \Gamma^{[S]}$ is given by the number in its second component. If $l < n$ the target of a link is the $p$th component of the $l$-stack in which the element resides. If $l \geq n$ we say that the link is 'dangling'.

When we write $top_1(s)$ where $s$ is a collapsible stack with top atomic element $(a, l, p)$ by abuse of notation we usually mean $top_1(s) := a$. If $(a, l, p)$ is intended, it is usually clear from the context. However, we have additional notation to explicitly refer to $l$.

▶ **Definition 5.** Let $s$ be a collapsible stack with top atomic element $(a, l, p)$, which by abuse of notation we denote $a$. We then define $\boldsymbol{l_o}(a) := l$ (the order of the link) and $\boldsymbol{l_a}(a) := l$ (the absolute target of the link relative to the bottom of the $p$-stack in which it resides). It is also useful to describe the target of the link in terms of an offset from the top: $\boldsymbol{l_r}(a) := |t|_l - 1$ where $t$ is the $l$ stack within $s$ in which the occurrence $a$ resides.

Note in particular that a $push_k$ operation will preserve the absolute targets of links when copying the $k-1$ stack. We replace the $push_1$ operation to allow the attachment of links: $push_1^{a,k}(s) := push_1^{(a,k,|top_{k+1}(s)|_k - 1)}(s)$ where $top_{n+1}(s) := s$ where $s$ is order-$n$. The *collapse* operation discards everything above the target of a pointer. This can neatly be described in terms of link offset where $\theta^m$ means the $m$-fold iteration of the operation $\theta$: $collapse(s) := pop_{\boldsymbol{l_o}(top_1(s))}{}^{\boldsymbol{l_r}(top_1(s))}$. Write $\Theta_{n_S}$ to denote the set of order-$n_S$ collapsible stack operations.

▶ **Example 6.** We exhibit operations on an order-3 collapsible stack, representing links graphically.

$$[[[abca]\ [ab]]] \xrightarrow{push_1^{a,2};push_2;push_3;push_1^{c,3}push_3}$$

$$[[[abca\ ]\ \ [ab\ \ a\ \ ]\ [ab\ \ a\ \ ]\ \ ]\ \ [[abca\ \ ]\ \ [ab\ \ a\ \ ]\ [ab\ \ a\ \ c\ \ ]]\ [[abca\ \ ]\ \ [ab\ \ a\ \ ]\ [ab\ \ a\ \ c\ \ ]]]$$

## 2.3 The Automata and their Graphs

▶ **Definition 7.** Let $n \in \mathbb{N}$ and let $S \subseteq [1..n]$. An $n_S$-CPDA (order-$n_S$ collapsible pushdown automaton) $\mathcal{A}$ is a tuple:

$$\langle\ \Sigma, \Pi, Q, q_0, \Gamma, R_{a_1}, R_{a_2}, \ldots, R_{a_r}, P_{b_1}, P_{b_2}, \ldots, P_{b_{r'}}\ \rangle$$

where $\Sigma$ is a finite set of transition labels $\{a_1, a_2, \ldots, a_r\}$; $\Pi$ is a finite set of configuration labels $\{b_1, b_2, \ldots, b_{r'}\}$; $Q$ is a finite set of control-states; $q_0 \in Q$ is an initial control-state; $\Gamma$ is a finite stack alphabet; each $R_{a_i}$ is the $a_i$-labelled transition relation with $R_{a_i} \subseteq Q \times \Gamma \times \Theta_{n_S} \times Q$; each $P_{b_i}$ is the $b_i$-labelled unary predicate specified by $P_{b_i} \subseteq Q \times \Gamma$.

We define $n$-CPDA and $n$-PDA in a manner consistent with the standard definitions in the literature:

▶ **Definition 8.** An $n$-CPDA is an $n_{[n..2]}$-CPDA and an $n$-PDA is an $n_\emptyset$-CPDA.

A *configuration* of an $n_S$-CPDA is a pair $(q, s)$ where $q$ is a control-state and $s$ is a stack.

▶ **Definition 9.** Let $(q, s)$ and $(q', s')$ be configurations of an $n_S$-CPDA $\mathcal{A}$. We say that $(q', s')$ *can be reached from* $(q, s)$ in $\mathcal{A}$ *with path labeled in* $\mathcal{L}$ for some $\mathcal{L} \subseteq \Sigma^*$ just in case:

$$(q, s)\boldsymbol{a_{i_1}}(q_1, s_1)\boldsymbol{a_{i_2}}(q_2, s_2)\boldsymbol{a_{i_3}} \cdots (q_{m-1}, s_{m-1})\boldsymbol{a_{i_m}}(q', s')$$

for some configurations $(q_1, s_1), \ldots, (q_{m-1}, s_{m-1})$ where $a_{i_1}a_{i_2}a_{i_3}\cdots a_{i_m} \in \mathcal{L}$. We write $(q, s)\boldsymbol{r}_{\mathcal{L}}(q', s')$ to mean this. We write $(q, s)\boldsymbol{r}(q', s')$ to mean $(q, s)\boldsymbol{r}_{\Sigma^*}(q', s')$.

The set of *reachable configurations* of $\mathcal{A}$ is given by:

$$\boldsymbol{R}(\mathcal{A}) := \{\ (q, s)\ :\ (q_0, \perp_n)\boldsymbol{r}(q, s)\ \}$$

▶ **Definition 10.** Let $\mathcal{A}$ be an $n_S$-CPDA with transition-labels $\Sigma$ and configuration-labels $\Pi$. The configuration graph of (graph generated by) $\mathcal{A}$ has domain $\boldsymbol{R}(\mathcal{A})$, unary predicates $\Pi$ and directed edges $\Sigma$ between configurations. We write $\mathcal{G}(\mathcal{A})$ to denote this graph.

The $\epsilon$-closure of such a graph $\mathcal{G}^\epsilon(\mathcal{A})$ is induced from $\mathcal{G}(\mathcal{A})$ by taking $a$-labelled edges between nodes related by $\boldsymbol{r}_{\epsilon^* a}$.

## 2.4 Logics

We consider first-order logic $\boldsymbol{FO}$ on graphs as is standardly defined. A $\Sigma_0 = \Pi_0 = \Delta_0$ formula $\phi(x_1, \ldots, x_k)$ is one without any quantifiers. A $\Sigma_{n+1}$ formula is one of the form $\exists\vec{y}.\phi(\vec{y}, x_1, \ldots, x_k)$ where $\phi(\vec{y}, x_1, \ldots, x_k)$ is $\Pi_n$ and a $\Pi_{n+1}$ formula is one of the form $\exists\vec{y}.\phi(\vec{y}, x_1, \ldots, x_k)$ where $\phi$ is $\Sigma_n$. A $\Delta_n$ formula is one that is equivalent to both a $\Sigma_n$ and $\Pi_n$ formula on every CPDA graph. MSO is $\boldsymbol{FO}$ extended with second-order quantification over set variables. Transitive closure logic $\boldsymbol{FO(TC)}$ is $\boldsymbol{FO}$ together with a binary predicate $\overline{\phi(x, y)}$ for every formula of $\boldsymbol{FO}$ $\phi(x, y)$ with two variables, which defines the relation that is the transitive closure of that defined by $\phi(x, y)$. When the formulae transitively closed are restricted to $\Delta_1$ we call the logic $\boldsymbol{FO(TC[\Delta_1])}$. A *sentence* is a formula with no free variables.

# 3 Undecidability

## 3.1 Post's Correspondence Problem

All of the undecidability results go via a reduction from Post's Correspondence Problem [15], which is known to be undecidable. Consider a finite-alphabet $\Sigma$ with $|\Sigma| \geq 2$. An instance of the Post Correspondence Problem (PCP) consists of two finite sets of strings over $\Sigma$: $u_1, \ldots, u_m$ and $v_1, \ldots, v_m$.

The question to be decided is whether there is a finite sequence $i_1 \dots i_k$ consisting of integers $1 \leq i_j \leq m$ such that $u_{i_1}.u_{i_2}.\cdots.u_{i_k} = v_{i_1}.v_{i_2}.\cdots.v_{i_k}$.

▶ **Example 11.** Consider the following two sets of strings over the alphabet $\{a, b, c\}$:

$$u_1 := ab \; u_2 := cababcabb \; u_3 := ca \qquad v_1 := ababc \; v_2 := ab \; v_3 := bca$$

Then the Post Correspondence Problem has answer 'yes' as witnessed by the solution 1123:

$$u_1.u_1.u_2.u_3 = ababcababcabbca = v_1.v_1.v_2.v_3$$

Given an instance of the PCP $P$ we define a pushdown automaton $\mathcal{A}_1^P$ that pushes elements of $\Sigma$ onto the stack together with indices indicating a partition into strings from the $u_i$ and $v_i$.

▶ **Definition 12.** The automaton $\mathcal{A}_1^P$ has stack alphabet: $\Sigma \uplus [1_u, 2_u \dots m_u] \uplus [1_v, 2_v \dots m_v]$ and behaves by non-deterministically choosing one of the following options:

- Push any member of $\Sigma$ onto the stack.
- If the $\Sigma$ symbols in the stack since the last symbol of the form $i_u$ (or the bottom of the stack if there is no such symbol) form the word $u_j$, then it may push $j_u$ onto the stack.
- If the $\Sigma$ symbols in the stack since the last symbol of the form $i_v$ (or the bottom of the stack if there is no such symbol) form the word $v_j$, then it may push $j_v$ onto the stack.

$P$ has a solution just in case $\mathcal{A}_1^P$ can generate a stack with 'matching' $i_u$ and $i_v$ subsequences.

▶ **Lemma 13.** *Let $P$ be an instance of Post's Correspondence Problem. $P$ has a solution just in case the automaton $\mathcal{A}_1^P$ can generate a stack $s$ such that:*

- *$s_u = s_v$ where $s_u$ is the subsequence of $s$ consisting of elements of the form $i_u$ and $s_v$ of elements of the form $i_v$ and equality is interpreted with respect to the indices $i$ only.*
- *The top two elements of $s$ form the set $\{i_u, i_v\}$ for some $1 \leq i \leq m$.*

▶ **Example 14.** To continue the running example from Example 11, which we call $P$, the solution as represented by a stack of $\mathcal{A}_1^P$ is: [ $ab\mathbf{1_u}ab\mathbf{1_u}c\mathbf{1_v}ababc\mathbf{1_v}ab\mathbf{2_v}b\mathbf{2_u}ca\mathbf{3_u}\mathbf{3_v}$ ]

## 3.2 Post's Correspondence Problem and $2$-CPDA

Hague *et al.* [7] showed that the model-checking problem for MSO on 2-CPDA graphs is undecidable; indeed the 2-CPDA graph that they exhibit witnesses the undecidability of transitive closure logic $\boldsymbol{FO(TC)}$. In order to introduce our basic technique, we first reprove the undecidability of $\boldsymbol{FO(TC)}$ on 2-CPDA graphs by a reduction from PCP—in fact $\boldsymbol{FO(TC[\Delta_1])}$.

The 2-CPDA $\mathcal{A}_2^P$ is very like $\mathcal{A}_1^P$ except that it ensures each index (the elements of the form $i_u$ or $i_v$) eminate a pointer to a distinct 1-stack in the 2-stack. This will enable first-order logic to 'ascertain corresponding positions' in two instances of a 1-stack by comparing the results of collapsing.

▶ **Definition 15.** Let $P$ be an instance of Post's Correspndence Problem (using the notation above). The automaton $\mathcal{A}_2^P$ has stack alphabet: $\Sigma \uplus [1_u, 2_u \dots m_u] \uplus [1_v, 2_v \dots m_v]$. It behaves by non-deterministically choosing one of the following options:

- Push any member of $\Sigma$ onto the stack.
- If the $\Sigma$ symbols in the stack since the last symbol of the form $i_u$ in the top 1-stack (or the bottom of the 1-stack if there is no such symbol) form the word $u_j$, then it may perform $push_2; push_1^{j_u,2}$.
- If the $\Sigma$ symbols in the stack since the last symbol of the form $i_v$ in the top 1-stack (or the bottom of the 1-stack if there is no such symbol) form the word $v_j$, then it may perform $push_2; push_1^{j_v,2}$.

As with $\mathcal{A}_1^P$, the finite control-states can enforce the preconditions.

A solution to $P$ can be formulated in terms of $\mathcal{A}_2^P$ in a very similar manner to before:

▶ **Lemma 16.** *Let $P$ be an instance of Post's Correspondence Problem. $P$ has a solution just in case the automaton $\mathcal{A}_2^P$ can generate a stack $s$ such that:*

- *$s_u = s_v$ where $s_u$ is the subsequence of $top_2(s)$ consisting of elements of the form $i_u$ and $s_v$ of elements of the form $i_v$ and equality is interpreted with respect to the indices $i$ only.*
- *The top two elements of $s$ form the set $\{i_u, i_v\}$ for some $1 \leq i \leq m$.*

**Proof.** A consequence of Lemma 13 given that the permissible changes to the $top_2$ 1-stack of the 2-stack of $\mathcal{A}_2^P$ are precisely those that could be made to the 1-stack of $\mathcal{A}_1^P$. ◀

▶ **Example 17.** To continue Example 11, the solution as represented by a stack of $\mathcal{A}_2^P$ is:

$$[[ab\ ]\ [ab\ \mathbf{1_u}\ ab\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ab\ \mathbf{2_v}\ b\ \mathbf{2_u}\ ca\ \mathbf{3_u}\ \mathbf{3_v}]]$$

Now consider a variant of $\mathcal{A}_2^P$ which we will call $\mathcal{A}_{2+}^P$. This behaves as follows:

- It initially behaves as $\mathcal{A}_2^P$. It may terminate this phase if its top two elements are in $\{i_u, i_v\}$ for some $i$. When it terminates this phase, it enters a distinguished control-state $\boldsymbol{guess}$.
- It then performs a sequence of operations of the form:
  $push_2; pop_1^{m_1}; push_2; pop_1^{m_2}; push_2; pop_1^{m_3}; push_2; pop_1^{m_4}$ with $m_1, m_2, m_2, m_4 \in \mathbb{N}$. It should ensure that the four 1-stacks so created form a set $\{u_1, u_2, v_1, v_2\}$ (in no particular order) with:
  - $top_1(u_1) = i_u$ and $top_1(v_1) = i_v$ for some $1 \leq i \leq m$.
  - Either $top_1(u_2) = i'_u$ and $top_1(v_2) = i'_v$ for some $1 \leq i' \leq m$ or else $u_2$ and $v_2$ are both empty.
  - Going from $u_1$ to $u_2$ can be done with the popping of precisely one symbol of the form $j_u$ (that is the one on top of $u_1$) and all other symbols popped should be letters.
  - Going from $v_1$ to $v_2$ can be done with the popping of precisely one symbol of the form $k_v$ (that is the one on top of $u_1$).

  Compliance with these requirements can be checked using a finite number of control states. If the automaton finds that it cannot avoid violating a requirement (*e.g.* it reaches the bottom of a stack whilst iterating $pop_1$) then it just enters a distinguished control-state $\boldsymbol{fail}$. Otherwise it enters one of the following distinguished control-states: $\boldsymbol{u_1v_1u_2v_2}^{\boldsymbol{start}}$, $\boldsymbol{v_1u_1v_2u_2}^{\boldsymbol{start}}$ or $\boldsymbol{u_1u_2v_1v_2}$, $\boldsymbol{u_1v_1u_2v_2}$, $\boldsymbol{u_1v_1v_2u_2}$, $\boldsymbol{v_1u_1u_2v_2}$, $\boldsymbol{v_1u_1v_2u_2}$, $\boldsymbol{v_1v_2u_1u_2}$ or $\boldsymbol{u_1v_1}^{\boldsymbol{end}}$, $\boldsymbol{v_1u_1}^{\boldsymbol{end}}$ specifying the order of creation of these four stacks along with a flag indicating $\boldsymbol{end}$ if the $u_2$ and $v_2$ are empty and $\boldsymbol{start}$ if both $u_1$ and $u_2$ have the top-most $i_u$ element of $s$ and both $v_1$ and $v_2$ the top-most $i_v$ element. Call these control-states *verifier states*.
- From one of the above verifier states, the automaton may perform any sequence of of $pop_2$ and $collapse$ operations via edges labelled $\boldsymbol{pop_2}$ and $\boldsymbol{collapse}$ which all end in control-state $\boldsymbol{test}$.

▶ **Example 18.** The automaton $\mathcal{A}_{2+}^P$ could reach the stack in Example 17 in control-state $\boldsymbol{guess}$. From here it could, for example, reach control-state $\boldsymbol{v_1v_2u_1u_2}$ with stack:

$$[[ab\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ ][ab\ \mathbf{1_u}\ ]]$$

▶ **Definition 19.** Let us fix some configuration of $\mathcal{A}_{2+}^P$ with stack $s$ and control-state $\boldsymbol{guess}$. Let us call the configurations reachable from $(\boldsymbol{guess}, s)$ associated with a verifier state the *$s$-verifier configurations*. Given an $s$-verifier configuration $c$ (with set of top four stacks $\{u_1, u_2, v_1, v_2\}$) we define its *successor* $c^+$ to be the configuration (which is unique if it exists) with top four stacks $\{u_1^+, u_2^+, v_1^+, v_2^+\}$ such that $u_1^+ = u_2$ and $v_1^+ = v_2$.

Additionally the unique $s$-verifier bearing the $\boldsymbol{start}$ flag is dubbed $0_s$. The unique $s$-verifier configuration whose control-state bears the $\boldsymbol{end}$ flag is dubbed $\boldsymbol{end}_s$.

We can now produce a version of Lemma 16 in terms of $s$-verifier configurations.

▶ **Lemma 20.** *Let $(\boldsymbol{guess}, s)$ be a reachable configuration of $\mathcal{A}^P_{2+}$ for some instance $P$ of Post's Correspondence Problem. Then $s$ represents a solution to $P$ in the sense of Lemma 16 if and only if there exists a chain of stacks $s_1, s_2, \ldots, s_k$ such that $s_1 = 0_s$, $s_k = \boldsymbol{end}_s$ and $s_{i+1} = s_i^+$ for each $1 \leq i \leq k-1$ and such that for each $i$ there exists a reachable $s_i$-verifier.*

We can define the $\_ = (\_)^+$ relation as a $\Delta_1$ formula. This can be achieved by ensuring that the element on top of $u_2$ in a verifier-configuration stack $s$ is the same as the element on top of $u_1$ in the purported $s^+$, and likewise for $v_1$ and $v_2$. Because every element in a $\mathcal{A}^P_{2+}$ 1-stack has a pointer to a different location, we can detect this by checking that collapsing on $u_1$ results in the same stack as collapsing on $u_2$. Individual $^+$ steps can be extended to a chain via transitive closure.

▶ **Lemma 21.** *There exists a $\Sigma_1$ sentence $\phi$ of $\boldsymbol{FO}(\boldsymbol{TC}[\boldsymbol{\Delta_1}])$ such that for all instances $P$ of Post's Correspondence Problem we have: $\mathcal{G}(\mathcal{A}^P_{2+}) \vDash \phi$ iff $P$ has a solution.*

## 3.3 Undecidability for $3_2$-CPDA

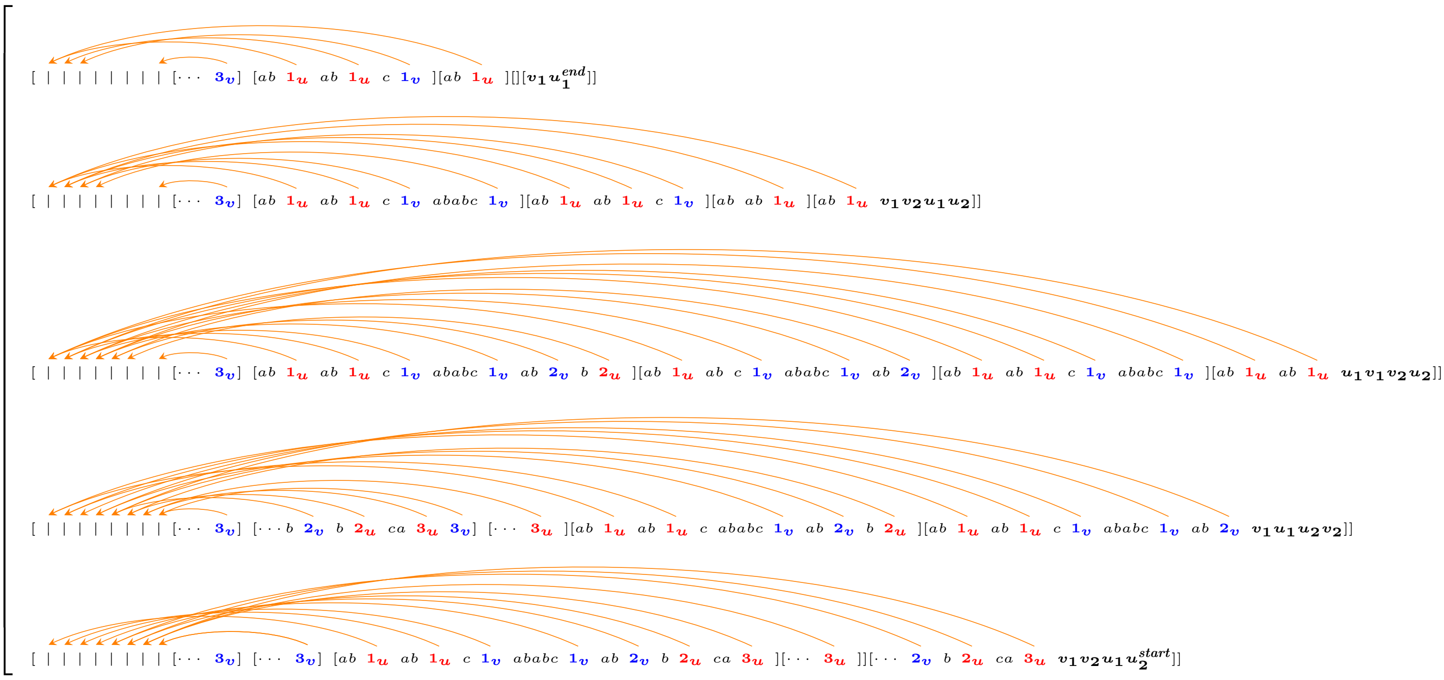
A $3_2$-CPDA can record the chain of 2-stacks mentioned in Lemma 20 directly in its stack—the members of the chain are piled on top of each other. This removes the burden of transitive closure from the logic, although for $3_2$-CPDA we require $\epsilon$-closure.

