Saturation of Concurrent Collapsible Pushdown Systems

M. Hague

Royal Holloway University of London, and LIGM, Marne-la-Vallée
matthew.hague@rhul.ac.uk

Abstract

Multi-stack pushdown systems are a well-studied model of concurrent computation using threads with first-order procedure calls. While, in general, reachability is undecidable, there are numerous restrictions on stack behaviour that lead to decidability. To model higher-order procedures calls, a generalisation of pushdown stacks called collapsible pushdown stacks are required. Reachability problems for multi-stack collapsible pushdown systems have been little studied. Here, we study ordered, phase-bounded and scope-bounded multi-stack collapsible pushdown systems using saturation techniques, showing decidability of control state reachability and giving a regular representation of all configurations that can reach a given control state.

1 Introduction

Pushdown systems augment a finite-state machine with a stack and accurately model first-order recursion. Such systems then are ideal for the analysis of sequential first-order programs and several successful tools, such as Moped [25] and SLAM [3], exist for their analysis. However, the domination of multi- and many-core machines means that programmers must be prepared to work in concurrent environments, with several interacting execution threads.

Unfortunately, the analysis of concurrent pushdown systems is well-known to be undecidable. However, most concurrent programs don't interact pathologically and many restrictions on interaction have been discovered that give decidability (e.g. [5, 6, 26, 14, 15]).

One particularly successful approach is *context-bounding*. This underapproximates a concurrent system by bounding the number of context switches that may occur [24]. It is based on the observation that most real-world bugs require only a small number of thread interactions [23]. Additionally, a number of more relaxed restrictions on stack behaviour have been introduced. In particular phase-bounded [29], scope-bounded [30], and ordered [7] (corrected in [2]) systems. There are also generic frameworks — that bound the tree- [20] or split-width [10] of the interactions between communication and storage — that give decidability for all communication architectures that can be defined within them.

Languages such as C++, Haskell, Javascript, Python, or Scala increasingly embrace higher-order procedure calls, which present a challenge to verification. A popular approach to modelling higher-order languages for verification is that of (higher-order recursion) schemes [11, 21, 16]. Collapsible pushdown systems (CPDS) are an extension of pushdown systems [13] with a "stack-of-stacks" structure. The "collapse" operation allows a CPDS to retrieve information about the context in which a stack character was created. These features give CPDS equivalent modelling power to schemes [13].

These two formalisms have good model-checking properties. E.g, it is decidable whether a μ-calculus formula holds on the execution graph of a scheme [21] (or CPDS [13]). Although, the complexity of such analyses is high, it has been shown by Kobayashi [15] (and Broadbent *et al.* for CPDS [9]) that they can be performed in practice on real code examples.

However concurrency for these models has been little studied. Work by Seth considers phase-bounding for CPDS without collapse [27] by reduction to a finite state parity game. Recent work by Kobayashi and Igarashi studies context-bounded recursion schemes [17].

Here, we study global reachability problems for ordered, phase-bounded, and scope-bounded CPDS. We use *saturation* methods, which have been successfully implemented by e.g. Moped [25] for pushdown systems and C-SHORe [9] for CPDS. Saturation was first applied to model-checking by Bouajjani *et al.* [4] and Finkel *et al.* [12]. We presented a saturation technique for CPDS in ICALP 2012 [8]. Here, we present the following advances.

1. Global reachability for ordered CPDSs (§5). This is based on Atig's algorithm [1] for ordered PDSs and requires a non-trivial generalisation of his notion of *extended* PDSs (§3). For this we introduce the notion of *transition automata* that encapsulate the behaviour of the saturation algorithm. In Appendix F we show how to use the same machinery to solve the global reachability problem for phase-bounded CPDSs.
2. Global reachability for scope-bounded CPDSs (§6). This is a backwards analysis based upon La Torre and Napoli's forwards analysis for scope-bounded PDSs, requiring new insights to complete the proofs.

Because the naive encoding of a single second-order stack has an undecidable MSO theory (we show this folklore result in Appendix A) it remains a challenging open problem to generalise the generic frameworks above ([20, 10]) to CPDSs, since these frameworks rely on MSO decidability over graph representations of the storage and communication structure.

2 Preliminaries

Before defining CPDSs, we define $2\uparrow_0(x) = x$ and $2\uparrow_{i+1}(x) = 2^{2\uparrow_i(x)}$.

2.1 Collapsible Pushdown Systems (CPDS)

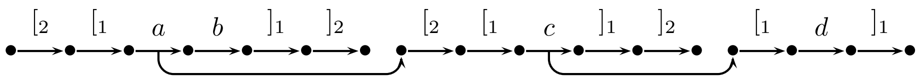
For a readable introduction to CPDS we defer to a survey by Ong [22]. Here, we can only briefly describe higher-order collapsible stacks and their operations. We use a notion of collapsible stacks called *annotated stacks* (which we refer to as collapsible stacks). These were introduced in ICALP 2012, and are essentially equivalent to the classical model [8].

Higher-Order Collapsible Stacks An order-1 stack is a stack of symbols from a stack alphabet Σ, an order-n stack is a stack of order-$(n-1)$ stacks. A collapsible stack of order n is an order-n stack in which the stack symbols are annotated with collapsible stacks which may be of any order $\leq n$. Note, often in examples we will omit annotations for clarity. We fix the maximal order to n, and use k to range between n and 1. We simultaneously define for all $1 \leq k \leq n$, the set Stacks^n_k of order-k stacks whose symbols are annotated by stacks of order at most n. Note, we use subscripts to indicate the order of a stack. Furthermore, the definition below uses a least fixed-point. This ensures that all stacks are finite. An order-k stack is a collapsible stack in Stacks^n_k.

Definition 2.1 (Collapsible Stacks) *The family of sets* $(\mathrm{Stacks}^n_k)_{1\leq k\leq n}$ *is the smallest family (for point-wise inclusion) such that:*

1. *for all* $2 \leq k \leq n$, Stacks^n_k *is the set of all (possibly empty) sequences* $[w_1 \dots w_\ell]_k$ *with* $w_1, \dots, w_\ell \in \mathrm{Stacks}^n_{k-1}$.
2. Stacks^n_1 *is all sequences* $[a_1{}^{w_1} \dots a_\ell{}^{w_\ell}]_1$ *with* $\ell \geq 0$ *and for all* $1 \leq i \leq \ell$, a_i *is a stack symbol in* Σ *and* w_i *is a collapsible stack in* $\bigcup\limits_{1 \leq k \leq n} \mathrm{Stacks}^n_k$.

An order-n stack can be represented naturally as an edge-labelled tree over the alphabet $\{[_{n-1}, \dots, [_1,]_1, \dots,]_{n-1}\} \uplus \Sigma$, with Σ-labelled edges having a second target to the tree representing the annotation. We do not use $[_n$ or $]_n$ since they would appear uniquely at the beginning and end of the stack. An example order-3 stack is given below, with only a few annotations shown (on a and c). The annotations are order-3 and order-2 respectively.