▶ **Definition 22.** The $3_2$-CPDA $\mathcal{A}^P_{3_2}$ behaves as follows:

- It begins by behaving in the same way as $\mathcal{A}^P_{2+}$, performing only 2-stack operations until it reaches the control-state $\boldsymbol{u_1u_2v_1v_2}^{\boldsymbol{start}}$ or $\boldsymbol{v_1v_2u_1u_2}^{\boldsymbol{start}}$. If it is unable to reach such a state, it goes into a distinguished $\boldsymbol{fail}$ state and aborts.
- The automaton then pushes a record of its $\mathcal{A}^P_{3_2}$ control-state on to the stack and performs: $push_3; pop_2; pop_2; pop_2; pop_2$. This will return the stack to the original $\mathcal{A}^P_{2+}$ $\boldsymbol{guess}$-configuration. It then behaves from here as though it were $\mathcal{A}^P_2$ in the $\boldsymbol{guess}$ control state (with a stack $s$) until it reaches an $s$-verifier-configuration (or finds it is unable to, in which case it aborts).
- The previous step is repeated until an $\boldsymbol{end}_s$ configuration is reached, at which point the CPDA pushes the $\mathcal{A}^P_{2+}$ control-state onto the stack and enters a distinguished $\boldsymbol{guess_{3_2}}$ control-state.

This gives the essence of the automaton, although some extra edges do have to be added to the graph for technical reasons. Let us continue with the running example:

▶ **Example 23.** Recall the stack of $\mathcal{A}^P_2$ in Example 17. $\mathcal{A}^P_{3_2}$ extends this upwards to form a 3-stack with contents disecting each stage in the $i_u$ and $i_v$ subsequences. In control-state $\boldsymbol{guess_{3_2}}$:




A stack in a $\boldsymbol{guess_{3_2}}$ configuration of $\mathcal{A}^P_{3_2}$ fixes not only the 2-stack representing a guess at the solution (as is done by $\mathcal{A}^P_{2+}$) but also a sequence of 2-stacks alleged to witness the correctness of this guess. With the aid of $\epsilon$-closure we can perform an arbitary number of $pop_3$ operations to quantify over the 2-stacks belonging to this alleged chain.

▶ **Lemma 24.** *There exists a $\Sigma_2$ sentence $\phi$ of **FO** such that for all instances $P$ of Post's Correspondence Problem we have: $\mathcal{G}^\epsilon(\mathcal{A}^P_{3_2}) \vDash \phi$ iff $P$ has a solution.*

## 3.4 The Non-Locality of $3_3$-CPDA

Adapting $\mathcal{A}^P_{3_2}$ to become a $3_3$-CPDA is straightforward—we can simply replace the 2-links with 3-links and replace the initial $push_2$ operations with $push_3$ operations to ensure different targets. Moreover, the undecidability result for $3_3$-CPDA is stronger; the non-locality of additional 3-links is exploited to alleviate the need for $\epsilon$-closure. We illustrate this idea of exploiting non-localilty in Figure 1. We can access all elements $z$ in the chain with a first-order formula along the lines of:

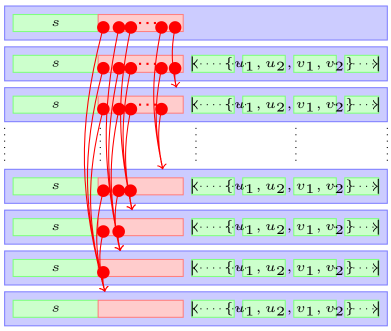



**Figure 1** Exploiting the non-locality of 3-links

$$(\exists x.x \text{ is a candidate stack })(\forall y.y \text{ is a candidate stack after some } pop_1\text{'s} \wedge pop_3(y) = pop_3(x)).z = collapse(y)$$

In this way we can construct a $3_3$-CPDA $\mathcal{A}^P_{3_3}$ such that:

▶ **Lemma 25.** *Let $P$ be an instance of Post's Correspondence Problem. Then there exists a $\Sigma_2$ sentence $\phi$ of **FO** such that: $\mathcal{G}(\mathcal{A}^P_{3_3}) \vDash \phi$ (note* no *$\epsilon$-closure) if and only if $P$ has a solution.*

## 3.5 $\Sigma_1$ Undecidability for $4_2$-CPDA

The $\Sigma_2$-sentence witnessing undecidability on $3_2$ graphs universally quantifies over elements in an alleged chain under the $\_ = (\_)^+$ relation in order to verify that it really is a chain. This is done by verifying that the result of *collapse* on adjacent 2-stacks is the same. However, with a $4_2$-CPDA it is possible to construct two 4-stacks where the 3-stacks to be compared are in corresponding positions. This is illustrated in Figure 2. Unfortunately it is only possible to make this work for half of the definition of successor. The figure demonstrates the $u_1/u_2$ comparison, but we cannot simultaneously do the $v_1/v_2$ comparison. This can be circumvented by revising the definition of chain, successor and automaton to one where only one such comparison is sufficient. This is slightly more fiddly than the definition presented here, but it works in a similar manner. This allows us to construct an automaton $\mathcal{A}^P_{4_2}$ such that:

▶ **Lemma 26.** *There exists a $\Sigma_1$-sentence $\phi$ such that for every instance $P$ of Post's Correspondence Problem we have $\mathcal{G}(\mathcal{A}^P_{4_2}) \vDash \phi$ iff $P$ has a solution.*

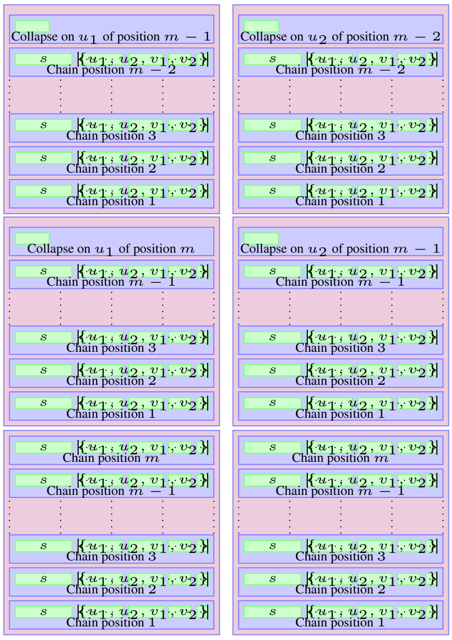



**Figure 2** The idea behind $\mathcal{A}^P_{4_2}$.

As a consequence of all these reductions:

▶ **Theorem 27.** **1.** *For every $n \geq 4$ and $2 \leq m \leq n-2$ the $\Sigma_1$-**FO** model-checking problem for $n_m$-CPDA graphs (even without $\epsilon$-closure) is undecidable.*

**2.** *For every $n \geq 3$ and $m \geq 3$ the $\Pi_2$-**FO** model-checking problem for $n_m$-CPDA graphs (even w/o $\epsilon$-closure) is undecidable.*

**3.** *For every $n \geq 3$ and $m \geq 2$ the $\Pi_2$-**FO** model-checking problem for the $\epsilon$-closures of $n_m$-CPDA graphs is undecidable.*

## 4 $\Sigma_1$ Decidability on $n_n$-CPDA

### 4.1 Monotonic CPDA

We wish to decompose $\epsilon^* a$-labelled runs of an automaton between two configurations into an *$\epsilon$-fall* and an *$\epsilon a$-climb*, which we will describe as a *bounce*. The fall is the first part of the run during which the stack will reach its lowest point, whilst the climb is the part of the run where the lowest point will be built up to the final configuration. This is illustrated in Figure 3.

This leads to the notion of a *monotonic $n$-CPDA*, which witnesses the reachability of all configurations in its graph without destroying $(n-1)$ stacks during its run. The relationship between the graphs of an CPDA $\mathcal{A}$ and its monotonic counterpart is one of *strong isomorphism*. We say that an isomorphism $L$ between two configuration graphs $\mathcal{G}$ and $\mathcal{G}'$ is *strong*, written $L : \mathcal{G} \mathrel{\widehat{\cong}} \mathcal{G}'$, if it maps a configuration $(q, s)$ to one of the form $(q, L(s'))$ where $L$ preserves stack structure and respects $\sqsubseteq_1$ prefixes.

The construction of monotonic CPDA is given in terms of devices called $\mu$CPDA. A $\mu$CPDA operates

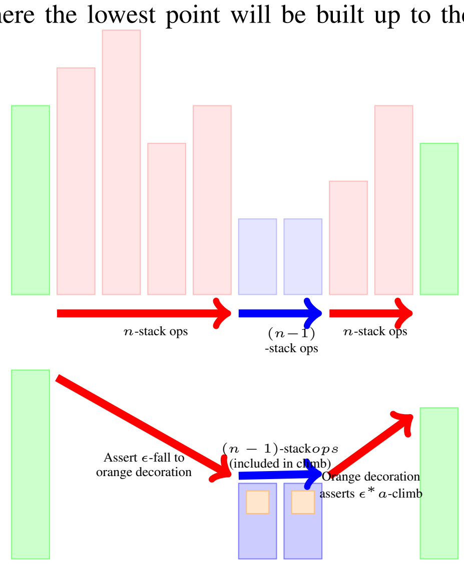



**Figure 3** Asserting the existence of a bounce without performing any stack operations.

on an underlying CPDA, which it manipulates indirectly, determining transitions on the basis of 'tests' for the satisfaction of a $\mu$-calculus sentence at the current configuration of the underlying CPDA. In particular, $\mu$CPDA are able to 'predict the future' and consequently avoid the need to actually perform certain transitions, facilitating monotonicity. Logical reflection [2] implies every $\mu$CPDA is strongly isomorphic to some CPDA.

► **Definition 28.** Let $\mathcal{A}$ be an $n$-CPDA. An $\epsilon^* a$*-climb* of $\mathcal{A}$ from a configuration $(q, s)$ to a configuration $(q', s')$, written $(q, s)\boldsymbol{r}^{\uparrow}_{\epsilon^* a}(q', s')$, is an $\epsilon^* a$-labelled run from the first configuration to the second such that each stack occurring in the run (including $s'$) has stack $t$ such that $pop_n(s) \sqsubset_n t$.

We say that an $n$-CPDA is *monotonic via* $r$ just in case no $r$-transition performs a $collapse$ on an $n$-link or a $pop_n$ operation. That is, when transitioning using $r$-edges, the number of $(n-1)$-stacks in an $n$-stack increases monotonically. The following Lemma constructs an automaton monotonic via an edge $r_\epsilon$ such that $\epsilon^*$-climbs are precisely captured by standard *reachability* using $r_\epsilon$-edges. Write $\mathcal{A} \upharpoonright_{\Pi,\Sigma}$ for the automaton formed from $\mathcal{A}$ by deleting all unary predicates not labelled in $\Pi$ and deleting all transitions not labelled in $\Sigma \cup \{\epsilon\}$.

► **Lemma 29.** *Let $\mathcal{A}$ be an $n$-CPDA with edge alphabet $\Sigma$ and unary predicates $\Pi$. Then there exists an $n$-CPDA $\mathcal{A}^{\uparrow}$ such that $\mathcal{G}^\epsilon(\mathcal{A}) \widehat{\cong} \mathcal{G}^\epsilon(\mathcal{A}^{\uparrow} \upharpoonright_{\Sigma,\Pi})$ but whose additional distinguished edge labels include $r_\epsilon \notin \Sigma$ such that $\mathcal{A}^{\uparrow}$ is monotonic via $r_\epsilon$ and $(q, s)\boldsymbol{r}^{\uparrow}_{\epsilon^*\ a}(q', s')$ just in case $(q, s)\boldsymbol{r}_{r^*_\epsilon\ a}(q', s')$.*

The dual of an $\epsilon$-climb is *an $\epsilon$-fall*; a configuration with a higher stack reaching a configuration with a lower stack such that no configuration in the run descends below the lower stack.

► **Definition 30.** Let $\mathcal{A}$ be an $n$-CPDA. An *$\epsilon$-fall* of $\mathcal{A}$ from a configuration $(q, s)$ to a configuration $(q', s')$ is an $\epsilon^*$-labelled run from $(q, s)$ to $(q', s')$ such that for every stack $t$ occurring in the run (including $s$) we have $pop_n(s') \sqsubset_n t$.

The quasi-analogue of $\mathcal{A}^{\uparrow}$ for falls is $\mathcal{A}^{\downarrow}$. However, we avoid needing to perform any destructive operations (including $collapse$) by instead making $\mathcal{A}^{\downarrow}$ aware of the predicates that $\mathcal{A}$ could satisfy after performing an $\epsilon$-fall.

► **Lemma 31.** *Let $\mathcal{A}$ be an $n$-CPDA with unary predicates $\Pi$. Then there exists an $n$-CPDA $\mathcal{A}^{\downarrow}$ with stack-alphabet $\Gamma^{\downarrow}$ and control-state space $Q^{\downarrow}$ such that $\mathcal{G}(\mathcal{A}) \widehat{\cong} \mathcal{G}(\mathcal{A} \upharpoonright_{\Pi}{}^{\downarrow})$ and that also has a predicate $P^{\downarrow}$ for each $P \in \Pi$ such that $P$ holds of precisely those configurations $c$ from which $\mathcal{A}$ has an $\epsilon$-fall to a configuration $c'$ satisfying $P$.*

► **Definition 32.** Let $\mathcal{A}$ be an $n$-CPDA. An *$a$-bounce* in $\mathcal{A}^{\uparrow}$ from $(q, s)$ to $(q', s')$ in $\mathcal{A}^{\uparrow}$ is a run consisting of an $\epsilon$-fall from $(q, s)$ to some configuration $(q'', s'')$ followed by an $r^*_\epsilon a$-climb from $(q'', s'')$ to $(q', s')$. Let us write $(q, s)\boldsymbol{b}_a(q', s')$ to indicate the existence of such a bounce.

The significance of bounces is summed up in the following lemma. Whilst we strictly only need to consider $\mathcal{A}^{\uparrow}$ we state the Lemma in terms of $\mathcal{A}^{\uparrow\downarrow}$ as when we come to make use of it (with meta-annotations) we will need to reference predicates holding at the bottom of the bounce:

► **Lemma 33.** *Let $(q, s)$ and $(q', s')$ be two configurations of a CPDA $\mathcal{A}$ and let $(q, L(s))$ and $(q', L(s'))$ be the corresponding configurations in $\mathcal{A}^{\uparrow\downarrow}$ via the strong isomorphism. Then $(q, s)\boldsymbol{r}_{\epsilon^* a}(q', s')$ just in case $(q, L(s))\boldsymbol{b}_a L(q', L(s'))$.*

## 4.2 Link Trails: Towards Link Elimination for Graphs

Part of our technique addressing the $\Sigma_1$ model-checking problem for CPDA graphs involves the elimination of outer-most links. The operational part of the simulation will make use of bouncing

(Lemma 33). But it is not enough just to simulate collapse; even if links are never used operationally, they still provide a feature by which stacks may be distinguished. For example the stacks

$[\,[\,abc\ \ ]\ \ [\,ab\ \ c\ \ ]\,]$ and $[\,[\,abc\ \ ]\ \ [\,a\ \ b\ \ c\,]\,]$ are different although removing the link from either gives us the same: $[\ [\ abc\ ]\ [\ abc\ ]\ ]$. We introduce the idea of *link-trails* in order to capture the differences created by links after they have been removed. Stacks and atomic elements are 'coloured' in a manner that is unique to a particular arrangement of $n$-links.

▶ **Definition 34.** We overload the $\boldsymbol{l}_a(s)$ operator to apply to stacks $s$ as well as individual elements. Let $s = [\ s_1\ \ s_2\ \ \cdots ; s_m\ ]$ be a $k$-stack with $2 \leq k$ (located within an $n$-stack for $k \leq n$). Define $\boldsymbol{l}_a(s)$ to be the position of the highest $(n-1)$-stack pointed to by an $n$-link: $\boldsymbol{l}_a(s) = \max(\{\ \boldsymbol{l}_a(s_i)\ :\ 1 \leq i \leq m\ \} \cup \{\ 0\ \})$ and when $s = [\ a_1\ \ a_2\ \ \cdots a_m\ ]$ is an order-1 stack: $\boldsymbol{l}_a(s) = \max(\{\ \boldsymbol{l}_a(a_i)\ :\ \boldsymbol{l}_o(a_i) = n \text{ and } 1 \leq i \leq m\ \} \cup \{\ 0\ \})$

So in particular $\boldsymbol{l}_a(s) = 0$ when $s$ contains no element with an $n$-link.

We now describe how stacks and atomic elements can be ascribed one of four colours in $\{\ c_<, c_=, c_>, \perp\ \}$.

▶ **Definition 35.** We ascribe colours to stacks and atomic elements via a function $\mathbf{col}(\_)$. Let $s = [\ a_1\ a_2\ \cdots\ a_m\ ]$ be a 1-stack located in an $n$-stack. Suppose that $\boldsymbol{l}_o(a_i) = n$ for some $i$:

$$\mathbf{col}(a_i) := \begin{cases} \perp & \text{if } \boldsymbol{l}_o(a_i) \neq n \\ c_> & \text{if for every } j < i \text{ such that } \boldsymbol{l}_o(a_j) = n \text{ we have} \\ & \boldsymbol{l}_a(a_i) > \boldsymbol{l}_a(a_j) \\ c_= & \text{if } \boldsymbol{l}_o(a_i) = n \text{ and the greatest } j < i \text{ such that} \\ & \boldsymbol{l}_o(a_j) = n \text{ satisfies } \boldsymbol{l}_a(a_j) = \boldsymbol{l}_a(a_i) \end{cases}$$

Note that for constructible stacks the above is exhaustive (it is impossible to construct a stack containing a link with target lower than the target of a link below it in the same 1-stack).

Now let $s := [\ s_1\ \ s_2\ \ \cdots\ \ s_m\ ]$ be an order-$k$ stack in an order-$n$ stack for $n \geq k \geq 2$ (in particular we allow $k = n$ in which case $s$ is the whole $n$-stack). We then set $\mathbf{col}(s_i)$ by:

$$\mathbf{col}(s_i) := \begin{cases} c_> & \text{if } \boldsymbol{l}_a(s_i) > \max(\{\ \boldsymbol{l}_a(s_j)\ :\ 1 \leq j < i\ \}) \\ c_= & \text{if } \boldsymbol{l}_a(s_i) = \max(\{\ \boldsymbol{l}_a(s_j)\ :\ 1 \leq j < i\ \}) \\ c_< & \text{if } \boldsymbol{l}_a(s_i) < \max(\{\ \boldsymbol{l}_a(s_j)\ :\ 1 \leq j < i\ \}) \end{cases}$$

For an $n$-stack $s$ let us write $\mathbf{stripln}(s)$ to denote the $n$-stack that results from deleting all of the $n$-links from $s$. The colouring uniquely specifies the $n$-links that it contains:

▶ **Lemma 36.** *Consider two* constructible *$n$-stacks $s$ and $s'$ with* $\mathbf{stripln}(s) = \mathbf{stripln}(s')$. *Assume for every stack or atomic element $a$ contained within $s$ and corresponding element $a'$ in $s'$ we have* $\mathbf{col}(a) = \mathbf{col}(a')$. *Then $s = s'$.*

▶ **Lemma 37.** *Let $\mathcal{A}$ be an $n_n$-CPDA. Then there exists an $n_n$-CPDA* $\mathbf{lum}(\mathcal{A})$ *such that* $\mathcal{G}^\epsilon(\mathbf{lum}(\mathcal{A})) \cong \mathcal{G}^\epsilon(\mathcal{A})$ *and further such that for any reachable configurations $(q,s),(q,s')$ of* $\mathbf{lum}(\mathcal{A})$ *we have $s = s'$ iff* $\mathbf{stripln}(s) = \mathbf{stripln}(s')$.