Given an order-n stack $w = [w_1 \dots w_\ell]_n$, we define $top_{n+1}(w) = w$ and

$$\begin{array}{rcll} top_n([w_1 \dots w_\ell]_n) & = & w_1 & \text{when } \ell > 0 \\ top_n([]_n) & = & []_{n-1} & \text{otherwise} \\ top_k([w_1 \dots w_\ell]_n) & = & top_k(w_1) & \text{when } k < n \text{ and } \ell > 0 \end{array}$$

noting that $top_k(w)$ is undefined if $top_{k'}(w) = []_{k'-1}$ for any $k' > k$.

We write $u :_k v$ — where u is order-$(k-1)$ — to denote the stack obtained by placing u on top of the top_k stack of v. That is, if $v = [v_1 \dots v_\ell]_k$ then $u :_k v = [uv_1 \dots v_\ell]_k$, and if $v = [v_1 \dots v_\ell]_{k'}$ with $k' > k$, $u :_k v = [(u :_k v_1)\, v_2 \dots v_\ell]_{k'}$. This composition associates to the right. E.g., the stack $[[[a^w b]_1]_2]_3$ above can be written $u :_3 v$ where u is the order-2 stack $[[a^w b]_1]_2$ and v is the empty order-3 stack $[]_3$. Then $u :_3 u :_3 v$ is $[[[a^w b]_1]_2[[a^w b]_1]_2]_3$.

Operations on Order-n Collapsible Stacks The following operations can be performed on an order-n stack where *noop* is the null operation $noop(w) = w$.

$$\mathcal{O}_n \quad = \quad \{noop, pop_1\} \cup \left\{rew_a, push^k_a, copy_k, pop_k \mid a \in \Sigma \wedge 2 \leq k \leq n\right\}$$

We define each $o \in \mathcal{O}_n$ for an order-n stack w. Annotations are created by $push^k_a$, which pushes a character onto w and annotates it with $top_{k+1}(pop_k(w))$. This, in essence, attaches a closure to a new character.

1. We set $pop_k(u :_k v) = v$.
2. We set $copy_k(u :_k v) = u :_k u :_k v$.
3. We set $collapse_k\left(a^{u'} :_1 u :_{(k+1)} v\right) = u' :_{(k+1)} v$ when u is order-k and $1 \leq k < n$; and $collapse_n(a^u :_1 v) = u$ when u is order-n.
4. We set $push^k_b(w) = b^u :_1 w$ where $u = top_{k+1}(pop_k(w))$.
5. We set $rew_b(a^u :_1 v) = b^u :_1 v$.

For example, beginning with $[[a]_1[b]_1]_2$ and applying $push^2_c$ we obtain $[[c^{[[b]_1]_2}a]_1[b]_1]_2$. In this setting, the order-2 context information for the new character c is $[[b]_1]_2$. We can then apply $copy_2; collapse_2$ to get $[[c^{[[b]_1]_2}a]_1[c^{[[b]_1]_2}a]_1[b]_1]_2$ then $[[b]_1]_2$. That is, $collapse_k$ replaces the current top_{k+1} stack with the annotation attached to c.

Collapsible Pushdown Systems We are now ready to define collapsible PDS.

Definition 2.2 (Collapsible Pushdown Systems) *An order-n* collapsible pushdown system (n-CPDS) *is a tuple* $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R})$ *where* $\mathcal{P}$ *is a finite set of control states,* Σ *is a finite stack alphabet, and* $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P})$ *is a set of rules.*

We write *configurations* of a CPDS as a pair $\langle p, w\rangle \in \mathcal{P} \times \text{Stacks}_n^n$. We have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ via a rule (p, a, o, p') when $top_1(w) = a$ and $w' = o(w)$.

Consuming and Generating Rules We distinguish two kinds of rule or operation: a rule (p, a, o, p') or operation o is *consuming* if $o = pop_k$ or $o = collapse_k$ for some k. Otherwise, it is *generating*. We write $\mathcal{R}_{\mathcal{G}_n}^{\mathcal{P},\Sigma}$ for the set of generating rules of the form (p, a, o, p') such that $p, p' \in \mathcal{P}$ and $a \in \Sigma$, and $o \in \mathcal{O}_n$. We simply write $\mathcal{R}_{\mathcal{G}_n}$ when no confusion may arise.

2.2 Saturation for CPDS

Our algorithms for concurrent CPDSs build upon the saturation technique for CPDSs [8]. In essence, we represent sets of configurations C using a $\mathcal{P}$-stack automaton A reading stacks. We define such automata and their languages $\mathcal{L}(A)$ below. Saturation adds new transitions to A — depending on rules of the CPDS and existing transitions in A — to obtain A' representing configurations with a path to a configuration in C. I.e., given a CPDS $\mathcal{C}$ with control states $\mathcal{P}$ and a $\mathcal{P}$-stack automaton A_0, we compute $Pre_{\mathcal{C}}^*(A_0)$ which is the smallest set s.t. $Pre_{\mathcal{C}}^*(A_0) \supseteq \mathcal{L}(A_0)$ and $Pre_{\mathcal{C}}^*(A_0) \supseteq \{\langle p, w\rangle \mid \exists \langle p, w\rangle \longrightarrow \langle p', w'\rangle \text{ s.t. } \langle p', w'\rangle \in Pre_{\mathcal{C}}^*(A_0)\}$.

Stack Automata Sets of stacks are represented using order-n stack automata. These are alternating automata with a nested structure that mimics the nesting in a higher-order collapsible stack. We recall the definition below.

Definition 2.3 (Order-n Stack Automata) *An* order-n stack automaton *is a tuple* $A = (\mathbb{Q}_n, \ldots, \mathbb{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1)$ *where* Σ *is a finite stack alphabet,* $\mathbb{Q}_n, \ldots, \mathbb{Q}_1$ *are disjoint, and*

1. *for all* $2 \leq k \leq n$, *we have* $\mathbb{Q}_k$ *is a finite set of states,* $\mathcal{F}_k \subseteq \mathbb{Q}_k$ *is a set of accepting states, and* $\Delta_k \subseteq \mathbb{Q}_k \times \mathbb{Q}_{k-1} \times 2^{\mathbb{Q}_k}$ *is a transition relation such that for all* q *and* Q *there is* at most one q' *with* $(q, q', Q) \in \Delta_k$, *and*
2. $\mathbb{Q}_1$ *is a finite set of states,* $\mathcal{F}_1 \subseteq \mathbb{Q}_1$ *is a set of accepting states, and the transition relation is* $\Delta_1 \subseteq \bigcup_{2 \leq k \leq n} \left(\mathbb{Q}_1 \times \Sigma \times 2^{\mathbb{Q}_k} \times 2^{\mathbb{Q}_1}\right)$.

States in $\mathbb{Q}_k$ recognise order-k stacks. Stacks are read from "top to bottom". A stack $u :_k v$ is accepted from q if there is a transition $(q, q', Q) \in \Delta_k$, written $q \xrightarrow{q'} Q$, such that u is accepted from $q' \in \mathbb{Q}_{(k-1)}$ and v is accepted from each state in Q. At order-1, a stack $a^u :_1 v$ is accepted from q if there is a transition (q, a, Q_{col}, Q) where u is accepted from all states in Q_{col} and v is accepted from all states in Q. An empty order-k stack is accepted by any state in $\mathcal{F}_k$. We write $w \in \mathcal{L}_q(A)$ to denote the set of all stacks w accepted from q. Note that a transition to the empty set is distinct from having no transition.

We show a part run using $q_3 \xrightarrow{q_2} Q_3 \in \Delta_3$, $q_2 \xrightarrow{q_1} Q_2 \in \Delta_2$, $q_1 \xrightarrow[Q_{col}]{a} Q_1 \in \Delta_1$.

$$\begin{tikzcd} q_3 \arrow[r, "{[_2}"] & q_2 \arrow[r, "{[_1}"] & q_1 \arrow[r, "a"] \arrow[rrrrr, bend right, to=Qc] & Q_1 \arrow[r, "b"] & \cdots \arrow[r, "{]_1}"] & Q_2 \arrow[r, "{]_2}"] & Q_3 & |[alias=Qc]| Q_{col} \arrow[r, "{[_2}"] & \cdots \end{tikzcd}$$

Long-form Transitions We will often use a *long-form* notation (defined below) that captures nested sequences of transitions. E.g. we can write $q_3 \xrightarrow[Q_{col}]{a} (Q_1, Q_2, Q_3)$ to represent the use of $q_3 \xrightarrow{q_2} Q_3$, $q_2 \xrightarrow{q_1} Q_2$, and $q_1 \xrightarrow[Q_{col}]{a} Q_1$ for the first three transitions of the run above. Note that this latter long-form transition starts at the very beginning of the stack and reads its top_1 character. Formally, for a sequence of transitions $q \xrightarrow{q_{k-1}} Q_k, q_{k-1} \xrightarrow{q_{k-2}} Q_{k-1}, \ldots, q_1 \xrightarrow[Q_{col}]{a} Q_1$ in Δ_k to Δ_1 respectively, we write $q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k)$.

$\mathcal{P}$-Stack Automata We define $\mathcal{P}$-automata [4] for CPDSs. Given control states $\mathcal{P}$, an *order-n $\mathcal{P}$-stack automaton* is an order-n stack automaton such that for each $p \in \mathcal{P}$ there exists a state $q_p \in \mathbb{Q}_n$. We set $\mathcal{L}(A) = \left\{ \langle p, w \rangle \;\middle|\; w \in \mathcal{L}_{q_p}(A) \right\}$.

The Saturation Algorithm We recall the saturation algorithm. For a detailed explanation of the saturation function complete with examples, we refer the reader to our ICALP paper [8]. Here we present an abstracted view of the algorithm, relegating details that are not directly relevant to the remainder of the main article to Appendix B.

The saturation algorithm iterates a *saturation function* Π that adds new transitions to a given automaton. Beginning with A_0 representing a target set of configurations, we iterate $A_{i+1} = \Pi(A_i)$ until $A_{i+1} = A_i$. Once this occurs, we have that $\mathcal{L}(A_i) = Pre^*_{\mathcal{C}}(A_0)$.

We define Π in terms of a family of auxiliary saturation functions Π_r (defined in Appendix B) which return a set of long-form transitions to be added by saturation. When r is consuming, $\Pi_r(A)$ returns the set of long-form transitions to be added to A due to the rule r. When r is generating Π_r also takes as an argument a long-form transition t of A. Thus $\Pi_r(t, A)$ returns the set of long-form transitions that should be added to A as a result of the rule r combined with the transition t (and possibly other transitions of A).

For example, if $r = (p, a, rew_b, p')$ and $t = q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ is a transition of A, then $\Pi_r(t, A)$ contains only the long-form transition $t' = q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$. The idea is if $\langle p', b^u :_1 w \rangle$ is accepted by A via a run whose first (sequence of) transition(s) is t, then by adding t' we will be able to accept $\langle p, a^u :_1 w \rangle$ via a run beginning with t' instead of t. We have $\langle p, a^u :_1 w \rangle \in Pre^*_{\mathcal{C}}(A)$ since it can reach $\langle p', b^u :_1 w \rangle$ via the rule r.

Definition 2.4 (The Saturation Function Π) *For a CPDS with rules $\mathcal{R}$, and given an order-n stack automaton A_i we define $A_{i+1} = \Pi(A_i)$. The state-sets of A_{i+1} are defined implicitly by the transitions which are those in A_i plus, for each $r = (p, a, o, p') \in \mathcal{R}$, when*

1. *o is consuming and $t \in \Pi_r(A_i)$, then add t to A_{i+1},*
2. *o is generating, t is in A_i, and $t' \in \Pi_r(t, A)$, then add t' to A_{i+1}.*

In ICALP 2012 we showed that saturation adds up to $\mathcal{O}(2 \uparrow_n (f(|\mathcal{P}|)))$ transitions, for some polynomial f, and that this can be reduced to $\mathcal{O}(2 \uparrow_{n-1} (f(|\mathcal{P}|)))$ (which is optimal) by restricting all Q_n to have size 1 when A_0 is "non-alternating at order-n". Since this property holds of all A_0 used here, we use the optimal algorithm for complexity arguments.

3 Extended Collapsible Pushdown Systems

To analyse concurrent systems, we extend CPDS following Atig [1]. Atig's extended PDSs allow words from arbitrary languages to be pushed on the stack. Our notion of extended CPDSs allows sequences of *generating operations* from a language $\mathcal{L}_g$ to be applied, rather than a single operation per rule. We can specify $\mathcal{L}_g$ by any system (e.g. a Turing machine).

Definition 3.1 (Extended CPDSs) *An order-n* extended CPDS (n-ECPDS) *is a tuple $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R})$ where $\mathcal{P}$ is a finite set of control states, Σ is a finite stack alphabet, and $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P}) \cup \left(\mathcal{P} \times \Sigma \times 2^{\left(\mathcal{R}^{\mathcal{P},\Sigma}_{\mathcal{G}_n}\right)^*} \times \mathcal{P}\right)$ is a set of rules.*

As before, we have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ of an n-ECPDS via a rule (p, a, o, p') with $top_1(w) = a$ and $w' = o(w)$. Additionally, we have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ when we have a rule $(p, a, \mathcal{L}_g, p')$, a sequence $(p, a, o_1, p_1)\,(p_1, a_2, o_2, p_2) \ldots (p_{\ell-1}, a_\ell, o_\ell, p') \in \mathcal{L}_g$ and $w' = o_\ell(\cdots o_1(w))$. That is, a single extended rule may apply a sequence of stack updates in one step. A run of an ECPDS is a sequence $\langle p_0, w_0\rangle \longrightarrow \langle p_1, w_1\rangle \longrightarrow \cdots$.

3.1 Reachability Analysis

We adapt saturation for ECPDSs. In Atig's algorithm, an essential property is the decidability of $\mathcal{L}_g \cap \mathcal{L}(A)$ for some order-1 $\mathcal{P}$-stack automaton A and a language $\mathcal{L}_g$ appearing in a rule of the extended PDS. We need analogous machinery in our setting. For this, we first define a class of finite automata called *transition* automata, written $\mathcal{T}$. The states of these automata will be long-form transitions of a stack automaton $t = q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$. Transitions $t \xrightarrow{r} t'$ are labelled by rules. We write $t \xrightarrow{\overrightarrow{r}}_* t'$ to denote a run over $\overrightarrow{r} \in (\mathcal{R}_{\mathcal{G}_n})^*$.