## 4.3 Meta-Annotations

Having enabled link removal whilst preserving equality between configurations, we now add a mechanism by which we can simulate edges in the graph without needing to actually perform the $collapse$ operations. Indeed it will remove the need to perform any stack operations at all. This is achieved by decorating $(n-1)$-stacks within an $n$-stack with information about the control-states from which a complete $n$-stack could be reached via an $\epsilon^* a$-climb. Because this requires quantification over multiple runs and a CPDA can only perform one run at a time, it is necessary for the CPDA to guess the appropriate annotations and to externally verify them. It turns out that this can be done using MSO over $\mathbf{lum}(\mathcal{A})^{\uparrow\downarrow}$ since Lemma 29 allows climbs to be expressed in terms of $r_\epsilon$-edges; we have $c\boldsymbol{r}^{\uparrow}_{\epsilon^* a}c'$ iff $c\boldsymbol{r}_{r^*_\epsilon a}c'$, and $\boldsymbol{r}_\sigma$ can always be expressed in MSO. Moreover, the reachability of a particular meta-annotation via an $\epsilon$-fall can be asserted by MSO over $\mathbf{lum}(\mathcal{A})^{\uparrow\downarrow}$ due to Lemma 31. This MSO formula makes an assertion of the 'consistency' of $k$-tuples of configurations $(q_1, s_1), \ldots, (q_k, s_k)$ with the constituent $(n-1)$ stacks of each $n$-stack decorated with a 'meta-annotation'. A meta-annotation is a $|\Sigma|.k$-tuple $((Q^a_1)_{a\in\Sigma}, \ldots, (Q^a_k)_{a\in\Sigma})$ where each component is a subset of control-states $Q$. The $k$-tuple is deemed *consistent* if for every $(n-1)$ stack $s'$ in $s_i$, the meta-annotation on top of $s_{i\leq s'}$ satisfies $q \in Q^a_j$ iff there is an $\epsilon^* a$ climb from $(q, s_{i\leq s'})$ to $(q_j, s_j)$.

By asserting the (non-)existence of a suitable $\epsilon$-fall to such a decoration, we can assert the (non-)existence of a bounce witnessing (non-)reachability, as illustrated in Figure 3. This corresponds to reachability in the original automaton due to Lemma 33. Since MSO is decidable on $n$-PDA:

▶ **Theorem 38.** *Let $\mathcal{A}$ be an $n_n$-CPDA. Then the $\Sigma_1$ theory of $\mathcal{G}^\epsilon(\mathcal{A})$ is decidable.*

## References


1 C.H Broadbent. *On Collapsible Pushdown Automata, their Graphs and the Power of Links*. PhD thesis, 2011.

2 C.H. Broadbent, Arnaud Carayol, C.-H.L. Ong, and Olivier Serre. Recursion Schemes and Logical Reflection. In *LICS*, pages 120—-129. IEEE Computer Society, 2010.

3 Thierry 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.

4 A. Carayol. Regular Sets of Higher-Order Pushdown Stacks. In *Proc. MFCS*, volume 3618 of *LNCS*, pages 168–179. Springer, 2005.

5 Arnaud Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, pages 112–123. Springer, 2003.

6 Didier Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290(1):79–115, 2003.

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

8 K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *Proc. FoSSaCS*, pages 490–501, 2005.

9 Alexander Kartzow. Collapsible Pushdown Graphs of Level 2 are Tree-Automatic. In *STACS*, pages 501–512, 2010.

10 T Knapik, D Niwinski, and P Urzyczyn. Higher-Order Pushdown Trees are Easy. In *Proc. FoSSaCS*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

11 T Knapik, D Niwinski, P Urzyczyn, and I Walukiewicz. Unsafe Grammars and Panic Automata. In *Proc. ALP*, volume 3580, pages 1450–1461, Berlin/Heidelberg, 2005. Springer-Verlag.

12 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, volume 44, pages 416–428. ACM, 2009.

13 C.-H.L. Ong and Steven J Ramsay. Verifying Higher-Order Functional Programs with Pattern-Matching Algebraic Data Types. In *POPL*, 2011.

14 Pawel Parys. Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown. In *STACS*, 2011.

15 E. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, April 1946.

# A Undecidability

## A.1 Post's Correspondence Problem

### Lemma 13

Let $P$ be an instance of Post's Correspondence Problem. $P$ has a solution just in case the automaton $\mathcal{A}_1^P$ can generate a stack $s$ such that:

- $s_u = s_v$ where $s_u$ is the subsequence of $s$ consisting of elements of the form $i_u$ and $s_v$ of elements of the form $i_v$ and equality is interpreted with respect to the indices $i$ only.
- The top two elements of $s$ form the set $\{i_u, i_v\}$ for some $1 \leq i \leq m$.

**Proof.** This is a quick consequence of definitions. Suppose that there is a solution $\sigma = u_{i_1} u_{i_2} \cdots u_{i_k} = v_{i_1} v_{i_2} \cdots v_{i_k}$ to $P$. Then $\mathcal{A}_1^P$ may construct a stack meeting the criteria by pushing the letters in $\sigma$ onto the stack whilst also pushing $i_{r\,u}$ onto the stack after completing the pushing of $u_{i_r}$ and likewise pushing $i_{r\,v}$ after completing $v_{i_r}$.

Conversely suppose that there is a stack $s$ meeting the criteria. Let $\sigma := \pi_\Sigma(s)$, which we claim is a string generated by a solution to $P$. Let $i_{r\,u}$ be the $r$th element of the form $j_u$ in the stack. Let $i_{r\,v}$ be the $r$th element of the form $j_v$ in the stack. Note that this is well defined since the first criterion requires the subsequences of $j_u$ and and $j_v$ to be the same. Let $k$ be the length of these two sequences.

Define $i_{0\,u}$ and $i_{0\,v}$ to be the bottom of the stack. Divide $\sigma$ into segments $u'_r$ (for $1 \leq r \leq k$) where $u_{r+1}$ is the subsequence of letters (in $\Sigma$) from $s$ that begins immediately after $i_{r\,u}$ and ends immediately before $i_{r+1\,u}$. Define $v'_r$ in a similar manner using $i_{r\,v}$. The first criterion ensures that $u'_r = u_{i_r}$ and that $v'_r = v_{i_r}$ for every $1 \leq r \leq k$. Moreover the second criterion ensures that $u'_1 u'_2 \cdots u'_k = v'_1 v'_2 \cdots v'_k = s$. That is $u_{i_1} u_{i_2} \cdots u_{i_k} = v_{i_1} v_{i_2} \cdots v_{i_k} = s$ and so we do indeed have a solution to $P$. ◀

## A.2 Post's Correspondence Problem and 2-CPDA

### Lemma 20

Let $(\boldsymbol{guess}, s)$ be a reachable configuration of $\mathcal{A}_{2+}^P$ for some instance $P$ of Post's Correspondence Problem. Then $s$ represents a solution to $P$ in the sense of Lemma 16 if and only if there exists a chain of stacks $s_1, s_2, \ldots, s_k$ such that $s_1 = 0_s$, $s_k = \boldsymbol{end}_s$ and $s_{i+1} = s_i^+$ for each $1 \leq i \leq k-1$ and such that for each $i$ there exists a reachable $s_i$-verifier.

**Proof.** From the definitions of $+$ and the relationship between $u_1$ and $u_2$ in each verifier configuration, it follows that the top element of $u_2$ is the element coming after the top element of $u_2^+$ in the subsequence of elements in $top_2(s)$ consisting of elements of the form $i_u$. The same holds for $v_2$ and $v_2^+$ and the subsequence of elements in $top_2(s)$ consisting of elements of the form $i_v$.

Also note that the top elements of $u_2$ and $v_2$ in $0_s$ will respectively be the last $i_u$ and $i_v$ in each of these subsequences and the top elements of $u_1$ and $v_1$ in $\boldsymbol{end}_s$ will be the first (since by definition of $\boldsymbol{end}_s$ there cannot be any $i_u$ or $i_v$ lying below the top element of $u_1$ and $v_1$ respectively).

Now we can get by an easy induction on $r$ that if there is a chain $s_1, s_2, \ldots, s_r$ such that $s_1 = 0_s$ and $s_{i+1} = s_i^+$ for each $1 \leq i < r$, then the following three assertions are true:

- The top element of the $u_2$ of $s_r$ is the $r$th element from the end of the subsequence of $top_2(s)$ consisting of elements of the form $i_u$.
- The top element of the $v_2$ of $s_r$ is the $r$th element from the end of the subsequence of $top_2(s)$ consisting of elements of the form $i_v$.
- If the top element of the $u_2$ of $s_j$ for $1 \leq j \leq r$ is $i_{j\,u}$, then the top element of the $v_2$ of $s_j$ is $i_{j\,v}$.

As a consequence, we get that the existence of such a chain implies that the final $r$ elements of the subsequence of $top_2(s)$ consisting of elements of the form $i_u$ matches the last $r$ elements of the subsequence of $top_2(s)$ consisting of elements of the form $i_v$.

So suppose there exists a chain $s_1, s_2, \ldots, s_k$ such that $s_1 = 0_s$, $s_k = \boldsymbol{end}_s$ and $s_{i+1} = s_i^+$ for each $1 \leq i \leq k-1$. The observations above ensure that $s$ meets the conditions of Lemma 16 and so we may conclude that $s$ does indeed witness a solution to $P$.

Conversely let us begin by assuming that $s$ witnesses a solution to $P$. It must then be the case that $top_2(s)$ satisfies the properties laid out in Lemma 16. It is again an easy induction on $r$ to see that under such circumstances one can generate a sequence $s_1, s_2, \ldots, s_r$ where $s_1 = 0_s$ and $s_{i+1} = s_i^+$ for each $1 \leq i < r$ so long as $r$ does not exceed the length of the subsequence of $top_2(s)$ consisting of elements of the form $i_u$ (which is the same as that consisting of elements of the form $i_v$). The induction step just needs to observe that we should order the $u_1^+, u_2^+, v_1^+, v_2^+$ with decreasing height, to enable the next to be formed from $pop_1$ing from the previous.

Finally observe that when $r$ reaches the length of the subsequences, the $u_2$ and $v_2$ of $s_r$ will have top elements corresponding to the initial $i_u$ and $i_v$ in the subsequences of $top_2(s)$ given by Lemma 16. This means that $s_r^+$ will have $u_1$ and $v_1$ with these same initial elements and so $u_2$ and $v_2$ must be empty. That is $s_r^+ = \boldsymbol{end}_s$. Hence we construct a chain of the required form. ◀

**Lemma 21**

There exists a $\Sigma_1$ sentence $\phi$ of $\boldsymbol{FO(TC)}$ containing only derived predicates $\overline{\psi}$ formed from $\Delta_1$ formulae such that for all instances $P$ of Post's Correspondence Problem we have:

$\mathcal{G}(\mathcal{A}_{2+}^P) \models \phi$ iff $P$ has a solution.

**Proof.** Let us first exhibit formulae witnessing the fact that the relation 'for some stack $s$ $x$ is an $s$-verifier and $y = x^+$' is $\Delta_1$-definable. The following is a $\Sigma_1$ formula representing the relation:

$$
\begin{aligned}
&\exists z.\,(\exists w_1 w_2 w_3.\,(\boldsymbol{pop_2}(x,w_1) \wedge \boldsymbol{pop_2}(w_1,w_2) \wedge \boldsymbol{pop_2}(w_2,w_3) \wedge \boldsymbol{pop_2}(w_3,z))\\
&\exists w_1' w_2' w_3'.\,(\boldsymbol{pop_2}(y,w_1') \wedge \boldsymbol{pop_2}(w_1',w_2') \wedge \boldsymbol{pop_2}(w_2',w_3') \wedge \boldsymbol{pop_2}(w_3',z))) \wedge\\
&\bigvee_{\substack{\{a,b,c,d\}\\ =\{u_1,u_2,v_1,v_2\}}} \boldsymbol{abcd}(x) \;\wedge \bigvee_{\substack{\{a,b,c,d\}\\ =\{u_1,u_2,v_1,v_2\}}} \boldsymbol{abcd}(y) \;\wedge\\
&((\boldsymbol{u_1u_2v_1v_2}(x) \wedge \boldsymbol{u_1u_2v_1v_2}(y)) \rightarrow ((\exists w_1 w_2.\exists w_1' w_2' w_3'.\exists c_u.\\
&\boldsymbol{pop_2}(x,w_1) \wedge \boldsymbol{pop_2}(w_1,w_2) \wedge \boldsymbol{collapse}(w_2,c_u)\\
&\boldsymbol{pop_2}(y,w_1') \wedge \boldsymbol{pop_2}(w_1',w_2') \wedge \boldsymbol{pop_2}(w_2',w_3') \wedge \boldsymbol{collapse}(w_3',c_u))\\
&\wedge(\exists w_1'.\exists c_v.\boldsymbol{collapse}(x,c_v) \wedge \boldsymbol{pop_2}(y,w_1') \wedge \boldsymbol{collapse}(w_1',c_v)))) \wedge \cdots\\
\\
&\cdots \wedge ((\boldsymbol{v_1u_1u_2v_2}(x) \wedge \boldsymbol{u_1v_1u_2v_2}(y)) \rightarrow ((\exists w_1.\exists w_1' w_2' w_3'.\exists c_u.\\
&\boldsymbol{pop_2}(x,w_1) \wedge \boldsymbol{collapse}(w_1,c_u) \wedge\\
&\boldsymbol{pop_2}(y,w_1') \wedge \boldsymbol{pop_2}(w_1',w_2') \wedge \boldsymbol{pop_2}(w_2',w_3') \wedge \boldsymbol{collapse}(w_3',c_u))\\
&\wedge(\exists w_1' w_2'.\exists c_v.\boldsymbol{collapse}(x,c_v) \wedge\\
&\boldsymbol{pop_2}(y,w_1') \wedge \boldsymbol{pop_2}(w_1',w_2') \wedge \boldsymbol{collapse}(w_2',c_v))))\\
\\
&\hspace{30em}\wedge \cdots
\end{aligned}
$$

We have only mentioned two of the $6 \times 8 + 2 \times 8 = 64$ elements of the final conjunction in the formula above, but the remainder follow the same pattern. There are 6 different predicates that characteise the order of the $u_1, u_2, v_1 v_2$ in $x$ but 8 for $y$ as these may enjoy the $\boldsymbol{end}$ flag and then there are additionally two with a $\boldsymbol{start}$ flag that $x$ may exhibit. The formula correctly captures the relation $y = x^+$ since the pointer from each atomic element in the stack is assigned a different target to its 2-pointer when it is created. In the final conjunction, $c_u$ represents the common target of the $u_2$ in $x$ and $u_1$ in $y$ and $c_v$ the common target of the $v_2$ in $x$ and the $v_1$ in $y$. The first clauses of the formula assert that $x$ and $y$ are both $s$-verifiers for some particular fixed stack $s$ (embodied by the configuration bound to $z$).

We now exhibit a $\Pi_1$ formula defining this relation, thereby showing that the relation is $\Delta_1$. This is effectively a rehashing of the $\Sigma_1$ formula above, exploiting the fact that all of the relations used are destructive operations on some fixed stacks and are consequently 'functional' (the result of a particular destructive operation on a fixed stack always gives a unique result):

$$\forall z. \forall w_1 w_2 w_3. \forall w_1' w_2' w_3' \, (\boldsymbol{pop_2}(x, w_1) \to \boldsymbol{pop_2}(w_1, w_2) \to \boldsymbol{pop_2}(w_2, w_3) \to \boldsymbol{pop_2}(w_3, z)$$

$$\to \boldsymbol{pop_2}(y, w_1') \to \boldsymbol{pop_2}(w_1', w_2') \to \boldsymbol{pop_2}(w_2', w_3') \to \boldsymbol{pop_2}(w_3', z)) \wedge$$

$$\bigvee_{\substack{\{a,b,c,d\} \\ = \{u_1, u_2, v_1, v_2\}}} \boldsymbol{abcd}(x) \; \wedge \bigvee_{\substack{\{a,b,c,d\} \\ = \{u_1, u_2, v_1, v_2\}}} \boldsymbol{abcd}(y) \; \wedge$$

$$((\boldsymbol{u_1 u_2 v_1 v_2}(x) \wedge \boldsymbol{u_1 u_2 v_1 v_2}(y)) \to ((\forall w_1 w_2. \forall w_1' w_2' w_3'. \forall c_u.$$

$$\boldsymbol{pop_2}(x, w_1) \to \boldsymbol{pop_2}(w_1, w_2) \to \boldsymbol{collapse}(w_2, c_u)$$

$$\boldsymbol{pop_2}(y, w_1') \to \boldsymbol{pop_2}(w_1', w_2') \to \boldsymbol{pop_2}(w_2', w_3') \to \boldsymbol{collapse}(w_3', c_u))$$

$$\wedge (\forall w_1'. \forall c_v. \boldsymbol{collapse}(x, c_v) \to \boldsymbol{pop_2}(y, w_1') \to \boldsymbol{collapse}(w_1', c_v)))) \wedge \cdots$$

$$\cdots \wedge ((\boldsymbol{v_1 u_1 u_2 v_2}(x) \forall \boldsymbol{u_1 v_1 u_2 v_2}(y)) \to ((\forall w_1. \forall w_1' w_2' w_3'. \forall c_u.$$

$$\boldsymbol{pop_2}(x, w_1) \to \boldsymbol{collapse}(w_1, c_u) \to$$

$$\boldsymbol{pop_2}(y, w_1') \to \boldsymbol{pop_2}(w_1', w_2') \to \boldsymbol{pop_2}(w_2', w_3') \to \boldsymbol{collapse}(w_3', c_u))$$

$$\wedge \, (\forall w_1' w_2'. \forall c_v. \boldsymbol{collapse}(x, c_v) \to$$

$$\boldsymbol{pop_2}(y, w_1') \to \boldsymbol{pop_2}(w_1', w_2') \to \boldsymbol{collapse}(w_2', c_v))))$$

$$\wedge \cdots$$

Let $\phi^+(x, y)$ be either of the formulae above.We can now define what it means to have a chain from $x$ to $y$ of $s$-verifiers (for some $s$) in $\boldsymbol{FO(TC)}$ using transitive closure on only a $\Delta_1$-definable relation:

$$\overline{\phi^+}(x, y)$$

We can easily assert that $y$ is equal to $\boldsymbol{end}_s$ for some stack $s$ with:

$$\boldsymbol{u_1 v_1^{end}} \vee \boldsymbol{v_1 u_1^{end}}$$

Likewise we can assert that $x$ is equal to $0_s$ for some stack $s$ with:

$$\boldsymbol{u_1 u_2 v_1 v_2^{start}}(x) \vee \boldsymbol{v_1 v_2 u_1 u_2^{start}}(x)$$

By Lemma 20 we can construct the required $\Sigma_1$ sentence $\phi$ by putting all of the above together to get:

$$\exists x. \exists y. \left( (\boldsymbol{u_1 u_2 v_1 v_2^{start}}(x) \vee \boldsymbol{v_1 v_2 u_1 u_2^{start}}(x)) \wedge \bigvee_{\substack{\{a,b,c,d\} \\ = \{u_1, u_2, v_1, v_2\}}} \boldsymbol{abcd^{end}}(y) \wedge \overline{\phi^+}(x, y) \right)$$

◀

## A.3 Undecidability for $3_2$-CPDA

We give the full definition of $\mathcal{A}^P_{3_2}$:

### Definition 22

The $3_2$-CPDA $\mathcal{A}^P_{3_2}$ behaves as follows:

- It begins by behaving in the same way as $\mathcal{A}^P_{2+}$, performing only 2-stack operations until it reaches the control-state $\boldsymbol{u_1u_2v_1v_2}^{\boldsymbol{start}}$ or $\boldsymbol{v_1v_2u_1u_2}^{\boldsymbol{start}}$. If it is unable to reach such a state, it goes into a distinguished $\boldsymbol{fail}$ state and aborts.
- The automaton then pushes a record of its $\mathcal{A}^P_{3_2}$ control-state on to the stack and performs $push_3; pop_2; pop_2; pop_2; pop_2;$. This will return the stack to the original $\mathcal{A}^P_{2+}$ $\boldsymbol{guess}$-configuration. It then behaves from here as though it were $\mathcal{A}^P_2$ in the $\boldsymbol{guess}$ control state (with a stack $s$) until it reaches an $s$-verifier-configuration (or finds it is unable to, in which case it aborts).
- The previous step is repeated until an $\boldsymbol{end}_s$ configuration is reached, at which point the CPDA pushes the $\mathcal{A}^P_{2+}$ control-state onto the stack and enters a distinguished $\boldsymbol{guess_{3_2}}$ control-state.

The following transitions are then added:

- An $\epsilon$-transition going from any configuration to a distinguished control-state $\boldsymbol{prototest}$ via a $pop_3$ operation.
- The *only* transition going from a $\boldsymbol{prototest}$ control-state is allowed when the $top_1$ stack-symbol is one of: $\boldsymbol{u_1u_2v_1v_2}$, $\boldsymbol{u_1v_1u_2v_2}$, $\boldsymbol{u_1v_1v_2u_2}$, $\boldsymbol{v_1u_1u_2v_2}$, $\boldsymbol{v_1u_1v_2u_2}$, $\boldsymbol{v_1v_2u_1u_2}$ (in particular when the top symbol does not have a $\boldsymbol{start}$ or $\boldsymbol{end}$ flag). This transition is given the label $\boldsymbol{toTest}$ and moves to control-state $\boldsymbol{test}$ with no stack operation.
- We also allow a $\boldsymbol{toTest}$-labelled transition from a configuration with control-state $\boldsymbol{guess_{3_2}}$ to control-state $\boldsymbol{test}$ whilst not performing any stack operation.