During the saturation algorithm we will build from A_i a transition automaton $\mathcal{T}$. Then, for each rule $(p, a, \mathcal{L}_g, p')$ we add to A_{i+1} a new long-form transition t if there is a word $\overrightarrow{r} \in \mathcal{L}_g$ such that $t \xrightarrow{\overrightarrow{r}}_* t'$ is a run of $\mathcal{T}$ and t' is already a transition of A_i.

For example, consider $(p, a, \mathcal{L}_g, p')$ where $\mathcal{L}_g = \{(p, a, rew_b, p')\}$. A transition

$$\left(q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)\right) \xrightarrow{(p,a,rew_b,p')} \left(q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)\right)$$

will correspond to the fact that the presence of $q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ in A_i causes $q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ to be added by Π. A run $t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} t_3$ comes into play when e.g. $\mathcal{L}_g = \{r_1 r_2\}$. If the rule were split into two ordinary rules with intermediate control states, Π would first add t_2 derived from t_3, and then from t_2 derive t_1. In the case of extended CPDSs, the intermediate transition t_2 is not added to A_{i+1}, but its effect is still present in the addition of t_1. Below, we repeat the above intuition more formally. Fix a n-ECPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R})$.

Transition Automata We build a transition automaton from a given $\mathcal{P}$-stack automaton A. Let A have order-n to order-1 state-sets $Q_n, \ldots, Q_1$ and alphabet Σ, let T_A be the set of all $q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ with $q \in Q_n$, for all k, $Q_k \subseteq \mathbb{Q}_k$, and for some k, $Q_{col} \subseteq \mathbb{Q}_k$.

Definition 3.2 (Transition Automata) *Given an order-n $\mathcal{P}$-stack automaton A with alphabet Σ, and $t, t' \in T_A$, we define the transition automaton $\mathcal{T}^A_{t,t'} = \left(T_A, \mathcal{R}^{\mathcal{P},\Sigma}_{\mathcal{G}_n}, \delta, t, t'\right)$ such that $\delta \subseteq T_A \times \mathcal{R}^{\mathcal{P},\Sigma}_{\mathcal{G}_n} \times T_A$ is the smallest set such that $t_1 \xrightarrow{r} t_2 \in \delta$ if $t_1 \in \Pi_r(t_2, A)$.*

We define $\mathcal{L}\big(\mathcal{T}^A_{t,t'}\big) = \left\{\overrightarrow{r} \;\middle|\; t \xrightarrow{\overrightarrow{r}}_* t'\right\}$.

Extended Saturation Function We now extend the saturation function following the intuition explained above. For $t = q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$, let $top_1(t) = a$ and $control(t) = p$.

Definition 3.3 (Extended Saturation Function Π) *The extended Π is Π from Definition 2.4 plus for each extended rule $(p, a, \mathcal{L}_g, p') \in \mathcal{R}$ and t, t', we add t to A_{i+1} whenever 1. $control(t) = p$ and $top_1(t) = a$, 2. t' is a transition of A_i with $control(t') = p'$, and 3. $\mathcal{L}_g \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$.*

Theorem 3.1 (Global Reachability of ECPDS) *Given an ECPDS $\mathcal{C}$ and a $\mathcal{P}$-stack automaton A_0, the fixed point A of the extended saturation procedure accepts $Pre^*_{\mathcal{C}}(A_0)$.*

In order for the saturation algorithm to be effective, we need to be able to decide $\mathcal{L}_g \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$. We argue in the appendix that number of transitions added by extended saturation has the same upper bound as the unextended case.

4 Multi-Stack CPDSs

We define a general model of concurrent collapsible pushdown systems, which we later restrict. In the sequel, assume a bottom-of-stack symbol $\perp$ and define the "empty" stacks $\perp_0=\perp$ and $\perp_{k+1}= [\perp_k]_{k+1}$. As standard, we assume that $\perp$ is neither pushed onto, nor popped from, the stack (though may be copied by $copy_k$).

Definition 4.1 (Multi-Stack Collapsible Pushdown Systems) *An order-n* multi-stack collapsible pushdown system (n-MCPDS) *is a tuple $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ where $\mathcal{P}$ is a finite set of control states, Σ is a finite stack alphabet, and for each $1 \leq i \leq m$ we have a set of rules $\mathcal{R}_i \subseteq \mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P}$.*

A configuration of $\mathcal{C}$ is a tuple $\langle p, w_1, \ldots, w_m\rangle$. There is a transition $\langle p, w_1, \ldots, w_m\rangle \longrightarrow \langle p', w_1, \ldots, w_{i-1}, w'_i, w_{i+1}, \ldots, w_m\rangle$ via $(p, a, o, p') \in \mathcal{R}_i$ when $a = top_1(w_i)$ and $w'_i = o(w_i)$.

We also need MCPD*Automata*, which are MCPDSs defining languages over an input alphabet Γ. For this, we add labelling input characters to the rules. Thus, a rule (p, a, γ, o, p') reads a character $\gamma \in \Gamma$. This is defined formally in Appendix D.

We are interested in two problems for a given n-MCPDS $\mathcal{C}$.

Definition 4.2 (Control State Reachability Problem) *Given control states p_{in}, p_{out} of $\mathcal{C}$, decide if there is for some $w_1, \ldots, w_m$ a run $\langle p_{in}, \perp_n, \ldots, \perp_n\rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, w_1, \ldots, w_m\rangle$.*

Definition 4.3 (Global Control State Reachability Problem) *Given a control state p_{out} of $\mathcal{C}$, construct a representation of the set of configurations $\langle p, w_1, \ldots, w_m\rangle$ such that there exists for some $w'_1, \ldots, w'_m$ a run $\langle p, w_1, \ldots, w_m\rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, w'_1, \ldots, w'_m\rangle$.*

We represent sets of configurations as follows. In Appendix D we show it forms an effective boolean algebra, membership is linear time, and emptiness is in PSPACE.

Definition 4.4 (Regular Set of Configurations) *A regular set R of configurations of a multi-stack CPDS $\mathcal{C}$ is definable via a finite set χ of tuples $(p, A_1, \ldots, A_m)$ where p is a control state of $\mathcal{C}$ and A_i is a stack automaton with designated initial state q_i for each i. We have $\langle p, w_1, \ldots, w_m\rangle \in R$ iff there is some $(p, A_1, \ldots, A_m) \in \chi$ such that $w_i \in \mathcal{L}_{q_i}(A_i)$ for each i.*

Finally, we often partition runs of an MCPDS $\sigma = \sigma_1 \ldots \sigma_\ell$ where each σ_i is a sequence of configurations of the MCPDS. A transition from c to c' occurs in segment σ_i if c' is a configuration in σ_i. Thus, transitions from σ_i to σ_{i+1} are said to belong to σ_{i+1}.

5 Ordered CPDS

We generalise *ordered multi-stack pushdown systems* [7]. Intuitively, we can only remove characters from stack i whenever all stacks $j < i$ are empty.

Definition 5.1 (Ordered CPDS) *An order-n* ordered CPDS *(n-OCPDS) is an n-MCPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ such that a transition from $\langle p, w_1, \ldots, w_m \rangle$ using the rule r on stack i is permitted iff, when r is consuming, for all $1 \leq j < i$ we have $w_j = \perp_n$.*

Theorem 5.1 (Decidability of Reachability Problems) *For n-OCPDSs the control state reachability problem and the global control state reachability problem are decidable.*

We outline the proofs below. In Appendix E we show control state reachability uses $\mathcal{O}\big(2 \uparrow_{m(n-1)} (\ell)\big)$ time, where ℓ is polynomial in the size of the OCPDS, and we have at most $\mathcal{O}(2 \uparrow_{mn} (\ell))$ tuples in the solution to the global problem. First observe that reachability can be reduced to reaching $\langle p_{\text{out}}, \perp_n, \ldots, \perp_n \rangle$ by clearing the stacks at the end of the run.

Control State Reachability Using our notion of ECPDS, we may adapt Atig's inductive algorithm for ordered PDSs [1] for the control state reachability problem. The induction is over the number of stacks. W.l.o.g. we assume that all rules $(p, \perp, o, p')$ of $\mathcal{C}$ have $o = push^n_a$.

In the base case, we have an n-OCPDS with a single stack, for which the global reachability problem is known to be decidable (e.g. [4]).

In the inductive case, we have an n-OCPDS $\mathcal{C}$ with m stacks. By induction, we can decide the reachability problem for n-OCPDSs with fewer than m stacks. We first show how to reduce the problem to reachability analysis of an extended CPDS, and then finally we show how to decide $\mathcal{L}_g \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$ using an n-OCPDS with $(m-1)$ stacks.

Consider the mth stack of $\mathcal{C}$. A run of $\mathcal{C}$ can be split into $\sigma_1\tau_1\sigma_2\tau_2 \ldots \sigma_\ell\tau_\ell$. During the subruns σ_i, the first $(m-1)$ stacks are non-empty, and during τ_i, the first $(m-1)$ stacks are empty. Moreover, during each σ_i, only generating operations may occur on stack m.

We build an extended CPDS that directly models the mth stack during the τ_i segments where the first $(m-1)$ stacks are empty, and uses rules of the form $(p, a, \mathcal{L}_g, p')$ to encapsulate the behaviour of the σ_i sections where the first $(m-1)$ stacks are non-empty. The $\mathcal{L}_g$ attached to such a rule is the sequence of updates applied to the mth stack during σ_i.

We begin by defining, from the OCPDS $\mathcal{C}$ with m stacks, an OCPDA $\mathcal{C}^L$ with $(m-1)$ stacks. This OCPDA will be used to define the $\mathcal{L}_g$ described above. $\mathcal{C}^L$ simulates a segment σ_i. Since all updates to stack m in σ_i are generating, $\mathcal{C}^L$ need only track its top character, hence only keeps $(m-1)$ stacks. The top character of stack m is kept in the control state, and the operations that would have occurred on stack m are output.

Definition 5.2 ($\mathcal{C}^L$) *Given an n-OCPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$, we define $\mathcal{C}^L$ to be an n-OCPDA with $(m-1)$ stacks $\big(\mathcal{P} \times \Sigma, \Sigma, \mathcal{R}'_1 \cup \mathcal{R}', \mathcal{R}'_2, \ldots, \mathcal{R}'_{m-1}\big)$ over input alphabet $\mathcal{R}_{\mathcal{G}_n}$ where for all i*

$$\mathcal{R}'_i = \{((p,a), b, (p, a, noop, p'), o, (p', a)) \mid a \in \Sigma \wedge (p, b, o, p') \in \mathcal{R}_i\}, \text{ and}$$

$$\begin{array}{rcl} \mathcal{R}' & = & \{((p,a), b, r, noop, (p', c)) \mid b \in \Sigma \wedge r = (p, a, rew_c, p') \in \mathcal{R}_m\} \cup \\ & & \{((p,a), b, r, noop, (p', a)) \mid b \in \Sigma \wedge r = (p, a, copy_k, p') \in \mathcal{R}_m\} \cup \\ & & \big\{((p,a), b, r, noop, (p', c)) \mid b \in \Sigma \wedge r = \big(p, a, push^k_c, p'\big) \in \mathcal{R}_m\big\} \cup \\ & & \{((p,a), b, r, noop, (p', a)) \mid b \in \Sigma \wedge r = (p, a, noop, p') \in \mathcal{R}_m\}\ . \end{array}$$

We define the language $\mathcal{L}^{b,i}_{p,a,p'}\big(\mathcal{C}^L\big)$ to be the set of words $\gamma_1 \ldots \gamma_\ell$ such that there exists a run of $\mathcal{C}^L$ over input $\gamma_1 \ldots \gamma_\ell$ from $\langle (p, a), w_1, \ldots, w_{m-1}\rangle$ to $\langle (p', c), \perp_n, \ldots, \perp_n\rangle$ for some c, where $w_i = push^n_b(\perp_n)$ and $w_j = \perp_n$ for all $j \neq i$. This language describes the effect on stack m of a run σ_j from p to p'. (Note, by assumption, all σ_j start with some $push^n_b$.)

We now define the extended CPDS $\mathcal{C}^R$ that simulates $\mathcal{C}$ by keeping track of stack m in its stack and using extended rules based on $\mathcal{C}^L$ to simulate parts of the run where the first $(m-1)$ stacks are not all empty. Note, since all rules operating on $\perp$ (i.e. $(p, \perp, o, p')$) have $o = push^n_b$, rules from $\mathcal{R}_1, \ldots, \mathcal{R}_{m-1}$ may only fire during (or at the start of) the segments where the first $(m-1)$ stacks are non-empty (and thus appear in $\mathcal{R}_{\mathcal{L}_g}$ below).

Definition 5.3 ($\mathcal{C}^R$) *Given an n-OCPDS $\mathcal{C} = (\mathcal{P} \times \Sigma, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ with m stacks, we define $\mathcal{C}^R$ to be an n-ECPDS such that $\mathcal{C}^R = (\mathcal{P}, \Sigma, \mathcal{R}')$ where $\mathcal{R}' = \mathcal{R}_m \cup \mathcal{R}_{\mathcal{L}_g}$ and*

$$\mathcal{R}_{\mathcal{L}_g} = \left\{ \big(p, a, \mathcal{L}^{b,i}_{p_1,a,p_2}\big(\mathcal{C}^L\big), p_2\big) \ \mid a \in \Sigma \wedge (p, \perp, push^n_b, p_1) \in \mathcal{R}_i \wedge 1 \leq i < m \right\}$$

Lemma 5.1 ($\mathcal{C}^R$ simulates $\mathcal{C}$) *Given an n-OCPDS $\mathcal{C}$ and control states p_{in}, p_{out}, we have $\langle p_{in}, w\rangle \in Pre^*_{\mathcal{C}^R}(A)$, where A is the $\mathcal{P}$-stack automaton accepting only the configuration $\langle p_{out}, \perp_n\rangle$ iff $\langle p_{in}, \perp_n, \ldots, \perp_n, w\rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, \perp_n, \ldots, \perp_n\rangle$.*

Lemma 5.1 only gives an effective decision procedure if we can decide $\mathcal{L}_g \cap \mathcal{L}\Big(\mathcal{T}^{A_i}_{t,t'}\Big) \neq \emptyset$ for all rules $(p, a, \mathcal{L}_g, p')$ appearing in $\mathcal{C}^R$. For this, we use a standard product construction between the $\mathcal{C}^L$ associated with $\mathcal{L}_g$, and $\mathcal{T}^{A_i}_{t,t'}$. This gives an ordered CPDS with $(m-1)$ stacks, for which, by induction over the number of stacks, reachability (and emptiness) is decidable. Note, the initial transition of the construction sets up the initial stacks of $\mathcal{C}^L$.