We further add these transitions:

- A transition $\boldsymbol{to2^+}$ performing a $pop_1$ and entering control-state $\boldsymbol{abcd}$ whenever in control-state $\boldsymbol{test}$ with top stack element $\boldsymbol{abcd}$ where $\boldsymbol{abcd}$ is a control-state of $\mathcal{A}^P_{2+}$. From this control-state $\mathcal{A}^P_{3_2}$ behaves in the same way as $\mathcal{A}^P_{2+}$, performing only order-2 operations.
- From any configuration with $\boldsymbol{test}$ control-state there exist transitions labelled $\boldsymbol{push_3}$ and $\boldsymbol{pop_3}$ performing the respective stack operations whilst remaining in control-state $\boldsymbol{test}$.

### Lemma 24

There exists a $\Sigma_2$ sentence $\phi$ of $\boldsymbol{FO}$ such that for all instances $P$ of Post's Correspondence Problem we have: $\mathcal{G}^\epsilon(\mathcal{A}^P_{3_2}) \models \phi$ iff $P$ has a solution.

**Proof.** Let us first interpret Lemma 20 in the context of $\mathcal{A}^P_{3_2}$. Since $\mathcal{A}^P_{3_2}$ is designed to generate arbitrary verifiers for some stack $s$ created at the outset, beginning with $0_s$ and ending with $\boldsymbol{end_s}$, Lemma 20 implies that there is a solution to $P$ if and only if $\mathcal{A}^P_{3_2}$ can reach a configuration of the form $(\boldsymbol{guess_{3_2}}, s)$ such that for every pair of 2-stacks $t, t'$ occurring in $s$ with $t'$ immediately above $t$, $t' = t^+$.

We already have a $\Pi_1$ formula $\psi(x, y)$ in first-order logic expressing that $y = x^+$ where $x$ and $y$ range over 2-stacks reachable by $\mathcal{A}^P_{2+}$. This was given in the proof of Lemma 21. Now observe that for any 3-stack $s$ and sequence of 2-*stack* operations $\vec{op}$ we have $\vec{\theta}(s) = \vec{\theta}(pop_3(push_3(s)))$ if and only if $\vec{\theta}(top_3(s)) = \vec{\theta}(top_3(pop_3(s)))$. We can thus exploit the $\mathcal{A}^P_{2+}$-simulation feature of $\mathcal{A}^P_{3_2}$ to express that some variable $x$ bound to a configuration of $\mathcal{A}_{3_2}$ of the form $(\boldsymbol{test}, s)$ has the property that $top_3(pop_3(s))^+ = top_3(s)$. This can be done by the $\Pi_1$ formula $\chi(x)$:

$$\forall y_1 y_2 y.\forall z.(\boldsymbol{pop_3}(x, y_1) \rightarrow \boldsymbol{push_3}(y_1, y_2) \rightarrow \boldsymbol{to2}^+(y_2, y) \rightarrow \boldsymbol{to2}^+(x, z) \rightarrow \psi(y, z))$$

over both the graph $\mathcal{G}(\mathcal{A}^P_{3_2})$ and $\mathcal{G}^\epsilon(\mathcal{A}^P_{3_2})$ ($\mathcal{A}^P_{3_2}$ has no $\epsilon$-transitions reachable from a configuration with control-state $\boldsymbol{test}$).

The construction of $\mathcal{A}^P_{3_2}$ ensures that the configurations we can reach from $(\boldsymbol{guess_{3_2}}, s)$ by a (possibly empty) sequence of $\epsilon$-transitions followed by a $\boldsymbol{toTest}$-transition are precisely those of the form $(\boldsymbol{test}, s')$ such that $s'$ is a prefix of $s$ whose 2-stacks all occur in $s$. Hence these stacks are precisely those reachable by a $\boldsymbol{toTest}$-transition in the $\epsilon$-closure of the configuration graph. Thus combining all of the observations above the required $\Sigma_2$ sentence $\phi$ is:

$$\phi := \exists x.\forall y(\boldsymbol{guess_{3_2}}(x) \wedge \boldsymbol{toTest}(x, y) \wedge \chi(y) \vee (\boldsymbol{u_1 u_2 v_1 v_2^{start}}(y) \vee \boldsymbol{v_1 v_2 u_1 u_2^{start}}(y)))$$

(disregarding when $y$ is a verifier with the $\boldsymbol{start}$ flag as this is at the bottom and so there is no stack to compare below this). ◀

## A.4 The Non-Locality of $3_2$-CPDA

▶ **Definition 39.** Let $P$ be an instance of Post's Correspondence Problem. The $3_3$-CPDA $\mathcal{A}^P_{3_3}$ has stack alphabet:

$$\Sigma \uplus [1_u, 2_u, \ldots, m_u] \uplus [1_v, 2_v, \ldots, m_v] \uplus \{\boldsymbol{u_1 u_2 v_1 v_2^{start}}, \boldsymbol{v_1 v_2 u_1 u_2^{start}}, \boldsymbol{u_1 v_1^{end}}, \boldsymbol{v_1 u_1^{end}}\}$$
$$\uplus \{\boldsymbol{abcd} \ : \ \{a, b, c, d\} = \{u_1, u_2, v_1, v_2\}\} \uplus \{\bullet\}$$

It initially behaves by non-deterministically choosing one of the following:
- Push any member of $\Sigma$ onto the stack.
- If the $\Sigma$ symbols in the stack since the last symbol of the form $i_u$ in the top 1-stack (or the bottom of the 1-stack if there is no such symbol) form the word $u_j$, then it may perform $push_3; push_1^{j_u,3}$.
- If the $\Sigma$ symbols in the stack since the last symbol of the form $i_v$ in the top 1-stack (or the bottom of the 1-stack if there is no such symbol) form the word $v_j$, then it may perform $push_3; push_1^{j_v,3}$.

Finite control-states enforce the precondition on the second and third options.

Once the top two elements of the stack are of the form $i_u, i_v$ for some $i$ (in either order), the automaton enters the next phase and generate the first element of the verification chain:
- Perform $push_2; push_2$
- Perform $pop_1; push_2; push_2$
- Either push $\boldsymbol{u_1 u_2 v_1 v_2^{start}}$ or $\boldsymbol{v_1 v_2 u_1 u_2^{start}}$ onto the stack depending on whether the top two elements were in the order $i_u i_v$ or $i_v i_u$.

The automaton then iteratively produces further candidates for elements in the verification chain:
- Perform $push_3$.
- Perform $pop_2; pop_2 pop_2 pop_2$
- Perform $push_1^{\bullet,3}$
- Perform $push_2$ followed by iterated $pop_1$ until the top element of the stack is no longer a $\bullet$.
- Generate a further four 2-stacks representing a $u_1, u_2, v_1, v_2$ in exactly the same way as for $\mathcal{A}^P_{3_2}$, recording the ordering (possibly with an end flag) on top.
- The automaton breaks from this iteration at this point iff an element with an $\boldsymbol{end}$ flag was just deployed.

After this phase the automaton enters a distinguished control-state $\boldsymbol{candidate}$. It then *leaves* this control-state to perform $push_3; pop_2; pop_2; pop_2; pop_2$ and enters control-state $\boldsymbol{chainpos}$. It then repeatedly performs the following:

- $pop_1$ entering a non-distinguished control-state.
- If the top element is a $\bullet$ it enters control-state $\boldsymbol{chainpos}$ and goes back to the item above.

We additionally add transitions from all configurations labelled by $\boldsymbol{pop_1}$, $\boldsymbol{pop_2}$, $\boldsymbol{pop_3}$ and $\boldsymbol{collapse}$ performing the respective stack operations whilst transitioning to a distinguished control-state $\boldsymbol{test}$.

**Lemma 25**

Let $P$ be an instance of Post's Correspondence Problem. Then there exists a $\Sigma_2$ sentence $\phi$ of $\boldsymbol{FO}$ such that:

$$\mathcal{G}(\mathcal{A}^P_{3_3}) \vDash \phi$$

(note *no* $\epsilon$-closure) if and only if $P$ has a solution.

**Proof.** For the same reasons as with $\mathcal{A}^P_{3_2}$ it is the case that $P$ has a solution if and only if $\mathcal{A}^P_{3_3}$ can reach a configuration $(\boldsymbol{candidate}, t)$ such that, for every pair of 2-stacks $s, s'$ occurring in $t$ with $s'$ is the 2-stack immediately above $s$, it is the case that $s' = s^+$.

Assuming that the variable $x$ is bound to a configuration $(\boldsymbol{test}, t')$ where $t'$ is an initial segment of a stack $t$ such that $(t, \boldsymbol{candidate})$ is reachable, we can assert that the top two 2-stacks of $t'$ form a pair with the credentials above using the following $\Pi_1$ formula $\psi(x)$ over $\mathcal{G}(\mathcal{A}_{3_3})$:

$$\begin{aligned}
&\forall s'.\forall u_2 v_2 u_1 v_1.\forall w w'.\forall c_u c_v.(\boldsymbol{pop_3}(x, s') \rightarrow \boldsymbol{u_1 u_2 v_1 v_2}(x) \rightarrow \boldsymbol{u_1 u_2 v_1 v_2}(s') \\
&\rightarrow \boldsymbol{pop_2}(x, v_1) \rightarrow \boldsymbol{pop_2}(v_1, w) \rightarrow \boldsymbol{pop_2}(w, u_1) \rightarrow \boldsymbol{pop_1}(s', v_2) \\
&\rightarrow \boldsymbol{pop_2}(s', w') \rightarrow \boldsymbol{pop_2}(w', u_2) \rightarrow \boldsymbol{collapse}(u_1, c_u) \rightarrow \boldsymbol{collapse}(v_1, c_v) \\
&\rightarrow (\boldsymbol{collapse}(u_2, c_u) \wedge \boldsymbol{collapse}(v_2, c_v))) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge \cdots
\end{aligned}$$

where as with the proof of Lemma 21 we have illustrated just one of 64 conjuncts that go through all possible orderings of the $u_1, u_2, v_1, v_2$ in the top and penultimate 2-stacks. The correctness of this formula follows from the fact that every element constituting the guess of a solution of $P$ is equipped with a 3-link pointing to a distinct 2-stack.

The design of $\mathcal{A}^P_{3_3}$ ensures that the elements in the verification-chain contained in $t$ (*i.e.* all candidates for the pairing $s, s'$ in $t$) are precisely those that can be reached by performing a $collapse$ operation on a $(\boldsymbol{chainpos}, t')$ configuration reached from $(\boldsymbol{candidate}, t)$. Assuming that $x$ is again bound to a $(\boldsymbol{candidate}, t)$ configuration, we can thus assert that $x$ represents a solution to $P$ with the following $\Pi_1$ formula $\chi(x)$ over $\mathcal{G}(\mathcal{A}^P_{3_3})$:

$$\begin{aligned}
\chi(x) := \forall y.\forall z.(&\boldsymbol{chainpos}(y) \rightarrow \boldsymbol{pop_3}(y, x) \rightarrow \boldsymbol{collapse}(y, z) \\
&\rightarrow \neg(\boldsymbol{u_1 u_2 v_1 v_2}^{start}(z) \vee \boldsymbol{v_1 v_2 u_1 u_2}^{start}(z)) \rightarrow \psi(z))
\end{aligned}$$

We can thus take the required $\Sigma_2$ formula $\phi$ (asserting the existence of such a solution witnessing configuration $x$) to be:

$$\phi := \exists x.(\boldsymbol{candidate}(x) \wedge \chi(x))$$

◀

### A.5 $\Sigma_1$ Undecidability for $4_2$-CPDA

#### A.5.1 The modification

We modify the verification chain so that only one pair of collapses needs to be compared for equality rather than previously where two comparisons were required (for the $u$ and the $v$). This allows us to avoid the problem above.

Proceeding with our running example, a solution to a PCP instance will be represented using a 2-stack in exactly the same manner as before:

$$[[ab\ ]\ [ab\ \mathbf{1_u}\ ab\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [.\ ]\ [ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ab\ \mathbf{2_v}\ b\ \mathbf{2_u}\ ca\ \mathbf{3_u}\ \mathbf{3_v}]\ ]$$

The difference is the manner in which the 'verification chain' used to check the correctness of an alleged representation is constructed and represented in a 3-stack. In this modified chain of 2-stacks it is the top three 1-stacks that are significant. Each element in the stack will posess either '$u$ $v_1$ and $v_2$' 1-stacks (in some order) or $v$, $u_1$ and $u_2$ 1-stacks. This contrasts with the original verification chaion where each point posessed a $u_1$, $u_2$, $v_1$ and $v_2$ 1-stack.

We refer to the $u$ or $v$ 1-stack in a chain position as *guaranteed* and the $u_2$ or $v_2$ stack as *tentative* with the associated $u_1$ or $v_1$ as the *condition stack*. If the condition stack represents the same position in either the $u$ or $v$ subsequence from the previous member in the chain, then the tentative stack correctly represents the next position in the $u$ or $v$ subsequence. This is the same idea as before. The guaranteed stack, however, will always *unconditionally* represent the next position in the $v$ or $u$ subsequence.

The following shows this new kind of a verification chain as a 3-stack:

$$[\ |\ |\ |\ |\ |\ |\ |\ |\ |\ [ab\ \mathbf{1_u}\ ][][\boldsymbol{u_1^{end}}]]$$

$$[\ |\ |\ |\ |\ |\ |\ |\ |\ |\ [ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ ][ab\ \mathbf{1_u}\ \boldsymbol{v u_1 u_2}]]$$

$$[\ |\ |\ |\ |\ |\ |\ |\ |\ |\ [ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ab\ \mathbf{2_v}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ \boldsymbol{v_1 v_2 u}]]$$

$$[\ |\ |\ |\ |\ |\ |\ |\ |\ |\ [ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ab\ \mathbf{2_v}\ b\ \mathbf{2_u}\ ca\ \mathbf{3_u}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ ab\ \mathbf{2_v}\ b\ \mathbf{2_u}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ab\ \mathbf{2_v}\ \boldsymbol{u_1 u_2 v}]\ ]$$

$$[\ |\ |\ |\ |\ |\ |\ |\ |\ |\ [ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ ab\ \mathbf{2_v}\ b\ \mathbf{2_u}\ ca\ \mathbf{3_u}\ \mathbf{3_v}\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ab\ \mathbf{2_v}\ b\ \mathbf{2_u}\ ca\ ][ab\ \mathbf{1_u}\ ab\ \mathbf{1_u}\ c\ \mathbf{1_v}\ ababc\ \mathbf{1_v}\ ab\ \mathbf{2_v}\ b\ \mathbf{2_u}\ ca\ \mathbf{3_u}\ \boldsymbol{v u_1 u_2^{start}}]]\ ]$$

As before, a token describing the ordering of the stacks is added to the top of each element in the chain.

Let us formally define a $3_2$-CPDA that generates verification chains in the manner illustrated in the example above.

▶ **Definition 40.** Let $P$ be an instance of PCP. The $3_2$-CPDA $\mathcal{A}^P_{3_2^{alt}}$ shares stack alphabet with $\mathcal{A}^P_2$ but with extra symbols: $\boldsymbol{vu_1u_2}$, $\boldsymbol{u_1vu_2}$, $\boldsymbol{u_1u_2v}$, $\boldsymbol{vu_1u_2^{start}}$, $\boldsymbol{u_1^{end}}$, $\boldsymbol{uv_1v_2}$, $\boldsymbol{v_1uv_2}$, $\boldsymbol{v_1v_2u}$, $\boldsymbol{uv_1v_2^{start}}$, $\boldsymbol{v_1^{end}}$.

$\mathcal{A}^P_{3_2^{alt}}$ initially behaves the same way as $\mathcal{A}^P_2$ in order to generate a 2-stack representing a postulated solution to $P$. At this point the top two elements of the stack will either be $i_ui_v$ or $i_vi_u$ for some $i \in [1..m]$. We then perform $push_2; push_2; pop_1; push_2$ followed by $push_1^{\boldsymbol{uv_1v_2^{start}}}$ if the top two symbols are $i_vi_u$ and $push_1^{\boldsymbol{vu_1u_2^{start}}}$ if $i_ui_v$. The automaton then performs $push_3$ and behaves as follows, examining the token on top of the stack to ascertain which option should be taken:

- If the current guaranteed stack of the $top_3$ element (which has just been copied) is below both the condition and tentative stacks, then we should perform $pop_2$ until the guaranteed stack is the $top_2$ stack—*i.e.* perform $pop_2; pop_2$. Then:
  - If the guaranteed stack is a $u$ stack (rather than a $v$ stack) then perform $pop_1$ until another $j_u$ or $j_v$ is found for some $j \in [1..m]$.
    - At the *first new* $j_u$ to be discovered, we deem the resulting $top_2$ to be the new $u$ stack. If $v_1$ and $v_2$ are still to be produced we perform $push_2$ and continue with the $pop_1$s.
    - At all *subsequent* $j_u$ discovered, we just perform $pop_1$ and continue with the $pop_1$s.
    - If a $j_v$ is found and we have not yet created a new $v_1$, then we non-deterministically choose whether to deem this stack to be the new $v_1$. If we choose not to we just proceed with more $pop_1$s. If we choose to do so, then we perform $push_2$.
    - The *first new* $j_v$ to be found *since creating* $v_1$ should become the new $v_2$ stack. If the new $u$ stack has not yet been created then we perform $push_2$ and continue with the popping.
    - All *subsequent* $j_v$ found *since creating* the new $v_2$ should just be popped and the $pop_1$s continued.
    - If the empty 1-stack is produced, then abort (and fail) if either $u_1$ has not yet been generated or either $u_2$ or $v$ have been generated. Otherwise perform $push_2; push_1^{\boldsymbol{u_1^{end}}}$.
    - If the new $v$, $u_1$ and $u_2$ have all been created, then cease the $pop_1$s and perform $push_1^{\boldsymbol{token}}$ where $\boldsymbol{token}$ is the token (without a start or end flag) denoting the order in which the stacks were produced.
  - If the guaranteed stack is a $v$ stack (rather than a $u$ stack) then do the same as above interchanging $u$ and $v$.
- If the condition stack of the $top_3$ element in the chain is below both the guaranteed and tentative stacks, then we should perform $pop_2$ until the condition stack is the $top_2$ stack—*i.e.* perform $pop_2; pop_2$. Then:
  - If the condition stack is a $u_1$ stack (rather than a $v_1$ stack) then perform $pop_1$ until another $j_u$ or $j_v$ is found for some $j \in [1..m]$.
    - At the *first new* $j_u$ to be discovered we just perform $pop_1$ and continue with the $pop_1$ operations. At the *second new* $j_u$ to be discovered, we deem the current $top_2$ stack to be the new $u$ stack. If the new $v_1$ and $v_2$ have not yet been created we perform $push_2$ and continue with the $pop_1$ operations.
    - At all *subsequent* $j_u$ discovered, we just perform $pop_1$ and continue with the $pop_1$s.
    - If a $j_v$ is discovered then we behave in the same way as in the case when the guaranteed stack $u$ is below the condition stack $v_1$ (see above). This creates the $v_1$ and $v_2$ stacks.
    - If an empty 1-stack is produced or we have just finished creating all of the new $u$, $v_1$ and $v_2$ then again we behave in the same manner as when the guaranteed stack is below the condition stack.
- Note that the condition stack will always be below the tentative stack, so we have already exhausted the possibilities.

- If it is not the case that $top_1(u) = i_u$ and $top_1(v_2) = i_v$ or $top_1(u_2) = i_u$ and $top_1(v) = i_v$ (depending on which combination of stacks the recently produced chain element uses) for some $i \in [1..m]$ then the automaton aborts into a fail state.
- If we have deployed a token with an $\boldsymbol{end}$ marker, then we halt and move into a distinguished control-state $\boldsymbol{guess_{3_2}}$. Otherwise we perform a $push_3$ operation and repeat.

▶ **Remark.** Suppose that an element in a chain generated by $\mathcal{A}^P_{3^{alt}_2}$ consists of a $v$ and $u_1$, $u_2$ stacks. If $v$ is the first (lowest) of these, then the next element in the chain will consist of a $v$, $u_1$ and $u_2$. If $u_1$ is the lowest, then the next element in the chain will consist of a $u$, $v_1$ and $v_2$.

We formalise what it means to be a correct verification chain in this new style using a revised successor operation $\oplus$ to be to $\mathcal{A}^P_{3^{alt}_2}$ what $+$ is to $\mathcal{A}^P_{3_2}$.

▶ **Definition 41.** Let $s$ be a 2-stack over the alphabet of $\mathcal{A}^P_2$. The *successor* $s^{\oplus}$ of $s$ is the unique stack such that:

- $pop_2; pop_2; pop_2(s^{\oplus}) = pop_2; pop_2; pop_2(s)$
- Where $a, b, c$ are (in order) the top three 1-stacks of $s^{\oplus}$: $c \sqsubseteq_1 b \sqsubseteq_1 a$.
- If the bottom of the top three 1-stacks in $s$ is a $u$ or $u_1$ stack, then $s^{\oplus}$ should consist of a $u$, $v_1$ and $v_2$ stack, if its a $v$ or $v_1$ stack then, $s^{\oplus}$ should consist of a $v$, $u_1$ and $u_2$ stack.
- Let $s_u$ be either the $u_2$ or $u$ stack in $s$ (whichever $s$ posesses). Let $s_v$ be either the $v_2$ or $v$ stack in $s$ (whichever $s$ posesses).
  - If $s^{\oplus}$ posesses $u_1$, then this $u_1 = s_u$ and its $u_2$ has as its $top_1$ element the highest element of the form $i_u$ below $top_1(s_u)$. If such an element does not exist, $u_2$ should be empty. Moreover the $v$ of $s^{\oplus}$ should have as its $top_1$ element the highest element of the form $i_v$ below $top_1(s_v)$. If such an element does not exist, $v$ should be empty.
  - If $s^{\oplus}$ posesses $v_1$, then this $v_1 = s_v$ and its $v_2$ has as its $top_1$ element the highest element of the form $i_v$ below $top_1(s_v)$. If such an element does not exist,$v_2$ should be empty. Moreover the $u$ of $s^{\oplus}$ should have as its $top_1$ element the highest element of the form $i_u$ below $top_1(s_u)$. If such an element does not exist, $u$ should be empty.