Definition 5.4 ($\mathcal{C}_\emptyset$) *Given the non-emptiness problem $\mathcal{L}^{b,i}_{p_1,a,p_2}\big(\mathcal{C}^L\big) \cap \mathcal{L}\Big(\mathcal{T}^{A_i}_{t,t'}\Big) \neq \emptyset$, where $top_1(t) = a$, $\mathcal{C}^L = (\mathcal{P} \times \Sigma, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_{m-1})$ and $\mathcal{T}^{A_i}_{t,t'} = (T_{A_i}, \mathcal{R}_{\mathcal{G}_n}, \delta, t, t')$, we define an n-OCPDS $\mathcal{C}_\emptyset = \big(\mathcal{P}^\emptyset, \Sigma, \mathcal{R}^\emptyset_1, \ldots, \mathcal{R}^\emptyset_i \cup \mathcal{R}_{I/O}, \ldots, \mathcal{R}^\emptyset_{m-1}\big)$ where, for all $1 \leq i \leq (m-1)$,*

$$\begin{aligned}
\mathcal{P}^\emptyset &= \{p_1, p_2\} \uplus \{(p, t_1) \ \mid t_1 \in T_{A_i} \wedge control(t_1) = p\} \ , \\
\mathcal{R}_{I/O} &= \{(p_1, \perp, push^n_b, (p_1, t))\} \cup \{((p_2, t), \perp, noop, p_2) \ \mid t \in T_{A_i}\} \ , \ \textit{and} \\
\mathcal{R}^\emptyset_i &= \{((p, t_1), c, o, (p', t_2)) \ \mid ((p, top_1(t_1)), c, r, o, (p', top_1(t_2))) \in \mathcal{R}_i \wedge (t_1, r, t_2) \in \Delta\}
\end{aligned}$$

Lemma 5.2 (Language Emptiness for OCPDS) *We have $\mathcal{L}^{b,i}_{p_1,a,p_2}\big(\mathcal{C}^L\big) \cap \mathcal{L}\Big(\mathcal{T}^{A_i}_{t,t'}\Big) \neq \emptyset$ iff, in $\mathcal{C}_\emptyset$ from Definition 5.4, we have that $\langle p_2, \perp_n, \ldots \perp_n\rangle$ is reachable from $\langle p_1, \perp_n, \ldots, \perp_n\rangle$.*

Global Reachability We sketch a solution to the global reachability problem, giving a full proof in Appendix E. From Lemma 5.1 ($\mathcal{C}^R$ simulates $\mathcal{C}$) we gain a representation $A_m = Pre^*_{\mathcal{C}^R}(A)$ of the set of configurations $\langle p, \perp_n, \ldots, \perp_n, w_m\rangle$ that have a run to $\langle p_{\text{out}}, \perp_n, \ldots, \perp_n\rangle$. Now take any $\langle p, \perp_n, \ldots, \perp_n, w_{m-1}, w_m\rangle$ that reaches $\langle p_{\text{out}}, \perp_n, \ldots, \perp_n\rangle$. The run must pass some $\langle p', \perp_n, \ldots, \perp_n, w'_m\rangle$ with $\langle p', w'_m\rangle$ accepted by A_m. From the product construction above, one can (though not immediately) extract a tuple (p, A_{m-1}, A'_m) such that w_{m-1} is accepted by A_{m-1} and w_m is accepted by A'_m. We repeat this reasoning down to stack 1 and obtain a tuple of the form $(p, A_1, \ldots, A_m)$. We can only obtain a finite set of tuples in this manner, giving a solution to the global reachability problem.

6 Scope-Bounded CPDS

Recently, scope-bounded multi-pushdown systems were introduced [30] and their reachability problem was shown to be decidable. Furthermore, reachability for scope- and phase-bounding was shown to be incomparable [30]. Here we consider scope-bounded CPDS.

A run $\sigma = \sigma_1 \ldots \sigma_\ell$ of an MCPDS is *context-partitionable* when, for each σ_i, if a transition in σ_i is via $r \in \mathcal{R}_j$ on stack j, then all transitions of σ_i are via rules in $\mathcal{R}_j$ on stack j. A *round* is a context-partitioned run $\sigma_1 \ldots \sigma_m$, where during σ_i only $\mathcal{R}_i$ is used. A *round-partitionable* run can be partitioned $\sigma_1 \ldots \sigma_\ell$ where each σ_i is a round. A run of an SBCPDS is such that any character or stack removed from a stack must have been created at most ζ rounds earlier. For this, we define pop- and collapse-rounds for stacks. That is, we mark each stack and character with the round in which it was created. When we copy a stack via $copy_k$, the pop-round of the new copy of the stack is the current round. However, all stacks and characters within the copy of u keep the same pop- and collapse-round as in the original u.

E.g. take $[u]_2$ where $u = [ab]_1$, u and a have pop-round 2, and b has pop-round 1. Suppose in round 3 we use $copy_2$ to obtain $[uu]_2$. The new copy of u has pop-round 3 (the current round), but the a and b appearing in the copy of u still have pop-rounds 2 and 1 respectively. If the scope-bound is 2, the latest each a and the original u could be popped is in round 4, but the new u may be popped in round 5.

We will write ${}_{\mathfrak{p}}w$ for a stack w with pop-round $\mathfrak{p}$ and ${}_{\mathfrak{p},\mathfrak{c}}a$ for a character with pop-round $\mathfrak{p}$ and collapse-round $\mathfrak{c}$. Pop- and collapse-rounds will be sometimes omitted for clarity. Note, the outermost stack will always have pop-round 0. In particular, for all $u :_k v$ in the definition below, the pop-round of v is 0.

Definition 6.1 (Pop- and Collapse-Round) *Given a round-partitioned run $\sigma_1 \ldots \sigma_\ell$ we define inductively the pop- and collapse-rounds. The pop- and collapse-round of each stack and character in the first configuration of σ_1 is 0. Take a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ with $\langle p', w'\rangle$ in σ_z via a rule (p, a, o, p'). If $o = noop$ then $w = w'$, otherwise when*