The following Lemma is almost an immediate consequence of the definitions and Lemma 16.

▶ **Lemma 42.** *Let $P$ be an instance of Post's Correspondence Problem and let $s$ be a 2-stack generated by $\mathcal{A}^P_2$. The 2-stack $s$ represents a solution to $P$ in the sense of Lemma 16 iff there is a 'verification chain' of 2-stacks $s_1, s_2, \ldots, s_k$ for some $k$ such that:*

- *$s_1 = push_2; push_2; pop_1; push_2(s)$ (with top three stacks defined to be $v$, $u_1$, $u_2$ or $u$, $v_1$, $v_2$ if the top two elements of $s$ are respectively of the form $i_v i_u$ or $i_u i_v$)*
- *$s_k$ has empty stacks as its top two 1-stacks*
- *$s_{i+1} = s_i^{\oplus}$ for every $1 \leq i < k$.*
- *For each element $s_i$ in the chain we have $top_1(v) = j_v$ and $top_1(u_2) = j_u$ or $top_1(u) = j_u$ and $top_1(v_2) = j_v$ for some $j \in [1..m]$ (depending on what selection of stacks $s_i$ has).*

**Proof.** We establish the result by arguing that such a sequence exists iff $s$ satisfies the conditions set out in Lemma 16.

Argue by induction on $l$ that an initial segment of such a chain $s_1, s_2, \ldots, s_l$ exists with $j_1, j_2, \ldots, j_l$ the associated indices mentioned in the fourth condition iff the top-most $l$ elements of $s$ of the form $i_u$ are $(j_1)_u, (j_2)_u, \ldots, (j_l)_u$, and the topmost $l$ elements of $s$ of the form $i_v$ are $(j_1)_v, (j_2)_v, \ldots, (j_l)_v$ (ordered top down).

First the $\Rightarrow$) direction. The base case ($l = 1$) is immediate since all stacks generated by $\mathcal{A}^P_2$ must have top two elements of the form $j_u j_v$ or $j_v j_u$ for some $j$. Suppose now that an initial segment of such a chain $s_1, s_2, \ldots, s_l, s_{l+1}$ exists with fourth-condition indices $j_1, j_2, \ldots, j_l, j_{l+1}$. By the induction hypothesis, we just need to check that if the top elements of either $v$ or $v_2$ in $s_l$ and either

$u$ or $u_2$ in $s_l$ are respectively the $l$th $j_v$ and $j_u$ elements from the top of $s$, then the top elements of either $v$ or $v_2$ in $(s_l)^{\oplus}$ and either $u$ or $u_2$ in $(s_l)^{\oplus}$ are respectively the $(l+1)$th elements of the form $j_v$ and $j_u$ from the top of $s$. But this is ensured directly by the fourth point in the definition of $^{\oplus}$.

Now consider the $\Leftarrow$) direction. Again the base case ($l = 1$) is straightforward since $s_1$ is explicitly defined to meet the criteria. Suppose that the topmost $l + 1$ elements of the form $j_u$ of $s$ are: $(j_1)_u, (j_2)_u, \dots, (j_l)_u, (j_{l+1})_u$ and the topmost $l + 1$ elements of the form $j_v$ of $s$ are: $(j_1)_v, (j_2)_v, \dots, (j_l)_v, (j_{l+1})_v$. By the induction hypothesis we already have a chain $s_1, s_2, \dots, s_l$ and so we just need to show that $(s_l)^{\oplus}$ both exists and satisfies the requisite criteria. If it does exist, then as above the fourth clause in the definition of $^{\oplus}$ ensures that the fourth requirement of the Lemma is satisfied, which is the only one that would need to be established (the first applies only to $s_1$, the second is not relevant to initial prefixes and the third is by assumption). Thus it is only existence that we need to establish. But the $^{\oplus}$ successor always exists. The top three stacks are all initial segments of $s$ (exhaustively defined—they are defined to be the empty stack if an appropriate element in $s$ does not exist) and so can be linearly ordered with resepct to $\sqsubseteq_1$. ◀

The following Lemma is critical—it tells us that $\mathcal{A}^P_{3^{alt}_2}$ is able to construct an appropriate chain of successors if one exists and moreover provides a sufficient (and necessary) condition for a stack it generates representing such a chain.

▶ **Lemma 43.** *Let $P$ be an instance of Post's Correspondence Problem. For any 2-stack $s$ generated by $\mathcal{A}^P_2$ there exists a sequence of 2-stacks $s_1, s_2, \dots, s_k$ satisfying the condition in Lemma 42 iff $\mathcal{A}^P_{3^{alt}_2}$ can reach a configuration $(\boldsymbol{guess_{3_2}}, [s'_1, s'_2, \dots, s'_k])$ such that:*

- *$s'_i = push_1^{\boldsymbol{token}}(s_i)$ for each $1 \le i \le k$ where $\boldsymbol{token}$ is a token indicating the ordering of the top three stacks with a $\boldsymbol{start}$ flag for $s'_1$ and a $\boldsymbol{end}$ flag for $s'_k$.*
- *For every $1 \le i < k$:*
  - *If $s'_{i+1}$ contains a $u_1$ then this is equal to $u_2$ or $u$ in $s'_i$ (depending on which one $s'_i$ contains)*
  - *or if $s'_{i+1}$ contains a $v_1$ then this is equal to $v_2$ or $v$ in $s'_i$ (depending on which one $s'_i$ contains).*

  *Note that for each $i$ only one of the above will hold.*

**Proof.** First let us argue in the $\Rightarrow$) direction. First argue by induction on $l$ that if we have an initial prefix $s_1, s_2, \dots, s_l$ of such a sequence $s_1, s_2, \dots, s_k$, then $\mathcal{A}^P_{3^{alt}_2}$ can generate a stack $[s'_1, s'_2, \dots, s'_l]$ without aborting, during the phase after generating the 2-stack $s$, such that:

- $s'_i = push_1^{\boldsymbol{token}}(s_i)$ for each $1 \le i \le l$ where $\boldsymbol{token}$ is a token indicating the ordering of the top three stacks with a $\boldsymbol{start}$ flag for $s'_1$ and a $\boldsymbol{end}$ flag for $s'_k$.
- For every $1 \le i < l$:
  - If $s'_{i+1}$ contains a $u_1$ then this is equal to $u_2$ or $u$ in $s'_i$ (depending on which one $s'_i$ contains)
  - or if $s'_{i+1}$ contains a $v_1$ then this is equal to $v_2$ or $v$ in $s'_i$ (depending on which one $s'_i$ contains).

  Note that for each $i$ only one of the above will hold.

For the base case ($l = 1$) the result is immediate as $\mathcal{A}^P_{3^{alt}_2}$ will construct $s'_1$ from $s$ in exactly the way that $s_1$ is defined in Lemma 42 (adding a token on top).

For the inductive step, suppose that $\mathcal{A}^P_{3^{alt}_2}$ has already generated $[s'_1, s'_2, \dots, s'_l]$ corresponding to a sequence $s_1, s_2, \dots, s_l$. Suppose now that this sequence extends to $s_1, s_2, \dots, s_l, s_{l+1}$. The automaton will perform $push_3; pop_2 pop_2$ in preparation to generate $s'_{l+1}$. We consider two cases:

*Case when the guaranteed stack of $s'_l$ (or by induction hypothesis equivalently $s_l$) is below both the condition and tentative stacks:* W.l.o.g. assume that the guaranteed stack of $s'_l$ is a $u$ stack (the case when it is a $v$ stack is similar). Due to the position of $v$ we must have in $s_l$: $v \sqsupseteq_1 u_1 \sqsupseteq_1 u_2$. Thus the $u_1$ of $s_{l+1} = s_l^{\oplus}$, which is the $u_1$ of $s_l$ can indeed be produced by performing $pop_1$ operations on the $v$ of $s_l$ ($s'_l$). The automaton is free to pick anything to be the $u_1$ of $s_{l+1}$ and so in particular can choose the correct value. The new $v$ of $s_{l+1}$ is restricted to be correct with respect to the $v$ of $s_l$ and similarly the restriction on $u_2$ is precisely what is required with respect to $u_1$ in $s_{l+1}$.

*Case when the condition stack of $s'_l$ (or by induction hypothesis equivalently $s_l$) is below both the guaranteed and tentative stacks:* Again w.l.o.g. assume that the condition stack of $s'_l$ is a $u_1$ stack (the case when it is a $v_1$ stack is similar). Due to the position of $u_1$ we must have in $s_l$: $u_1 \sqsupseteq_1 v$ and $u_1 \sqsupseteq_1 u_2$. Since the new $v_1$ in $s_{l+1}$ should be equal to $v$ in $s_l$ and the new $u$ in $s_{l+1}$ an initial prefix of $u_2$ in $s_l$, it must be possible to form both of these from performing $pop_1$ operations on the $u_1$ in $s_l$. The automaton is unconstrained in picking the $v_1$, so in particular it is able to guess the correct position—again the constraint on picking the new $v_2$ relative to $v_1$ is precisely the correct one. Note also that the constraint on picking the new $u$ relative to the old $u_1$ is precisely the correct one—popping to the next $j_u$ will yield the old $u_2$ and so popping from that to the second $j_u$ will yield the correct new $u$.

Either way since the sequence $s_1, s_2, \ldots, s_l, s_{l+1}$ by assumption exhibits equality of the top elements of the $u_2$ and $v$ or $v_2$ and $u$ in each element in the chain, it will carry out the above without aborting. Furthermore the second condition on the $s'_i$ is satisfied since the $s_i$ form a $^{\oplus}$-successor chain.

This establishes the induction hypothesis is true for all $l \leq k$. In particular when $l = k$ the top two 1-stacks of $s_k$ (or $s'_k$ ignoring the token on top) will be empty and so the automaton will halt in control-state $\boldsymbol{guess_{3_2}}$ as required.

Now let us consider the $\Leftarrow$) direction. We argue by induction on the converse hypothesis to what we had before. Suppose that $\mathcal{A}^P_{3^{alt}_2}$ generates a stack $[s'_1\ s'_2\ \cdots\ s'_l]$ (in the phase following the construction of the 2-stack $s$) that corresponds to a correct initial segment of a verification chain $s_1, s_2, \ldots, s_l$. Suppose now that $\mathcal{A}^P_{3^{alt}_2}$ proceeds to generate $[s'_1\ s'_2\ \cdots\ s'_l\ s'_{l+1}]$ that satisfies the conditions in the converse of the induction hypothesis used previously. This stack must have been produced from $[s'_1\ s'_2\ \cdots\ s'_l]$ by beginning with a $push_3; pop_2; pop_2$. Again we should consider the same two cases as before, noting that the additional assumed constraint on $s'_{i+1}$ relative to $s'_i$ ensures that the guessed new $u_1/v_1$ for $s'_{l+1}$ must indeed be the $u_1/v_1$ for $s^{\oplus}_l$ (*i.e.* the $u_2/v_2$ of $s'_l$ or equivalently $s_l$). The new $u_2/v_2$ and $v/u$ will be correctly created by the automaton, as discussed before when arguing in the $\Rightarrow$ direction.

The fact that the automaton does not fail means that it must have successfully found that the $u/u_2$ and $v_2/v$ stacks in $s_{l+1}$ share $top_1$ elements. Thus $s_{l+1} = s^{\oplus}_l$ as required and also satisfy the $top_1$ equality requirement.

The automaton will only halt in control-state $\boldsymbol{guess_{3_2}}$ if it detects two empty 1-stacks (modulo the token), constituting the final $s_k$ in the chain. ◀

### A.5.2 Exploiting $\mathcal{A}^P_{3^{alt}_2}$ in a $4$-CPDA

Since only one comparison needs to be made between adjacent elements, the problem illustrated in Figure 2 is no longer an issue. The same idea that took us from $\mathcal{A}^P_{3_2}$ to $\mathcal{A}^P_{5_2}$ can thus be used to go from $\mathcal{A}^P_{3^{alt}_2}$ to a $4_2$-CPDA $\mathcal{A}^P_{4_2}$.

▶ **Definition 44.** Let $P$ be an instance of Post's Correspondence Problem. The $4_2$-CPDA $\mathcal{A}^P_{4_2}$ shares the same stack-alphabet as $\mathcal{A}^P_{3^{alt}_2}$. It begins by behaving as $\mathcal{A}^P_{3^{alt}_2}$ until this automaton halts in control-state $\boldsymbol{guess_{3_2}}$. It then performs a $push_4$ operation and non-deterministically decides whether to operate in '$A$-mode' or '$B$-mode'. If it decides to operate in $A$-mode:

- Perform $collapse$ on the conditional stack (either $u_2$ or $v_2$ depending on which it has) of the $top_3$ element of the verification chain.
- Perform $push_4; pop_3$—this reveals the previous member of the verification chain as the $top_3$ stack.

- Repeat until $collapse$ has been performed on the second member of the verification chain (we do not do this to the first member). Once this stage is reached, enter distinguished control state $\boldsymbol{A}$.

If it decides to operate in $B$-mode, it proceeds as follows:
- First examines the token on top of the $top_3$ stack to determine whether the condition stack of the $top_3$ element in the chain is $u_1$ or $v_1$. If it is $u_1$ set $w := u$ and if it is $v_1$ set $w := v$.
- Performs $pop_3; push_3$ so that a copy of the previous element in the chain is now the $top_3$ stack. The automaton then performs $collapse$ on either the guaranteed stack $w$ or the tentative stack $w_2$ depending on which this previous element in the chain posesses.
- Perform $push_4; pop_3$.
- Repeat until $collapse$ has been performed on copies of all but the last members of the verification chain (including the first represented by the bottom 3-stack). Once done enter distinguished control-state $\boldsymbol{B}$.

We add an additional transition labelled $\boldsymbol{toCandidate}$ from both $\boldsymbol{A}$ and $\boldsymbol{B}$ to a distinguished control-state $\boldsymbol{candidate}$.

### Lemma 26

There exists a $\Sigma_1$-sentence $\phi$ such that for every instance $P$ of Post's Correspondence Problem we have $\mathcal{G}(\mathcal{A}^P_{4_2}) \vDash \phi$ iff $P$ has a solution.

**Proof.** Combine Lemmas 42 and 43. Thus $P$ has a solution if and only if $\mathcal{A}^P_{3_2^{alt}}$ can reach a configuration $(\boldsymbol{guess_{3_2}}, [s_1, s_2, \ldots, s_k])$ such that: For every $1 \leq i < k$:
- If $s_{i+1}$ contains a $u_1$ then this is equal to $u_2$ or $u$ in $s_i$ (depending on which one $s_i$ contains)
- or if $s_{i+1}$ contains a $v_1$ then this is equal to $v_2$ or $v$ in $s_i$ (depending on which one $s_i$ contains).

We claim that this is the case iff $\mathcal{A}^P_{4_2}$ can both reach a configuration $(\boldsymbol{A}, t)$ and a configuration $(\boldsymbol{B}, t)$ for some stack $t$.

Suppose first that such a pair of configurations is indeed reachable for $\mathcal{A}^P_{4_2}$. Since a stack produced by either an $A$-mode run or a $B$-mode run from a $\mathcal{A}^P_{3_2^{alt}}$ stack $[s_1, s_2, \ldots, s_k]$ will have this 3-stack as its bottom most 3-stack we may conclude that the configuration $(\boldsymbol{A}, t)$ as well as the configuration $(\boldsymbol{B}, t)$ must be produced beginning with the same $\mathcal{A}^P_{3_2^{alt}}$ stack. Since a $collapse$ on two elements in copies of some 2-stack $s_i$ will yield the same result iff they are the same, the construction of the $A$ and $B$ modes ensures that the equalities required to relate each $s_i$ to $s_{i+1}$ (for $1 \leq i < k$) must hold. After all, the $i$th $collapse$ performed in $B$-mode will be on the appropriate component of $s_i$ for $1 \leq i < k$ whilst the $i$th $collapse$ performed in $A$-mode will be on the correspondingly appropriate component of $s_{i+1}$. The results of these $collapse$s are directly compared since $collapse$ is performed on a copy of the relevant 2-stack that has precisely the same set of 2-stacks below it in each case.

It follows from the above that $P$ does indeed have a solution.

Conversely begin by assuming that $P$ has a solution. It follows that $\mathcal{A}^P_{3_2^{alt}}$ must be able to reach a configuration $(\boldsymbol{guess_{3_2}}, [s_1\ s_2\ \cdots\ s_k])$ satisfying the conditions above. By the converse considerations to before (in terms of comparing $collapse$s) these conditions must ensure that the $A$-mode and the $B$-mode both generate the same stack from this starting point, as required.

We can therefore take as the required $\Sigma_1$-sentence the following:

$$\exists x.\exists y.\exists z.(\boldsymbol{A}(x) \wedge \boldsymbol{B}(y) \wedge \boldsymbol{toCandidate}(x, z) \wedge \boldsymbol{toCandidate}(y, z))$$

◀

# B Decidability on $n_n$-CPDA

## $\mu$CPDA

### B.0.3 The Extended Model

An $n_S$-$\mu$CPDA is a device that has an $n_S$-CPDA at its disposal but may intervene and manipulate it beyond its normal course depending on whether the original $n_S$-CPDA satisfies various $\mu$-calculus sentences in its current configuration.

▶ **Definition 45.** An $n_S$-$\mu$CPDA $\mathcal{B}$ is a tuple:

$$\left\langle \Sigma, \Pi, Q, q_0, \Gamma, R'_{a_1}, R'_{a_2}, \ldots, R'_{a_r}, P'_{b_1}, P'_{b_2}, \ldots, P'_{b_{r'}}, R_{a_1}, R_{a_2}, \ldots, R_{a_r}, P_{b_1}, P_{b_2}, \ldots, P_{b_{r'}} \right\rangle$$

where $\left\langle \Sigma, \Pi, Q, q_0, \Gamma, R'{a_1}, R'{a_2}, \ldots, R'{a_r}, P'{b_1}, P'_{b_2}, \ldots, P'_{b_{r'}} \right\rangle$ is an $n_S$-CPDA called *the underlying $n_S$-CPDA*; $R'{a_i} \subseteq L^0_\mu \times \Theta_{n_S} \times Q$ and $P'{b_j} \in L^0\mu$ for each $1 \leq i \leq r$ and $1 \leq j \leq r'$.

As with CPDA configurations are elements of $Q \times stack^{\mathcal{C}}_{n_S}(\Gamma)$ but the only transitions allowed are specified by the $R'_{b_i}$ with reference to the $R_{b_j}$ rather than by the $R_{b_j}$ themselves. Likewise $P'_{b_i}$ are the only unary predicates it has.

▶ **Definition 46.** Let $\mathcal{B}$ be an $n_S$-$\mu$CPDA with underlying $n_S$-CPDA $\mathcal{A}$. Let $(q,s),(q',s')$ be configurations of $\mathcal{B}$ (and hence also of $\mathcal{A}$). There is an $a_i$ labelled transition from $(q,s)$ to $(q',s')$ in $\mathcal{B}$ just in case $\mathcal{G}(\mathcal{A}),(q,s) \vDash \phi$ where $(\phi,\theta,q') \in R'_{a_i}$ and $s' = \theta(s)$. Likewise we have $(q,s)$ satisfying the predicate $b_i$ just in case $\mathcal{G}(\mathcal{A}),(q,s) \vDash P'_{b_i}$.

Given these transition edges and predicates of $\mathcal{B}$ the graphs $\mathcal{G}(\mathcal{B})$ and $\mathcal{G}^\epsilon(\mathcal{B})$ are defined in the same way as with conventional CPDA.

▶ **Example 47.** Consider a standard order-1 pushdown automaton that has control-states $\{ q_0, q_1 \}$ and stack alphabet $\{ a, b \}$. Give it has a transition relation $R_c := \{ (q_0, pop_1, q_0) \}$ and predicates $P_a := \{ (q_0, a) \}$, $P_b := \{ (q_0, b) \}$.

Suppose that we extend this to a 1-$\mu$PDA with a sole $\mu$PDA transition relation $R'_c := \{ ((\mu X.(\boldsymbol{a} \vee [\boldsymbol{c}]X) \wedge \boldsymbol{b}), push^a_1, q_1) \}$. Then this $\mu$PDA will have a $c$-labelled transition from the configuration $(q_0, [bbaaabbbbbb])$ to the configuration $(q_1, [bbaaabbbbbba])$ but *no other* transitions from this configuration.

The $\mu$-calculus sentence asserts that the current configuration has $b$ on top of the stack but that repeated popping will yield $a$ on top.

### B.0.4 Strong Isomorphisms

Two graphs are said to be isomorphic if *qua* graphs they are essentially the same. As expected the formal definition is as follows:

▶ **Definition 48.** Let $\mathcal{G}$ and $\mathcal{G}'$ be graphs sharing a signature $\mathfrak{S}$ with respective node sets $N$ and $N'$. We say that $\mathcal{G}$ and $\mathcal{G}'$ are *isomporphic*, written $\mathcal{G} \cong \mathcal{G}'$ just in case there is a bijection $f : N \longrightarrow N'$ (called an isomorphism) such that for every $u \in N$ and unary predicate $\boldsymbol{b}$ of $\mathfrak{S}$ interpreted as $\boldsymbol{b}_{\mathcal{G}}$ in $\mathcal{G}$ and $\boldsymbol{b}_{\mathcal{G}'}$ in $\mathcal{G}'$ we have $u \in \boldsymbol{b}_{\mathcal{G}}$ iff $f(u) \in \boldsymbol{b}_{\mathcal{G}'}$ and for every edge $\boldsymbol{a}$ we have $u\boldsymbol{a}u'$ in $\mathcal{G}$ iff $f(u)\boldsymbol{a}f(u')$ in $\mathcal{G}'$.

It is well known that the theories in all logics we have introduced are invariant under isomorphism—a sentence will hold in a graph $\mathcal{G}$ just in case it holds in all isomporphic graphs $\mathcal{G}'$.

Note that every CPDA $\mathcal{A}$ can be viewed as a $\mu$CPDA $\mathcal{B}$. We simply take $\mathcal{B}$ to have underlying CPDA $\mathcal{A}$ and give it a predicate for every control-state/stack-alphabet pair in $Q \times \Gamma$ to facilitate a $\mu$-calculus sentence asserting that we are currently in a particular control-state with a particular symbol on top of the stack. This allows us to reconstruct the original transition relation of $\mathcal{A}$ in $\mathcal{B}$. It thus follows that for every CPDA $\mathcal{A}$ there exists a $\mu$CPDA $\mathcal{B}$ such that $\mathcal{G}^\epsilon(\mathcal{B}) \cong \mathcal{G}^\epsilon(\mathcal{A})$.

For CPDA there is a stronger notion of isomorphism where stack structure and control-states are preserved as well. This will be the form of isomporphism to which we usually appeal. In particular the definition makes sense when comparing $\mu$CPDA and CPDA.

▶ **Definition 49.** Let $\mathcal{A}$ and $\mathcal{A}'$ be $n_S$-$\mu$CPDA (and in particular either or both may be an $n_S$-CPDA). We say that $\mathcal{G}(\mathcal{A})$ and $\mathcal{G}(\mathcal{A}')$ (*resp.* $\mathcal{G}^\epsilon(\mathcal{A})$ and $\mathcal{G}^\epsilon(\mathcal{A}')$) are *strongly isomorphic* just in case there is an isomporphism $L$ between the graphs where for any configuration $(q, s)$ of $\mathcal{A}$ we can define $L$ by an expression of the form $L(q, s) := (L(q), L(s))$ where $L(pop_k(s)) = pop_k(L(s))$ for every $1 \leq k \leq n$; $L(collapse(s)) = collapse(L(s))$ and $L(\perp_n) = \perp_n$, overloading $L$ to additionally denote a map on both control-states and constructible stacks.

We write $\mathcal{G}(\mathcal{A}) \mathrel{\widehat{\cong}} \mathcal{G}(\mathcal{A}')$ (*resp.* $\mathcal{G}^\epsilon(\mathcal{A}) \mathrel{\widehat{\cong}} \mathcal{G}^\epsilon(\mathcal{A}')$) to indicate this.

Note that whilst $L$ is a bijection between the domains of each graphs, in the $\epsilon$-transition there may be intermediate control-states accessed during the course of $\epsilon$-transitions that do not appear in nodes of the $\epsilon$-closure of the graph. Therefore $L$ may not be a bijection between control-states. Nevertheless, for control states belonging to the $\epsilon$-closure (on which $L$ *is* a bijection) we adopt the convention '$L(q) := q$'. The correponding convention for stacks '$L(s) := s$' is not used as it would be highly misleading; two occurrences of a symbol $a$ in $s$ may map to different symbols in $L(s)$.

### B.0.5 Representing as Conventional CPDA

Just as every $n_S$-CPDA can be viewed as an $n_S$-$\mu$CPDA it turns out that the converse holds as well. Indeed we can view the main result of [2] as saying precisely this.

▶ **Theorem 50.** *Given any $n_S$-$\mu$CPDA $\mathcal{B}$ there exists an $n_S$-CPDA $\mathcal{A}$ such that $\mathcal{G}(\mathcal{B}) \mathrel{\widehat{\cong}} \mathcal{G}(\mathcal{A})$ and so also $\mathcal{G}^\epsilon(\mathcal{B}) \mathrel{\widehat{\cong}} \mathcal{G}^\epsilon(\mathcal{A})$.*

**Proof.** Let $\mathcal{A}^-$ be the $n_S$-CPDA underlying $\mathcal{B}$ with control-states $Q_{\mathcal{A}^-}$. Extend $Q_{\mathcal{A}^-}$ with a fresh distinguished control-state $\star$. Add fresh distinguished edges $\hat{q}$ for every $q \in Q_{\mathcal{A}^-}$ from the configuration $(\star, s)$ to the configuration $(q, s)$ for every stack $s$ and have a transition $\boldsymbol{\theta}$ for every $n_S$-stack operation connecting $(\star, s)$ to $(\star, \theta(s))$. Making $\star$ the initial state call the resulting automaton $\mathcal{A}^\star$.

Now let $\phi_1, \phi_2, \ldots, \phi_m$ be a list of all of the $\mu$-calculus sentences occurring in transition relations of $\mathcal{B}$. Let $\phi_i^q$ be the $\mu$-calculus sentence $[\hat{q}]\phi_i$ for every $q \in Q_{\mathcal{A}^-}$ and $1 \leq i \leq m$. Logical reflection for CPDA, as established in [2], allows us to construct an automaton $\mathcal{A}^\star_{LR}$ such that there exists an isomorphism $f : \mathcal{G}^\epsilon(\mathcal{A}^\star) \cong \mathcal{G}^\epsilon(\mathcal{A}^\star_{LR})$ and additionally there is a set $S_{[\hat{q}]\phi_i} \subseteq Q_{\mathcal{A}^\star_{LR}} \times \Gamma_{\mathcal{A}^\star_{LR}}$ such that a configuration $(p, t)$ of $\mathcal{A}^\star_{LR}$ satisfies $\mathcal{G}^\epsilon(\mathcal{A}^\star_{LR}), (p, t) \vDash [\hat{q}]\phi_i$ just in case $(p, top_1(t)) \in S_{[\hat{q}]\phi_i}$. That is $\mathcal{A}^\star_{LR}$ generates the same $\epsilon$-closure as $\mathcal{A}^\star$ but is also 'aware' of what $\mu$-calculus properties are satisfied at each configuration.

Note that we do not quite have a strong isomorphism here. Whilst [2] tells us the stacks either side of the isomorphism satisfy the required structural similarity, the control-state in the image of the isomorphism depends on the stack in the input configuration as well as the control-state. That said, the converse does hold: the control-state in the image determines the control-state in the input of the isomorphism. In particular it is well-defined to delete all control-states from $\mathcal{A}^\star_{LR}$ that are not associated with $\star$ in $\mathcal{A}^\star$. We also remove all edges other than those of the form $\boldsymbol{\theta}$ for $\theta \in \Theta_n$. Call the resulting automaton $\mathcal{A}^{\star\star}_{LR}$.

Now we construct the $n$-CPDA $\mathcal{B}$ to have the same control-states $Q_{\mathcal{A}}$ as $\mathcal{A}$ and the stack-alphabet of $\mathcal{A}^{\star\star}_{LR}$. In order to simulate $\mathcal{A}$ when in control-state $q$ it may do the following:

- Pick a $\mathcal{A}$-transition dependent on $\mu$-calculus sentence $\phi$ that performs stack-operation $\theta$ whilst moving to control-state $q'$.
- Check that the top element of the stack belongs to $S_{[\hat{q}]\phi}$ in which case we are indeed in a configuration corresponding to a $\mathcal{A}$-configuration in control-state $q$ at which $\phi$ holds.
- Transition into control-state $q'$ whilst performing the stack-operation dictated by the $\mathcal{A}^{\star\star}_{LR}$-transition $\boldsymbol{\theta}$.

Then $g \ : \ \mathcal{G}(\mathcal{B}) \widehat{\cong} \mathcal{G}(\mathcal{A})$ with $g(q, s) := g(q, t)$ where $f(\star, s) = (\_, t)$.

◀

## B.1 Monotonic CPDA

*We will without loss of generality make the assumption that $push_n$, $pop_n$ and $collapse$ on $n$-links is* only *performed during $\epsilon$-transitions.* This avoids the need for case separation when monotising automata—we can just focus on $\epsilon$-edges. Generality is not lost since $\epsilon$-closure allows us to decompose an $a$-labelled $push_n$ edge (for example) into an $\epsilon$-transition with $push_n$ followed by an $a$-edge with $nop$.

We will also use $\Sigma$ to denote the set of *non-*$\epsilon$ transition labels and view $\epsilon$ as lying outside of $\Sigma$.

### B.1.0.1 Lemma 29

Let $\mathcal{A}$ be an $n$-CPDA with edge alphabet $\Sigma$ and unary predicates $\Pi$. Then there exists an $n$-CPDA $\mathcal{A}^{\uparrow}$ such that $\mathcal{G}^\epsilon(\mathcal{A}) \widehat{\cong} \mathcal{G}^\epsilon(\mathcal{A}^{\uparrow} \upharpoonright_{\Sigma,\Pi})$ but whose additional distinguished edge labels include $r_\epsilon \notin \Sigma$ such that $\mathcal{A}^{\uparrow}$ is monotonic via $r_\epsilon$ and $(q, s)\boldsymbol{r}^{\uparrow}_{\epsilon^* \ a}(q', s')$ just in case $(q, s)\boldsymbol{r}_{r_\epsilon^* \ a}(q', s')$.

**Proof.** Due to Theorem 50 it is sufficient to define an order-$n$ $\mu$CPDA $\mathcal{A}^{\uparrow^\mu}$ that satisfies the requirements. Conversely it is easy to construct an order-$n$ $\mu$CPDA $\mathcal{A}^\mu$ that shares the same configuration graph as $\mathcal{A}$ since the current control-state and top stack symbol can trivially be detected with a $\mu$-calculus sentence. Extend this to a $\mu$CPDA $\mathcal{A}^{\uparrow^\mu}$ as follows:

- Add a unary predicate $\boldsymbol{q}$ for each control-state $q$ of $\mathcal{A}$.
- Add a marker **marker**$[\gamma]$ to the stack-alphabet for each $\gamma$ in the stack alphabet of $\Gamma$. The automaton ensures that at most one of these is on the stack at any one time. Extend the $\Sigma$ transitions to treat **marker**$[\gamma]$ as $\gamma$ and add a single unary predicate **marker** asserting that the marker is on top of the stack.
- We add edges labelled **deployMarker** that simply rewrites the top element of the stack $\gamma$ to **marker**$[\gamma]$ without changing the control-state.
- Add edges labelled **removeMarker** that rewrite a **marker**$[\gamma]$ on top of the stack to $\gamma$ without altering the control-state.
- Add edges $\boldsymbol{\epsilon_{<n}}$ for each $\epsilon$-transition in $\mathcal{A}$ *not* performing a $collapse$ on an $n$-link; a $pop_n$ nor a $push_n$
- Add edges $\boldsymbol{\epsilon_{push_n}}$ for each $\epsilon$-transition in $\mathcal{A}$ that performs a $push_n$ operation.

Now let $\phi_q$ be the $\mu$-calculus assertion: 'We can perform **deployMarker** and then perform arbitrary $\epsilon$ transitions, beginning with a $push_n$ and immediately removing the marker from the copy, ending up back with the stack at which we started, and indeed stopping precisely when we end up back where we started, with the marker on top and in control-state $q$.'

This can be expressed in the $\mu$-calculus by the following:

$$\phi_q := < \textsf{deployMarker} >< \epsilon_{push_n} >< \textsf{removeMarker} > \mu X.((\boldsymbol{q} \wedge \textsf{marker}) \vee (< \boldsymbol{\epsilon} > X \wedge \neg\textsf{marker}))$$

We define an $r_\epsilon$-edge to occur whenever we have an $\boldsymbol{\epsilon_{<n}}$-edge or an $\boldsymbol{\epsilon_{push_n}}$-edge. We additionally add an $r_\epsilon$-edge to control-state $q'$ via $nop$ whenever $\phi_{q'}$ holds in the current configuration (possible since it is a $\mu$CPDA).

Observe that reachability via $r_\epsilon$-edges preserves the original stack alphabet—markers are only implicitly deployed in the definition of each $\phi_q$, they are never introduced by an actual transition of $\mathcal{A}^{\uparrow^\mu}$.

Now we argue for correctness. We disregard the single $a \in \Sigma$-transition at the end of the path since by our w.l.o.g. assumption this is an order $(n-1)$-operation and so not pertinent to the definition of a climb.

First suppose that $(q, s)\boldsymbol{r}^{\uparrow}_{\epsilon^*}(q', s')$ (derived from $\mathcal{A}$). All operations featuring in this path other than a $pop_n$ or a $collapse$ on an $n$-link can be replaced directly by an $r_\epsilon$-edge. So we just need to show that $pop_n$ and $collapse$ on $n$-links can also be replaced. The fact that we are considering a climb rather than an arbitrary run tells us that for every stack $t$ occurring in the run witnessing $(q, s)\boldsymbol{r}^{\uparrow}_{\epsilon^*}(q', s')$ we must have $pop_n(s) \sqsubset_n t$. It must thus be that for every instance of $collapse$ on an $n$-link or $pop_n$ resulting in a configuration $(p', t)$ there must be an earlier configuration of the form $(p, t)$ in the run that is followed by a $push_n$ operation. But then $\phi_{p'}$ holds at this configuration and so there is an $r_\epsilon$-transition from $(p, t)$ to $(p', t)$, as required.

Conversely suppose that there is an $r^*_\epsilon$ path from $(q, s)$ to $(q', s')$. Argue by induction on the length of the path. If $pop_n(s) \sqsubset_n t$, then $t' = push_n(t)$ and $t' = \theta(t)$ for any $\theta \in \Theta_{n-1}$ must satisfy $pop_n(s) \sqsubset_n t'$. Moreover these operations for $r_\epsilon$-edges are directly inherited from the original $\epsilon$-edges and so the path is as required for these operations. It just remains to consider $r_\epsilon$ resulting from a $\phi_{p'}$-test at a configuration $(p, t)$ with $s \sqsubset_n t$, resulting in $(p', t)$. But $\phi_{p'}$ asserts the existence of precisely such an $\epsilon$-path. ◀

▶ **Remark.** Since the initial configuration has the empty stack $\perp_n$ and we can w.l.o.g. view '$pop_n(\perp_n)$' $\sqsubset_n t$ as holding for any stack $t$ (for example by treating the initial stack as $push_n(\perp_n)$ and ensuring the automaton never pops down below this) we get that all reachable configurations are *monotonically reachable* from the initial configuration via $\{r_\epsilon \cup \Sigma\}$-labelled paths.

### Lemma 31

Let $\mathcal{A}$ be an $n$-CPDA with unary predicates $\Pi$. Then there exists an $n$-CPDA $\mathcal{A}^{\downarrow}$ with stack-alphabet $\Gamma^{\downarrow}$ and control-state space $Q^{\downarrow}$ such that $\mathcal{G}(\mathcal{A}) \mathrel{\widehat{\cong}} \mathcal{G}(\mathcal{A} \upharpoonright_{\Pi^{\downarrow}})$ and that also has a predicate $P^{\downarrow}$ for each $P \in \Pi$ such that $P$ holds of precisely those configurations $c$ from which $\mathcal{A}$ has an $\epsilon$-fall to a configuration $c'$ satisfying $P$.

**Proof.** As with the proof of Lemma 29 we work with $n$-$\mu$CPDA instead of $n$-CPDA, as permitted by Theorem 50. In fact we begin the construction of $\mathcal{A}^{\downarrow^\mu}$ (the $\mu$CPDA meeting the requirements that can then be translated to the CPDA $\mathcal{A}^{\downarrow}$) in exactly the same way as $\mathcal{A}^{\uparrow^\mu}$ from Lemma 29. We further add an edge $\boldsymbol{q}$ for each $q \in Q$ that transitions to control-state $q$ without altering the stack; a $\boldsymbol{pop_n}$ edge performing a $pop_n$ operation without altering the control-state; and a $\textsf{destroy}_{\boldsymbol{n}}$ edge for every $\epsilon$-edge performing a $collapse$ on an $n$-link or a $pop_n$ operation, making the same transition as the $\epsilon$-edge.

We can define the property $\textbf{marker}^{\downarrow}$ asserting that a marker has already been deployed to the top of some $(n-1)$-stack below in $L_\mu$:

$$\textbf{marker}^{\downarrow} \;:=\; \mu X.(\textbf{marker} \vee \; < \boldsymbol{pop_n} > X)$$

For each predicate $P \in \Pi$ the following $\mu$-calculus $\psi_{P^{\downarrow}}$ sentence defines the required predicate $P^{\downarrow}$. It asserts reachability whilst making sure the final result is an $\epsilon$-fall by deploying the marker every time we make a $push_n$ operation, thereby enforcing that we should eventually descend below the marker:

$$\begin{aligned}\psi_{P^{\downarrow}} \quad := \quad & \mu X.(((P \;\wedge\; \neg\textbf{marker}^{\downarrow}) \\ & \vee \; < \textbf{destroy}_{\boldsymbol{n}} \cup \; < \boldsymbol{\epsilon_{<n}} > X \\ & \vee (\neg\textbf{marker}^{\downarrow} \wedge \; < \textbf{deployMarker} >< \boldsymbol{\epsilon_{push_n}} > X) \\ & \vee (\textbf{marker}^{\downarrow} \wedge \; < \boldsymbol{\epsilon_{push_n}} > X)))\end{aligned}$$

Note that some stacks during the run asserted to exist by $\psi_{P^{\downarrow}}$ may contain multiple markers at one time (unlike with $\mathcal{A}^{\uparrow}$). With $\mathcal{A}^{\uparrow}$ we were concerned about knowing when we return to exactly the same stack, whereas here we just want to make sure that we do not stop until we have returned to the last stack at which we performed a $push_n$ (in order to get an $\epsilon$-fall). ◀

### Lemma 33

Let $(q, s)$ and $(q', s')$ be two configurations of a CPDA $\mathcal{A}$ and let $(q, L(s))$ and $(q', L(s'))$ be the corresponding configurations in $\mathcal{A}^{\uparrow\downarrow}$ via the strong isomorphism. Then $(q, s)\boldsymbol{r}_{\epsilon^* a}(q', s')$ just in case $(q, L(s))\boldsymbol{b}_a L(q', L(s'))$.

**Proof.** Suppose first that $(q, L(s))\boldsymbol{b}_a(q', L(s'))$. Then by definition there must be an $\epsilon$-fall from $(q, L(s))$ to some configuration $(q'', L(s''))$ in $\mathcal{A}^{\uparrow\downarrow}$ such that $(q'', L(s''))\boldsymbol{r}^{\uparrow}_{r^*_\epsilon\ a}(q', L(s'))$. But the latter implies $(q'', L(s''))\boldsymbol{r}^{\uparrow}_{\epsilon^*\ a}(q', L(s'))$ and so there is an $\epsilon^*\ \ a$ path in $\mathcal{A}^{\uparrow\downarrow}$ from $(q, L(s))$ to $(q', L(s')$. But this uses edges inherited from the original $\mathcal{A}$ and so $(q, s)\boldsymbol{r}_{\epsilon^* a}(q', s')$ in $\mathcal{A}$.

Conversely suppose that $(q, s)\boldsymbol{r}_{\epsilon^* a}(q', s')$ in $\mathcal{A}$. This must be witnessed by a run and we may take the right-most element $(q'', s'')$ in the run such that $pop_n(s'') \;\sqsubseteq_n\; pop_n(t)$ for stacks $t$ to the left of $s''$ in the run. This is the 'lowest point' the $n$-stack reaches in the run. By definition of $\epsilon$-fall we have an $\epsilon$-fall from $(q, s)$ to $(q'', s'')$. By Lemma 29 we must also have $(q'', L(s''))\boldsymbol{r}^{\uparrow}_{r^*_\epsilon\ a}(q', L(s'))$ in $\mathcal{A}^{\uparrow\downarrow}$since $pop_n(s'') \;\sqsubset_n\; pop_n(s')$ (due to it being the lowest point in the run). We thus have the required bounce. ◀

## B.2 Link Trails: Towards Link Elimination for $n_n$-CPDA

### Lemma 36

Consider two *constructible* $n$-stacks $s$ and $s'$ with $\textbf{stripln}(s) = \textbf{stripln}(s')$. Assume for every stack or atomic element $a$ contained within $s$ and corresponding element $a'$ in $s'$ we have $\textbf{col}(a) = \textbf{col}(a')$. Then $s = s'$.