1. *$o = copy_k$ and $w = {}_{\mathfrak{p}}u :_k v$, then $w' = {}_{z}u :_k ({}_{\mathfrak{p}}u :_k v)$ where ${}_{z}u = {}_{z}[{}_{\mathfrak{p}_1}u_1 \ldots {}_{\mathfrak{p}_\ell}u_\ell]_{k-1}$ when ${}_{\mathfrak{p}}u = {}_{\mathfrak{p}}[{}_{\mathfrak{p}_1}u_1 \ldots {}_{\mathfrak{p}_\ell}u_\ell]_{k-1}$.*
2. *$o = push^k_b$, then $w' = {}_{z,\mathfrak{c}}b^{\left({}_{\mathfrak{p}'}u\right)} :_1 w$ where ${}_{\mathfrak{p}'}u = top_{k+1}(pop_k(w))$ and $\mathfrak{c}$ is the pop-round of $top_k(w)$. (Note, when $k = n$, we know $\mathfrak{p}' = 0$ since the top_{n+1} stack is outermost.)*
3. *$o = pop_k$, when $w = u :_k v$ then $w' = v$.*
4. *We set $collapse_k\left(a^{\left({}_{\mathfrak{p}}u'\right)} :_1 u :_{(k+1)} v\right) = {}_{\mathfrak{p}}u' :_{(k+1)} v$ when u is order-k and $1 \leq k < n$; and $collapse_n\left(a^{({}_0u)} :_1 v\right) = {}_0u$ when u is order-n.*
5. *$o = rew_b$ and $w = {}_{\mathfrak{p},\mathfrak{c}}a^{\left({}_{\mathfrak{p}'}u\right)} :_1 v$, then $w' = {}_{\mathfrak{p},\mathfrak{c}}b^{\left({}_{\mathfrak{p}'}u\right)} :_1 v$.*

Definition 6.2 (Scope-Bounded CPDS) *A ζ-scope-bounded n-CPDS (n-SBCPDS) $\mathcal{C}$ is an order-n MCPDS whose runs are all runs of $\mathcal{C}$ that are round-partitionable, that is $\sigma_1 \ldots \sigma_\ell$, such that for all z, if a transition in σ_z from $\langle p, w\rangle$ to $\langle p', w'\rangle$ is*

1. *a pop_k transition with $1 < k \leq n$ and $w = {}_{\mathfrak{p}}u :_k v$, then $z - \zeta \leq \mathfrak{p}$,*
2. *a pop_1 transition with $w = {}_{\mathfrak{p},\mathfrak{c}}a^u :_1 v$, then $z - \zeta \leq \mathfrak{p}$, or*
3. *a $collapse_k$ transition with $w = {}_{\mathfrak{p},\mathfrak{c}}a^u :_1 v$, then $z - \zeta \leq \mathfrak{c}$.*

La Torre and Napoli's decidability proof for the order-1 case already uses the saturation method [30]. However, while La Torre and Napoli use a forwards-reachability analysis, we must use a backwards analysis. This is because the forwards-reachable set of configurations is in general not regular. We thus perform a backwards analysis for CPDS, resulting in a similar approach. However, the proofs of correctness of the algorithm are quite different.

Theorem 6.1 (Decidability of Reachability Problems) *For n-OCPDSs the control state reachability problem and the global control state reachability problem are decidable.*

In Appendix E we show our non-global algorithm requires $\mathcal{O}(2\uparrow_{n-1}(\ell))$ space, where ℓ is polynomial in ζ and the size of the SBCPDS, and we have at most $\mathcal{O}(2\uparrow_{n}(\ell))$ tuples in the global reachability solution. La Torre and Parlato give an alternative control state reachability algorithm at order-1 using *thread interfaces*, which allows sequentialisation [19] and should generalise order-n, but, does not solve the global reachability problem.

Control State Reachability Fix initial and target control states p_{in} and p_{out}. The algorithm first builds a *reachability graph*, which is a finite graph with a certain kind of path iff p_{out} can be reached from p_{in}. To build the graph, we define layered stack automata. These have states q_p^i for each $1 \leq i \leq \zeta$ which represent the stack contents i rounds later. Thus, a layer automaton tracks the stack across ζ rounds, which allows analysis of scope-bounded CPDSs.

Definition 6.3 (ζ-Layered Stack Automata) *A* ζ-layered stack automaton *is a stack automaton A such that $\mathbb{Q}_n = \left\{ q_p^i \mid p \in \mathcal{P} \wedge 1 \leq i \leq \zeta \right\}$.*

A state q_p^i is of layer i. A state q' labelling $q \xrightarrow{q'} Q$ has the same layer as q. We require that there is no $q \xrightarrow{q'} Q$ with $q'' \in Q$ where q is of layer i and q'' is of layer $j < i$. Similarly, there is no $q \xrightarrow[Q_{col}]{a} Q$ with $q' \in Q \cup Q_{col}$ where q is of layer i and q' is of layer $j < i$.

Next, we define several operations from which the reachability graph is constructed. The $\texttt{Predecessor}_j$ operation connects stack j between two rounds. We define for stack j

$$\texttt{Predecessor}_j(A, q_p, q_{p'}) = \texttt{Saturate}_j\big(\texttt{EnvMove}\big(\texttt{Shift}(A), q^1_{p_1}, q^2_{p_2}\big)\big)$$

where definitions of $\texttt{Shift}$, $\texttt{EnvMove}$ and $\texttt{Saturate}_j$ are given in Appendix G. $\texttt{Shift}$ moves transitions in layer i to layer $(i+1)$. E.g. $q_p^1 \xrightarrow{q} \left\{q_{p'}^2\right\}$ would become $q_p^2 \xrightarrow{q} \left\{q_{p'}^3\right\}$. Moreover, transitions involving states in layer ζ are removed. This is because the stack elements in layer ζ will "go out of scope". $\texttt{EnvMove}$ adds a new transition (analogously to a (p_1, a, rew_a, p_2) rule) corresponding to the control state change from p_1 to p_2 effected by the runs over the other stacks between the current round and the next (hence layers 1 and 2 in the definition above). $\texttt{Saturate}_j$ gets by saturation all configurations of stack j that can reach via $\mathcal{R}_j$ the stacks accepted from the layer-1 states of its argument (i.e. saturation using initial states $\left\{ q_p^1 \mid p \in \mathcal{P} \right\}$, which accept stacks from the next round).

The current layer automaton represents a stack across up to ζ rounds. The predecessor operation adds another round on to the front of this representation. A key new insight in our proofs is that if a transition goes to a layer i state, then it represents part of a run where the stack read by the transition is removed in i rounds time. Thus, if we add a transition at layer 0 (were it to exist) that depends on a transition of layer ζ, then the push or copy operation would have a corresponding pop $(\zeta+1)$ scopes away. Scope-bounding forbids this.

The Reachability Graph The reachability graph $\mathcal{G}_{\mathcal{C}}^{p_{\text{out}}} = (\mathcal{V}, \mathcal{E})$ has vertices $\mathcal{V}$ and edges $\mathcal{E}$. Firstly, $\mathcal{V}$ contains some *initial* vertices $(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$ where $p_m = p_{\text{out}}$, and for all $1 \leq i \leq m$ we have that A_i is the layer automaton $\texttt{Saturate}_i(A)$ where for all w, A accepts $\langle p_i, w\rangle$ from $q_{p_i}^1$. Furthermore, we require that there is some w such that $\langle p_{i-1}, w\rangle$ is accepted by A_i from $q_{p_i}^1$. That is, there is a run from $\langle p_{i-1}, w\rangle$ to p_i. Intuitively, initial vertices model the final round of a run to p_{out} with context switches at $p_0, \ldots, p_m$.