**Proof.** Suppose for contradiction that despite the conditions holding we have $s \neq s'$. Since $\textbf{stripln}(s) = \textbf{stripln}(s')$ the difference must be entirely down to a discrepancy in $n$-links. Since colouring determines which atomic elements are the source of a link (those not coloured $\perp$) it must, more specifically, be down to a discrepancy in the target of a link eminating from a particular atomic element. Let $a$ be the lowest atomic element in $s$ (with corresponding element $a'$ in $s'$) such that the

$n$-link from $a$ and the $n$-link from $a'$ have different targets. By assumption of $a$ being the lowest such element we must have $s_{<a} = s'_{<a'}$.

Suppose for contradiction that neither $a$ nor $a'$ were produced by a $push_1^{a,n}$ or $push_1^{a',n}$ operation. Since $s$ and $s'$ are both constructible, there must be a sequence of stack operations constructing $s$ and $s'$. Let $i$ be the order of the last $push_i$ operation that occurs in this sequence for $s$ that creates the position $a$ and after which the position $a$ is never discarded. Let $i'$ be similar for $s'$ and $a'$. So $i \geq 2$ and $i' \geq 2$. Note that since $s_{<a} = s'_{<a'}$ it must in particular be the case $top_i(pop_i(s_{<a})) = top_i(pop_i(s'_{<a'}))$ and indeed that $top_{i'}(pop_{i'}(s_{<a})) = top_{i'}(pop_{i'}(s'_{<a'}))$. Thus the $push_i$ and $push_{i'}$ operations must both create an $a$ and $a'$ that is a copy of an element with the same link. This is a contradiction since we are assuming $a$ and $a'$ have different targets.

So w.l.o.g. assume that $a$ is produced by a $push_i$ operation whilst $a'$ is produced by a $push_1^{a',n}$ operation. Let $t_j$ be the $j$-stack containing position $a$ (in $s$) and $t'_j$ be the $j$-stack containing $a'$ (in $s'$) for $1 \leq j \leq i-1$. By assumption we have $\mathbf{col}(t_j) = \mathbf{col}(t'_j)$ for each $j$. It will be helpful to further define $t_0 := a$ and $t'_0 = a'$.

We now argue by induction that for all $0 \leq j < n-1$ it is the case that $\mathbf{col}(t_j) = \mathbf{col}(t'_j) = c_>$ and that there is no freshly created $n$-link (*i.e.* $n$-link from an element $b$ such that $\boldsymbol{l_r}(b) = 1$) in $t_{j+1}$.

For the base case take $j = 0$. It must be the case that $\mathbf{col}(a) = \mathbf{col}(a') = c_>$ since the only other option is $c_=$ which is impossible since $\boldsymbol{l_a}(a) \neq \boldsymbol{l_a}(a')$. Since any $n$-link above $a'$ in $t'_1$ must share the target of $a'$ (as it would be another freshly created link in the same $n$-stack) it follows that these would also have colour $c_=$. This means that there can be no freshly created $n$-links above $a$ in $t_1$ since these would have colour $c_>$ which would not match the corresponding colour in $t'_1$. There can be no freshly created $n$-links below $a$ in $t_1$ as $a$ is not freshly created and so any stack containing such an $n$-link below $a$ would not be constructible.

For the induction step consider $j > 0$. We have $\mathbf{col}(t_j) = \mathbf{col}(t'_j)$ by assumption. We also assume as part of the induction hypothesis that $t_j$ contains no freshly created $n$-link. Suppose that there is a freshly created $n$-link in a $j$-stack below $t_j$ in $t_{j+1}$. Then there must be a corresponding freshly created $n$-link in $t'_{j+1}$ below $t'_j$ (from the assumption that $a$ and $a'$ form the lowest $n$-link discrepancy in $s$ and $s'$). It follows that $\mathbf{col}(t'_j) = c_=$ (since $t'_j$ also contains a fresh $n$-link) and so by equality of colouring $\mathbf{col}(t_j) = c_=$. But due to the fact that $t_j$ contains no freshly-created $n$-link we would also have $\mathbf{col}(t_j) = c_<$, a contradiction. Thus there is no freshly created $n$-link below $t_j$ in $t_{j+1}$. If there is a freshly created $n$-link in $t_{j+1}$ above $t_j$ then there must be a lowest $j$-stack $u_j$ containing such a link. But then $\mathbf{col}(u_j) = c_>$ since there are no freshly created $n$-links below it in $t_{j+1}$. So the corresponding $j$-stack $u'_j$ in $t'_j$ must also have $\mathbf{col}(u'_j) = c_>$. But this is impossible since there is a freshly created $n$-link in $t'_j$ and so $\mathbf{col}(u_j)' \in \{ c_<, c_= \}$. It follows that there is no freshly created $n$-link in $t_{j+1}$.

Now observe that $\mathbf{col}(t'_j) = c_>$ since from the paragraph above no fresh $n$-links can occur below it in $t'_{j+1}$. Thus we also have $\mathbf{col}(t_j) = c_>$. We have thus established the induction hypothesis.

Recall that the position $a$ was produced by a $push_i$ operation for $2 \leq i \leq n$. Suppose first that $2 \leq i < n$. The result above tells us that $\mathbf{col}(t_{i-1}) = c_>$ but also that $t_{i-1}$ contains no fresh $n$-links. This is a contradiction since the only way an $(i-1)$-stack derived from a copy of the $(i-1)$-stack below it could contain a link with a higher target is if it creates a fresh link.

Now suppose that $i = n$. We know from the induction hypothesis that $t_{n-1}$ contains no fresh $n$-link. Thus we have $\mathbf{col}(t_{n-1}) \in \{ c_=, c_< \}$. But since $t'_{n-1}$ does contain a fresh $n$-link we must have $\mathbf{col}(t'_{n-1}) = c_>$. Thus $\mathbf{col}(t_{n-1}) \neq \mathbf{col}(t'_{n-1})$, which is the required contradiction. ◀

The next step is to show how a CPDA can dynamically assign colours to its stacks correctly. We restrict ourselves to automata that *only* have $n$-links so that the only way to destroy internal stacks is using a higher-order *pop* operation. Let $s$ be an $n_n$-CPD stack over the alphabet $\Gamma$. We define the *colour tracking* stack $\mathsf{colTr}(s)$ to be the stack over the alphabet:

$$\Gamma \times \{ \perp, c_=, c_> \} \times \prod_{i=0}^{n-2} \{ c_<, c_=, c_> \}^{n-1-i} \quad \cup \quad [\, 1..(n-1) \,] \times \{ c_<, c_=, c_> \}$$

We construct $\mathsf{colTr}(s)$ from $s$ by first replacing each atomic element $a$ in $s$ with an element

$$(a, c_0, (b_1^0, \ldots, b_{n-1}^0), (b_2^1, \ldots, b_{n-1}^1), \ldots, (b_{n-1}^{n-2}))$$

where:

- $c_0 := \mathsf{col}(a)$
- $b_j^0 \in \{ c_<, c_= \}$ for $1 \leq j \leq n-1$ just in case $a$ sources an $n$-link and one of the following holds:
  - $\mathsf{col}(top_{j+1}(s_{\leq a})) = c_>$ but $\mathsf{col}(top_{j+1}(s_{<a})) \neq c_>$ in which case $b_j^0 = \mathsf{col}(top_{j+1}(s_{<a}))$.
  - There is another atomic element $a'$ anywhere below $a$ in $top_{j+1}(s_{\leq a})$ also sourcing an $n$-link such that $\boldsymbol{l}_a(a') = \boldsymbol{l}_a(a)$ and such that $\mathsf{col}(top_{j+1}(s_{\leq a})) = c_>$ but $\mathsf{col}(top_{j+1}(s_{<a})) = b_j^0 \neq c_>$.
- $b_j^0 = c_>$ otherwise.
- For each $1 \leq i \leq n-2$ let $s_i := top_{i+1}(s_{\leq a})$. Then for each $i < j \leq n-1$ we have $b_j^i \in \{ c_<, c_= \}$ just in case one of the following holds:
  - $\mathsf{col}(s_i) = c_>$ but $\mathsf{col}(top_{j+1}(s_{<s_i})) = b_j^i \neq c_>$
  - There is another $i$-stack $s_i'$ occurring anywhere below $s_i$ in $top_{j+1}(s_{<s_i})$ such that $\boldsymbol{l}_a(s_i') = \boldsymbol{l}_a(s_i)$ and $\mathsf{col}(s_i') = c_>$ but $\mathsf{col}(top_{j+1}(s_{<s_i'})) = b_j^i \neq c_>$.
- $b_i^j = c_>$ otherwise.

We finish the construction of $\mathsf{colTr}(s)$ by adding a decoration $(i, \mathsf{col}(s_i))$ on top of each $i$-stack $s_i$ in $s$ for $1 \leq i \leq n-1$.

The idea is that each component stack of $s$ is annotated with its colour and the additional decorations provide the necessary information to determine how a stack operation affects the colours.

▶ **Lemma 51.** *Let $s$ be an $n_n$-stack over an alphabet $\Gamma$. Suppose that for $0 \leq k < k' \leq n-1$ we have $\mathsf{col}(top_{i+1}(s)) = c_>$ for every $i$ with $k \leq i < k'$. Then there is no $k$-stack $s_k$ strictly below $top_{k+1}(s)$ in $top_{k'+1}(s)$ such that $\boldsymbol{l}_a(s_k) \geq \boldsymbol{l}_a(top_{k+1}(s))$.*

**Proof.** First we claim that $\boldsymbol{l}_a(top_{i+1}(s)) = \boldsymbol{l}_a(top_{k+1})$ for $k \leq i \leq k'$. To see this argue by induction on $i$ (for $k \leq i \leq k'$). The base case is trivial (since $i = k$). For the induction step note that since $\mathsf{col}(top_{i+1}(s)) = c_>$ it must be that $\boldsymbol{l}_a(top_{i+1}(s))$ is greater than $\boldsymbol{l}_a(s_i)$ for any $i$-stack $s_i$ below $top_{i+1}(s)$ in $top_{i+2}(s)$. It then follows by definition that $\boldsymbol{l}_a(top_{i+2}(s)) = \boldsymbol{l}_a(top_{i+1}(s)) = \boldsymbol{l}_a(top_{k+1}(s))$.

It follows that for any given $k \leq i < k'$ we cannot have a $k$-stack $s_k$ with $\boldsymbol{l}_a(s_k) \geq \boldsymbol{l}_a(top_{k+1}(s))$ below $top_{i+1}(s)$ in $top_{i+2}(s)$ since that would contradict the assumption that $top_{i+1}(s) = c_>$. This in turn implies that there is no $k$-stack $s_k$ strictly below $top_{k+1}(s)$ in $top_{k'+1}(s)$ such that $\boldsymbol{l}_a(s_k) \geq \boldsymbol{l}_a(top_{k+1}(s))$. ◀

Hence we can use the decorations according to the following lemma:

▶ **Lemma 52.** *Let $s$ be an $n_n$-stack over an alphabet $\Gamma$. Suppose that for $0 \leq k < k' \leq n-1$ we have $\mathsf{col}(top_{i+1}(s)) = c_>$ for every $k \leq i < k'$. Suppose further that $s'$ is an $n_n$-stack such that $pop_{k+1}(\mathsf{colTr}(s)) = pop_{k+1}(\mathsf{colTr}(s'))$ but where $top_{k+1}(s') \neq c_>$. Then where*

$$top_1(\mathsf{colTr}(s)) = (a, c_0, (b_1^0, \ldots, b_{n-1}^0), (b_2^1, \ldots, b_{n-1}^1), \ldots, (b_{n-1}^{n-2}))$$

*we have:*

$$\mathbf{col}(top_{k'+1}(s')) = \begin{cases} b^k_{k'} & \textit{if } \mathbf{col}(top_{k'+1}(s)) = c_> \\ c_< & \textit{otherwise} \end{cases}$$

**Proof.** We may appeal to Lemma 51 in order to get that there is no $k$-stack $s_k$ below $top_{k+1}(s)$ in $top_{k'+1}(s)$ such that $\boldsymbol{l}_a(s_k) = \boldsymbol{l}_a(top_{k+1}(s))$. Moreover we must have $\boldsymbol{l}_a(top_{k'+1}(s')) = \boldsymbol{l}_a(s_{<top_{k+1}(s)})$ due to the fact that $\mathbf{col}(top_{k'+1}(s')) \neq c_>$ but $s$ and $s'$ are equal everywhere below $top_{k'+1}(s')$. In the case when $\mathbf{col}(top_{k'+1}(s')) = c_>$ the definition of $b^k_{k'}$ thus implies that $\mathbf{col}(top_{k'+1}(s')) = b^k_{k'}$.

Consider now the only other case when $\mathbf{col}(top_{k'+1}(s)) \in \{ c_=, c_< \}$. Again by Lemma 52 we know that $\boldsymbol{l}_a(top_{k+1}(s))$ is *strictly* greater than $\boldsymbol{l}_a(s_k)$ for any $k$-stack below it in $top_{k'+1}(s)$. We also know that $\boldsymbol{l}_a(top_{k+1}(s')) < \boldsymbol{l}_a(top_{k+1}(s))$ due to colouring. Thus we can conclude that $\boldsymbol{l}_a(top_{k'+1}(s')) < \boldsymbol{l}_a(top_{k'+1}(s))$ meaning that $\mathbf{col}(top_{k'+1}(s')) = c_<$. ◀

It is also helpful to be able to use colour annotations to recover which stacks contain fresh links.

▶ **Lemma 53.** *Let $s$ be an $n_n$-stack. Then for each $2 \leq i \leq n$, the stack $top_i(s)$ contains an $n$-link from an atom $a$ with $\boldsymbol{l}_r(a) = 1$ iff $\mathbf{col}(top_n(s)) = c_>$ and additionally for each $j$ such that $i \leq j < n$ we have $\mathbf{col}(top_j(s)) \in \{ c_=, c_> \}$.*

**Proof.** First suppose that $top_i(s)$ contains an $n$-link from an atom $a$ with $\boldsymbol{l}_r(a) = 1$. Since no link in the $(n-1)$-stack below can source a link with the same target, we must have $\mathbf{col}(top_n(s)) = c_>$. Moreover, no link in the $top_n(s)$ stack can have a target above that of $a$. Since $top_i(s)$ contains $a$, $top_j(s)$ must contain $a$ for $i \leq j < n$ and so $\mathbf{col}(top_j(s)) \in \{ c_=, c_> \}$, as required.

Now suppose that the right-hand-side of the 'iff' holds. Since $\mathbf{col}(top_n(s)) = c_>$ the $top_n$ stack must contain a fresh $n$-link as all other $n$-links in the $top_n$ stack would have been created and so exist in an $n$-stack below it. Suppose for contradition that $top_i(s)$ does not contain a fresh $n$-link. Then there must be a maximum $j$ with $i < j \leq n-1$ such that $top_j(s)$ does not contain a fresh $n$-link. But since $top_n$ *does* contain a fresh $n$-link there must exist an $n$-link below $top_j(s)$ in $top_{j+1}(s)$ whence we would have $\mathbf{col}(top_j(s)) = c_<$, a contradiction. ◀

The following Lemma tells us that we can preserve the correct annotations whilst manipulating an $n_n$-stack. Unfortunately we do not have a version of this Lemma for $n$-stacks containing links of other orders.

▶ **Lemma 54.** *Let $s$ be an $n_n$-stack over an alphabet $\Gamma$ and let $\theta$ be an $n$-stack operation. There then exists a compound stack operation $\theta'$ such that $\mathbf{colTr}(\theta(s)) = \theta'(\mathbf{colTr}(s))$*—i.e. *$\theta'$ could be implemented by an $n_n$-CPDA. Moreover the number of operations in $\theta'$ is bounded.*

**Proof.** Consider each possible $\theta$ in turn.

If it is a $push_1$ operation, then if no link is attached, no colour is affected. We can thus just pop off the colour annotations on top of the stack (which are bounded in number), and push

$$(a, \perp, (b^0_1, \ldots, b^0_{n-1}), (b^1_2, \ldots, b^1_{n-1}), \ldots, (b^{n-2}_{n-1}))$$

on the stack with colour $\perp$ where $b^0_j := c_>$ for all $1 \leq j \leq n-1$ and for each $1 \leq i < j \leq n-1$ we set $b^i_j$ to be the $b^i_j$ from the previous $top_1$ element on the stack (since this new element has no affect on any colour).

If it is a $push^{a,n}_1$ operation, then again we first pop off the colour annotations on top of the stack. In the light of Lemma 53 these colour annotations allow us to deduce the set $F \subseteq [2..n]$ of elements $i$ such that $top_i(s)$ contains a fresh $n$-link. The colour of any $top_i$-stack with $i \in F$ is unchanged since they already contain a link with the highest possible target. The colour of $top_i$-stacks with $i \notin F$ (with $i \in [1..n]$) are set as follows (which do not necessarily but may result in a change of colour):

- if $i+1 \in F$ then a stack below $top_i(s)$ in $top_{i+1}(s)$ already contains a link and so the new colour of the new $top_i$ stack is $c_=$.
- otherwise this is the first fresh link and so the new $top_i$ stack has colour $c_>$.

Note that this ensures the colour $c_0$ of the new atomic element to be created is either $c_=$ or $c_>$ depending on whether there already exists a fresh link below it in the $top_2$ stack. The actual element being pushed onto the stack has the form:

$$(a, c_0, (b^0_1, \ldots, b^0_{n-1}), (b^1_2, \ldots, b^1_{n-1}), \ldots, (b^{n-2}_{n-1}))$$

For all $i \in F$ and all $j$ with $i < j \leq n$ we have $b^{i-1}_{j-1}$ set to the $b^{i-1}_{j-1}$ from the previous $top_1$ element (as discarding the $top_i$ $(i-1)$-stack is no different to before if it already contained a fresh link). For all $i, j \notin F$ and $j$ with $1 \leq i < j \leq n$ we set $b^{i-1}_{j-1}$ to be the colour of the $top_j$ $(j-1)$-stack prior to pushing the new element on the stack. This is correct since discarding $top_i$ for $i \in F$ will eliminate the newly added fresh $n$-link, which is the only fresh $n$-link in both the $top_j$ stack thereby returning the $top_j$ colour to that which it was before it was added. Note that it is impossible for $j \notin F$ but $i \in F$ since $i < j$ and so all cases are covered.

If it is a $push_k$ operation for $2 \leq k \leq n$, then we discard all colour annotations $(i, c_i)$ on top of the $top_{i+1}$ stack for $i \geq k$, perform a $push_k$ operation changing the colour annotation on top of the resulting top-most $k-1$ stack to $(k-1, c_=)$ and replacing all of the previously discarded decorations unchanged. The colour annotations on $i$-stacks for $i > k-1$ will remain correct as no new links are created and those on $i$-stacks for $i < k-1$ will remain correct as they depend only on what was copied in its entirety by the $push_k$ operation.

Observe how none of the $b^i_j$ values of atomic elements in the copied stack need changing. Overloading the notation $a$ consider an atom

$$a := (a, c_0, (b^0_1, \ldots, b^0_{n-1}), (b^1_2, \ldots, b^1_{n-1}), \ldots, (b^{n-2}_{n-1}))$$

occurring in the newly created $(k-1)$-stack. For each $i, j$ with $i < j < k-1$ note that $top_{j+2}(push_k(\mathbf{colTr}(s))_{\leq a})$ is a copy of a $(j+1)$-stack in the $(k-1)$-stack below. Since the meaning of $b^i_j$ is completely determined by the $(j+1)$-stack in which it resides, this remains correct. For $i < j = k-1$ we have it that $s_i$ is a copy of some stack $s'_i$ below it in the $top_{j+2}(push_k(\mathbf{colTr}(s))_{\leq a}) = top_{k+2}(push_k(\mathbf{colTr}(s))_{\leq a})$ $(j+1)$-stack (so in particular $\boldsymbol{l}_a(s_i) = \boldsymbol{l}_a(s'_i)$). But $s'_i$ would have existed prior to performing $push_k$ and must also contain an atom annotated with $b^i_j$ and so $b^i_j$ must be a correct annotation. For $j > k-1$ (and $i < j$ as usual) we have the case $i \leq k-1$, which implies that $top_{i+1}(push_k(\mathbf{colTr}(s))_{\leq a})$ is a copy of an $i$-stack below it in its $(j+1)$-stack, and also the case $i > k-1$ in which case the conditions remain unchanged to before the $push_k$ operation—in particular $push_k$

If it is a $pop_k$ operation for $k < n$, then first discard all of the decorations from the stack associated with the $top_i$ stack for $k \leq i \leq n$. If $k = 1$ and the top element of the stack is now a linkless element for some $a$, then we can just $pop_1$ it off without affecting the colour and restore the decorations unchanged. Otherwise we must be able to see the colour of the $top_k$ stack (even when $k = 1$). If the $top_k$ stack has colour $c_=$ or $c_<$ then discarding it will not change any other colours and so we simply perform $pop_k$ and restore the previously discarded decorations unchanged. If the $top_k$ stack has colour $c_>$, then again we perform $pop_k$ but we also need to recompute the colours for the $top_i$ stack where $k < i \leq n$ and amend the previously discarded colour decorations accordingly when restoring them. Let $k < l \leq n-1$ be the least $l > k$ such that either $l = n-1$ or else $\mathbf{col}(top_{l+1}(s)) \neq c_>$. We can make use of Lemma 52 to soundly set the colour of

$top_{j+1}(pop_k(\mathsf{colTr}(s)))$ to $b_j^{k-1}$ for $k < j < l$. If $top_{l+1} = c_>$ then Lemma 52 allows us to set its new colour to $b_l^{k-1}$, if $top_{l+1} \in \{ c_<, c_= \}$ then the same lemma tells us to set it to $c_<$.

There is no need to adjust the colour of stacks above $l$; either there are no stacks outside of $l$ that have a colour assignment (*i.e.* when $l = n-1$) or else for $j > l$ we have $\mathsf{col}(top_{j+1}(pop_k(\mathsf{colTr}(s)))) = \mathsf{col}(top_{j+1}(\mathsf{colTr}(s)))$ since $\mathsf{col}(top_{l+1}(\mathsf{colTr}(s))) \in \{ c_<, c_= \}$.

If it is a $pop_n$ operation or a $collapse$ operation (which must be on an $n$-link) we do not need to do anything special as all decorations will be accurate. For the $collapse$ we just need to $pop_1$ down to the element on which to collapse. ◀

**Lemma 37**

Let $\mathcal{A}$ be an $n_n$-CPDA. Then there exists an $n_n$-CPDA $\mathsf{lum}(\mathcal{A})$ such that $\mathcal{G}^\epsilon(\mathsf{lum}(\mathcal{A})) \cong \mathcal{G}^\epsilon(\mathcal{A})$ and further such that for any reachable configurations $(q, s), (q, s')$ of $\mathsf{lum}(\mathcal{A})$ we have $s = s'$ iff $\mathsf{stripln}(s) = \mathsf{stripln}(s')$.

**Proof.** We define $\mathsf{lum}(\mathcal{A})$ to be the $n_n$-CPDA that replaces all operations of $\mathcal{A}$ generating an $a$-edge with a compound operation from Lemma 54 where the compound generates a path of the form $\epsilon^* a$. Since $\mathsf{colTr}(s)$ for any stack $s$ includes an annotation of the colour of each constituent stack, Lemma 36 ensures that $s = s'$ iff $\mathsf{stripln}(s) = \mathsf{stripln}(s')$ for any stacks $s$ and $s'$ reachable by $\mathsf{lum}(\mathcal{A})$. The predicates for $\mathsf{lum}(\mathcal{A})$ are induced by those for $\mathcal{A}$ by projecting the stack alphabet and control-states of $\mathsf{lum}(\mathcal{A})$ onto those of $\mathcal{A}$. ◀

## B.3 Meta-Annotations

▶ **Definition 55.** Fix $k \in \mathbb{N}$ and let $\mathcal{A}$ be an $n_n$-CPDA with control-states $Q$ and edge-labels in $\Sigma$. A *$k$-meta-annotation* for $\mathcal{A}$ is a $|\Sigma|.k$-tuple $((Q_1^a)_{a\in\Sigma}, \ldots, (Q_k^a)_{a\in\Sigma})$ where each component is a subset of $Q$.

Given an $n_n$-CPDA $\mathcal{A}$ and $k \in \mathbb{N}$ the $n$-PDA $\mathsf{GrStripln}_{\boldsymbol{k}}(\mathcal{A})$ is formed using the following recipe:

- Take the $n_n$-CPDA $\mathsf{lum}(\mathcal{A})$ and modify it so that a single $k$-meta-annotation $((Q_1^a)_{a\in\Sigma}, \ldots, (Q_k^a)_{a\in\Sigma})$ is kept at the top of every $(n-1)$-stack. No restriction is placed on what this may be (it is non-deterministically chosen from amongst all $k$-meta-annotations). We add a predicate $\mathsf{Met}(((Q_1^a)_{a\in\Sigma}, \ldots, (Q_k^a)_{a\in\Sigma}))$ holding at all configurations with the corresponding meta-annotation on top together with a predicate $\mathsf{Met}(q, ((Q_1^a)_{a\in\Sigma}, \ldots, (Q_k^a)_{a\in\Sigma}))$ additionally asserting that the automaton is in control-state $q$. Unary predicates are inherited directly from $\mathcal{A}$ on the basis of control-state and stack symbol immediately below the meta-annotation. Call this $n_n$-CPDA $\mathsf{lum}(\mathcal{A})_k^+$.
- Let $\mathsf{GrStripln}_{\boldsymbol{k}}(\mathcal{A})^-$ be the automaton $\mathsf{lum}(\mathcal{A})_k^{+\uparrow\downarrow}$.
- Finally $\mathsf{GrStripln}_{\boldsymbol{k}}(\mathcal{A})$ is $\mathsf{GrStripln}_{\boldsymbol{k}}(\mathcal{A})^-$ restricted to edges that do not perform a $collapse$ or a $pop_n$ operation, and we remove all links. Thus $\mathsf{GrStripln}_{\boldsymbol{k}}(\mathcal{A})$ is an $n$-PDA. We further add an edge $\mathsf{stackComp}$ from each configuration $(q, s)$ to a configuration $(\boldsymbol{s?}, s)$ for a distinguished control-state $\boldsymbol{s?}$. This allows stacks from different configurations to be manipulated and compared without prejudice to their control-states. Also add an edge labelled $\boldsymbol{p}$ for every $p \in Q$ such that for any control-state $\hat{q}$ of $\mathsf{GrStripln}_{\boldsymbol{k}}(\mathcal{A})$ and control-state $\hat{p}$ corresponding to $p$ we have $(\hat{q}, s)\boldsymbol{p}(\hat{p}, s)$. Add an edge $\boldsymbol{pop_n}$ from each $(q, s)$ to $(q, pop_n(s))$.

Let us break down each stage of this construction. We need to classify the stacks for which the $k$-meta-annotations are considered ‘correct’—something that $\mathsf{lum}(\mathcal{A})_k^+$ has no control over itself—it is a constraint imposed from the outside. Indeed correctness is only defined for $k$-tuples of configurations since every meta-annotation references reachability to each of $k$ different configurations. This correctness property is known as *consistency*.

▶ **Definition 56.** Let $(q_1, s_1), \dots (q_k, s_k)$ be reachable configurations of $\mathsf{lum}(\mathcal{A})^+_k$. Then we say that this $k$-tuple of configurations is *consistent* just in case the following conditions are met:

- For each $i$ with $1 \leq i \leq k$ it is the case that each $(n-1)$-stack in $s_i$ contains precisely one meta-annotation, which must occur on top of it.
- Suppose that $t \sqsubseteq_n s_i$ for some $1 \leq i \leq k$. Then the meta-annotation on top of $top_n(t)$ must be $((Q^a_1)_{a\in\Sigma}, \dots, (Q^a_k)_{a\in\Sigma})$ where for each $1 \leq j \leq k$:

  $$Q^a_j := \{\, q \in Q \;:\; (q, t)\boldsymbol{r}^{\uparrow}_{\epsilon^* a}(q_j, s_j) \,\}$$

  Note that $Q^a_j = \emptyset$ if there is no $\epsilon^* a$-climb from any configuration $(q, t)$ to $(q_j, s_j)$, which in particular is the case if $pop_n(t) \not\sqsubseteq_n s_j$.

Now let $L$ be the map from $\mathsf{lum}(\mathcal{A})^+$-stacks to $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-$-stacks witnessing the strong isomorphism $\mathcal{G}^\epsilon(\mathsf{lum}(\mathcal{A})^+) \stackrel{\cong}{\cong} \mathcal{G}^\epsilon(\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-) \upharpoonright_{\Pi,\Sigma}$, where $\Pi$ and $\Sigma$ are the unary predicates and edge labels from $\mathsf{lum}(\mathcal{A})^+$ . So if $(q, s)$ is a node of $\mathcal{G}^\epsilon(\mathsf{lum}(\mathcal{A})^+)$, the corresponding node in $\mathcal{G}^\epsilon(\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-)$ is $(q, L(s))$. Note further that since $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-$ is monotonic, deleting destructive transitions will not change the set of reachable configurations. Thus $(q, \mathsf{stripIn}(L(s)))$ is also a reachable configuration of $\mathcal{G}^\epsilon(\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A}))$. By Lemma 37 $L(s)$ is completely determined by $\mathsf{stripIn}(L(s))$, and so for notational convenience we drop the $\mathsf{stripIn}()$ and view $(q, L(s))$ as the 'configuration of $\mathcal{G}^\epsilon(\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A}))$ corresponding to $(q, s)$'.

▶ **Lemma 57.** *For each $k \in \mathbb{N}$ and $n_n$-CPDA $\mathcal{A}$, there exists an MSO formula* $\mathsf{con}(x_1, x_2, \dots, x_k)$ *such that reachable configurations $(q_1, s_1), \dots, (q_k, s_k)$ of* $\mathsf{lum}(\mathcal{A})^+$ *are consistent just in case:*

$$\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A}) \;\vDash\; \mathsf{con}((q_1, L(s_1)), \dots, (q_k, L(s_k)))$$

*and such that for every $i$ with $1 \leq i \leq k$,* $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A}) \;\vDash\; \mathsf{con}(c_1, \dots, c_k)$ *implies that $c_i = (q, L(s))$ for some reachable configuration $(q, s)$ of* $\mathsf{lum}(\mathcal{A})^+$.

**Proof.** First observe that for $\mathsf{lum}(\mathcal{A})^+_k$ configurations $(q, s)$ and $(q', s')$ with corresponding $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$ configurations $x = (q, L(s))$ and $y = (q', L(s'))$ we can MSO-define $s \sqsubseteq_n s'$ in $\mathcal{G}(\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A}))$ using a standard least fixed-point construction:

$$\begin{aligned} x \;\sqsubseteq_n\; y \;:=\; &\exists X.(\exists x'.x\mathsf{stackComp}x')(\exists y'.y\mathsf{stackComp}y'). \\ &\qquad (x' \in X \;\wedge\; \phi_{\sqsubseteq_n}(X, y') \;\wedge\; \forall Y.(\phi_{\sqsubseteq_n}(Y, y') \;\rightarrow\; X \;\subseteq Y)) \end{aligned}$$

where

$$\phi_{\sqsubseteq_n}(X, x', y') \;:=\; \forall z.(z \in X \;\leftrightarrow\; (z = y' \;\vee\; (\exists z' \in X).z'\boldsymbol{pop_n}z))$$

We do indeed have $s \sqsubset_n s'$ iff $L(s) \sqsubseteq_n L(s')$ since strong isomorphisms preserve stack structure.

We additionally need a way of capturing the configurations of $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$ that correspond to the reachable configurations of $\mathsf{lum}(\mathcal{A})^+$, namely those monotonically generated by $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-$:

$$\boldsymbol{R}(x) \;:=\; \exists X.\exists x'.(x' \in X \;\wedge\; \phi_{\boldsymbol{R}}(X) \;\wedge\; \forall Y.(\phi_{\boldsymbol{R}}(Y) \;\rightarrow\; X \;\subseteq Y) \;\wedge\; \bigvee_{a\in\Sigma} x'\boldsymbol{a}x)$$

where

$$\phi_{\boldsymbol{R}}(X) \;:=\; \forall z.(z \in X \;\leftrightarrow\; (z = c_0 \;\vee\; (\exists z' \in X).(z'\boldsymbol{r_\epsilon}z \;\vee \bigvee_{a\in\Sigma\cup\{\epsilon\}} z'\boldsymbol{a}z))$$

where $c_0$ is the initial configuration. Using a standard least-fixed-point construction, $\boldsymbol{R}(x)$ defines those configurations of $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$ reachable via an $(r_\epsilon + \Sigma + \epsilon)^*\Sigma$-labelled path. By Remark B.1.0.1 these are precisely the configurations of $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$ corresponding to those in

$\mathsf{lum}(\mathcal{A})^+$, noting that no $r_\epsilon$-edge is deleted in forming $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$ from $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-$ since the latter is monotonic via $r_\epsilon$. Also note that our w.l.o.g. assumption that all $\Sigma$-labelled edges perform an order-$(n-1)$ operation prevents any $\Sigma$-labelled edge from being deleted.

For $\mathsf{lum}(\mathcal{A})^+_k$ configurations $(q,s)$ and $(q',s')$ with corresponding $\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$ configurations $x = (q, L(s))$ and $y = (q', L(s'))$ we can MSO-define $(q,s)\boldsymbol{r}^{\uparrow}_{\epsilon^* a}(q',s')$ for $a \in \Sigma$ in $\mathcal{G}(\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A}))$ with the $\epsilon^* a$-climb interpreted over $\mathsf{lum}(\mathcal{A})^+$ by defining $x\boldsymbol{r}_{r^*_\epsilon a}y$ in $\mathcal{G}(\mathsf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A}))$. These two are equivalent due to Lemma 29 together with the w.l.o.g assumption that all $\Sigma$-labelled operations are order-$(n-1)$.

$$x\boldsymbol{r}^{\uparrow}_{\epsilon^* a}y \;:=\; x\boldsymbol{r}_{r^*_\epsilon a}y \;:=\; \exists X.(\exists y'.y'\boldsymbol{a}y).(y' \in X \;\wedge\; \phi_{\boldsymbol{r}_{r^*_\epsilon}}(X,x) \wedge \forall Y.(\phi_{\boldsymbol{r}_{r^*_\epsilon}}(Y,x) \;\rightarrow\; X \;\subseteq Y))$$

where

$$\phi_{\boldsymbol{r}_{r^*_\epsilon}}(X,x) \;:=\; \forall z.(z \in X \;\leftrightarrow\; (z = x \;\vee\; (\exists z' \in X).z'\boldsymbol{r_\epsilon}z))$$

We define a predicate **meta** asserting that a configuration has a meta-annotation on top:

$$\mathsf{meta}(x) \;:=\; \bigvee_{m \in M} \mathsf{Met}(m)(x)$$

where $M$ is the set of meta-annotations. We also define the following predicates that can be used to express some basic properties about the meta-annotation on top of a stack:

$$[\boldsymbol{q} \in \boldsymbol{Q^a_i}](x) \quad := \quad \bigvee_{\substack{m=((Q^a_1)_{a\in\Sigma},\ldots,(Q^a_k)_{a\in\Sigma})\in M \\ q \in Q^a_i}} \mathsf{Met}(m)(x)$$

for each $1 \le i \le k$, $a \in \Sigma$ and $q \in Q$. We can now read off the definition of consistency to define:

$$\mathsf{con}(x_1, x_2, \ldots, x_k) := \forall x.\left(\bigvee_{i=1}^{k} x \;\sqsubseteq_n\; x_i\right) \;\rightarrow\; \mathsf{meta}(x) \;\wedge\; \bigwedge_{\substack{1\le i\le k \\ a\in\Sigma, q\in Q}} [\boldsymbol{q} \in \boldsymbol{Q^a_i}](x) \;\leftrightarrow\; \exists y.(\boldsymbol{R}(y) \;\wedge\; x\boldsymbol{q}y \;\wedge\; y\boldsymbol{r}^{\uparrow}_{\epsilon^* a}x_i)$$

◀

► **Lemma 58.** *Let $\mathcal{A}$ be an $n$-CPDA with stack-alphabet $\Gamma$. For every quanfitifer free formula $\phi(x_1, \ldots, x_k)$ in **FO** and configurations $(q_1, s_1), \ldots, (q_k, s_k)$ in $\mathcal{G}^\epsilon(\mathcal{A})$:*

$$\mathcal{G}^\epsilon(\mathcal{A}) \;\vDash\; \phi((q_1,s_1),\ldots,(q_k,s_k)) \quad\Leftrightarrow\quad \mathcal{G}^\epsilon(\mathsf{lum}(\mathcal{A})^+_k) \;\vDash\; \phi((q_1,t_1),\ldots,(q_k,t_k))$$

*whenever $(q_1,t_1),\ldots,(q_k,t_k)$ are consistent reachable configurations of $\mathsf{lum}(\mathcal{A})^+_k$ and $\pi_\Gamma(t_i) = s_i$ for each $1 \le i \le k$.*

*Moreover for every set of $\mathcal{A}$ configurations $(q_1,s_1),\ldots,(q_k,s_k)$ there exists a consistent set of reachable configurations $(q_1,t_1),\ldots,(q_k,t_k)$ of $\mathsf{lum}(\mathcal{A})^+_k$ such that $\pi_\Gamma(t_i) = s_i$ for each $1 \le i \le k$.*

**Proof.** For the first part argue by induction on the structure of $\phi$. For the base case note that unary predicates are inherited directly from $\mathcal{A}$ and the fact we are considering $\epsilon$-closure ensures that binary relations are also directly inherited, despite the additional steps of maintaining meta-annotations. For equality we must appeal to consistency. The $Q_i^a$ component of any meta-annotation $m$ containined in one of the $t_j$ is uniquely determined by $(q_i, s_i)$ (or indeed by $(q_i, t_i)$) and $t_{j \leq m}$ by definition of consistency. Thus for any $1 \leq i, j \leq k$ we will have $(q_i, t_i) = (q_j, t_j)$ just in case $s_i = s_j$.

Conjunction and negation are straightforward applications of the induction hypothesis.

The second part is immediate from the fact that we just need to choose $Q_i^a$ for each meta-annotation to be the unique set specified by $(q_i, s_i)$. ◀

▶ **Lemma 59.** *Let $\mathcal{A}$ be an $n$-CPDA. For every $\Sigma_1$ sentence $\phi$ in **FO** we can construct an MSO sentence $\hat{\phi}$ such that:*

$$\mathcal{G}^{\epsilon}(\mathcal{A}) \vDash \phi \quad \Leftrightarrow \quad \mathcal{G}(\mathbf{GrStripIn}_{\mathcal{A}}(k)) \vDash \hat{\phi}$$

**Proof.** Without loss of generality (due to prenex normal form) let us assume that $\phi = \exists x_1.\exists x_2. \cdots \exists x_k.\phi'(x_1, \ldots, x_k)$ where $\phi'$ is quantifier free. For each reachable configuration $(q, s)$ of $\mathbf{lum}(\mathcal{A})_k^+$ let $(q, L(s))$ be the corresponding reachable configuration of $\mathbf{GrStripIn}_{\mathcal{A}}(k)$. By Lemma 58 it suffices to construct a quantifier free MSO formula $\hat{\phi}'(x_1, \ldots, x_k)$ such that for *consistent* $(q_1, s_1), \ldots, (q_k, s_k)$:

$$\mathcal{G}^{\epsilon}(\mathbf{lum}(\mathcal{A})_k^+) \vDash \phi'((q_1, s_1), \ldots, (q_k, s_k)) \quad \Leftrightarrow$$
$$\mathcal{G}(\mathbf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})) \vDash \hat{\phi}'((q_1, L(s_1)), \ldots, (q_k, L(s_k)))$$

We can then take

$$\hat{\phi} := \exists x_1 \cdots x_k \left( \bigwedge_{i=1}^{k} \boldsymbol{R}(x_i) \wedge \mathbf{con}(x_1, \ldots, x_k) \wedge \hat{\phi}'(x_1, \ldots, x_k) \right)$$

where **con** is the MSO formula taken from Lemma 57 and $\boldsymbol{R}$ is the reachability predicate taken from the proof of Lemma 57.

We define $\hat{\phi}'$ by induction on the structure of $\phi'$. The atomic cases of equality and unary predicates can have $\hat{\phi}' = \phi'$ due to the strong isomorphism between $\mathbf{lum}(\mathcal{A})_k^+$ and $\mathbf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-$ together with the fact that removing links does not affect equality for $\mathbf{lum}(\mathcal{A})_k^+$. Binary relations must be given a more sophisticated translation since some edges are removed in forming $\mathbf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$. We therefore appeal to Lemma 33 telling us that $(q, s)\boldsymbol{a}(q', s')$ in $\mathcal{G}^{\epsilon}(\mathbf{lum}(\mathcal{A})_k^+)$ just in case $(q, L(s))\boldsymbol{b}_a(q', L(s'))$ in $\mathbf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})^-$. When $\phi'(x, y) = x\boldsymbol{a}x_i$ we thus express the existence of such a bounce by taking:

$$\hat{\phi}'(x, x_i) := \bigvee_{q \in Q} \bigwedge_{m \in M_{q,a}^i} (\mathbf{Met}(q, m)(x))^{\downarrow}$$

where $Q$ is the set of control-states of $\mathcal{A}$ and $M_{q,a}^i$ is the set of meta-annotations $((Q_1^a)_{a \in \Sigma}, \ldots, (Q_k^a)_{a \in \Sigma})$ such that $q \in Q_i^a$.

Negation and conjunction is a trivial application of the induction hypothesis. ◀

▶ Remark. The method behind the proof of Lemma 59 does not generalise to sentences with quantifier alternation since consistency requires a $k$-tuple of stacks to be fixed. If one fixes a stack with an existential quantification and then endeavours to add a universal quantification, there may be some stack over which the universal quantifier ranges that does not honour the information in the meta-annotations embedded in the fixed (existentially quantified) stack.

Since $\mathbf{GrStripIn}_{\boldsymbol{k}}(\mathcal{A})$ is an $n$-PDA it must be the case that $\mathcal{G}^{\epsilon}(\mathbf{GrStripIn}_{\mathcal{A}}(k))$ has decidable MSO theory [5]. Lemma 59 therefore implies:

## Theorem 38

Let $\mathcal{A}$ be an $n_n$-CPDA. Then the $\Sigma_1$ theory of $\mathcal{G}^{\epsilon}(\mathcal{A})$ is decidable.