The complete set $\mathcal{V}$ is the set of all tuples $(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$ where there is some w such that $\langle p_{i-1}, w\rangle$ is accepted by A_i from state $q^1_{p_{i-1}}$. To ensure finiteness, we can bound A_i to at most N states. The value of N is $\mathcal{O}(2 \uparrow_{n-1} (\ell))$ where ℓ is polynomial in ζ and the size of $\mathcal{C}$. We give a full definition of N and proof in Appendix G.

We have an edge from a vertex $(p_0, A_1, \ldots, A_m, p_m)$ to $(p'_0, A'_1, \ldots, A'_m, p'_m)$ whenever $p_m = p'_0$ and for all i we have $A_i = \mathtt{Predecessor}_i\left(A'_i, q_{p_i}, q_{p'_{i-1}}\right)$. An edge means the two rounds can be concatenated into a run since the control states and stack contents match up.

Lemma 6.1 (Simulation by $\mathcal{G}^{p_{out}}_{\mathcal{C}}$) *Given a scope-bounded CPDS $\mathcal{C}$ and control states p_{in}, p_{out}, there is a run of $\mathcal{C}$ from $\langle p_{in}, w_1, \ldots, w_m\rangle$ to $\langle p_{out}, w'_1, \ldots, w'_m\rangle$ for some $w'_1, \ldots, w'_m$ iff there is a path in $\mathcal{G}^{p_{out}}_{\mathcal{C}}$ to a vertex $(p_0, A_1, \ldots, A_m, p_m)$ with $p_0 = p_{in}$ from an initial vertex where for all i we have $\langle p_{i-1}, w_i\rangle$ accepted from $q^1_{p_i}$ of A_i.*

Global Reachability The $(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$ in $\mathcal{G}^{p_{out}}_{\mathcal{C}}$ reachable from an initial vertex are finite in number. We know by Lemma 6.1 that there is such a vertex accepting all $\langle p_{i-1}, w_i\rangle$ iff $\langle p_0, w_1, \ldots, w_m\rangle$ can reach the target control state. Let χ be the set of tuples $(p_0, A_1, \ldots, A_m)$ for each reachable vertex as above, where A_i is restricted to the initial state $q^1_{p_{i-1}}$. This is a regular solution to the global control state reachability problem.

7 Conclusion

We have shown decidability of global reachability for ordered and scope-bounded collapsible pushdown systems (and phase-bounded in the appendix). This leads to a challenge to find a general framework capturing these systems. Furthermore, we have only shown upper-bound results. Although, in the case of phase-bounded systems, our upper-bound matches that of Seth for CPDSs without collapse [27], we do not know if it is optimal. Obtaining matching lower-bounds is thus an interesting though non-obvious problem. Recently, a more relaxed notion of scope-bounding has been studied [18]. It would be interesting to see if we can extend our results to this notion. We are also interested in developing and implementing algorithms that may perform well in practice.

Acknowledgments Many thanks for initial discussions with Arnaud Carayol and to the referees for their helpful remarks. This work was supported by Fond. Sci. Math. Paris; AMIS [ANR 2010 JCJC 0203 01 AMIS]; FREC [ANR 2010 BLAN 0202 02 FREC]; VAPF (Région IdF); and the Engineering and Physical Sciences Research Council [EP/K009907/1].

References

[1] M. F. Atig. Model-checking of ordered multi-pushdown automata. *Logical Methods in Computer Science*, 8(3), 2012.

[2] M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *Developments in Language Theory*, pages 121–133, 2008.

[3] T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *POPL*, pages 1–3, Portland, Oregon, Jan. 16–18, 2002.

[4] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.

[5] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *SIGPLAN Not.*, 38(1):62–73, 2003.

[6] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. *CONCUR 2005 - Concurrency Theory*, pages 473–487, 2005.

[7] L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

[8] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *ICALP*, pages 165–176, 2012.

[9] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-shore: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24, 2013.

[10] A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, pages 547–561, 2012.

[11] W. Damm. The io- and oi-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.

[12] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY*, volume 9, pages 27–37, 1997.

[13] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.

[14] A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *Proc. 13th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'10), Paphos, Cyprus, Mar. 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010.

[15] V. Kahlon. Reasoning about threads with bounded lock chains. In *CONCUR*, pages 450–465, 2011.

[16] T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, pages 1450–1461, 2005.

[17] N. Kobayashi and A. Igarashi. Model-checking higher-order programs with recursive types. In *ESOP*, pages 431–450, 2013.

[18] S. La Torre and M. Napoli. A temporal logic for multi-threaded programs. In *IFIP TCS*, pages 225–239, 2012.

[19] S. La Torre and G. Parlato. Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In *FSTTCS*, pages 173–184, 2012.

[20] P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294, 2011.

[21] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

[22] L. Ong. Recursion schemes, collapsible pushdown automata and higher-order model checking. In *LATA*, pages 13–41, 2013.

[23] S. Qadeer. The case for context-bounded verification of concurrent programs. In *Proceedings of the 15th international workshop on Model Checking Software*, SPIN '08, pages 3–6, Berlin, Heidelberg, 2008. Springer-Verlag.

[24] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, pages 93–107, 2005.

[25] S. Schwoon. *Model-checking Pushdown Systems.* PhD thesis, Technical University of Munich, 2002.

[26] K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, pages 300–314, 2006.

[27] A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, pages 203–216, 2009.

[28] A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In *CAV*, pages 615–628, 2010.

[29] S. L. Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170, 2007.

[30] S. L. Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, pages 203–218, 2011.

A Undecidability of MSO Over The Naive Encoding of Order-2 Stacks

We show that the naive graph representation of an order-2 stack leads to the undecidability of MSO. By naive graph representation we mean a graph where each node is a configuration on a run of the CPDS, and we have an edge labelled S between c_1 and c_2 if the configurations are neighbouring on the run. We have an further edge labelled 1 if c_2 was obtained by popping a character via pop_1 that was first pushed on to the stack by a $push^k_a$ at node c_1. More formally, we define the *originating configuration* for each character.

Definition A.1 (Originating Configuration) *Given a run as a sequence of configurations $c_1, c_2, \ldots$ we define inductively the originating configuration of each character. The originating configuration of each character in c_1 is 1. Take a transition $c_i \longrightarrow c_{i+1}$ via a rule (p, a, o, p'). If*

1. *$o = copy_k$, then each character copied inherits its originating configuration from the character it is a copy of. All other characters keep the same originating configuration.*
2. *$o = push^k_b$, all characters maintain the same originating configuration except the new b character that has originating configuration i.*
3. *$o = rew_b$, all characters maintain the same originating configuration except the new b character that has the originating configuration of the a character it is replacing.*
4. *$o = noop, pop_k$ or $collapse_k$, all originating configurations are inherited from the previous stack.*

Thus, from a run $c_1, c_2, \ldots$ we define a graph $(\mathcal{V}, \mathcal{E}_1, \mathcal{E}_2)$ with vertices $\mathcal{V} = \{c_1, c_2, \ldots\}$ and edge sets $\mathcal{E}_1$ and $\mathcal{E}_2$, where $\mathcal{E}_1 = \{(c_i, c_{i+1}) \mid 1 \leq i\}$ and $\mathcal{E}_2$ contains all pairs (c_i, c_j) where c_j was obtained by a pop_1 from c_{j-1} and the originating configuration of the character removed is i.

Now, consider the CPDS generating the following run

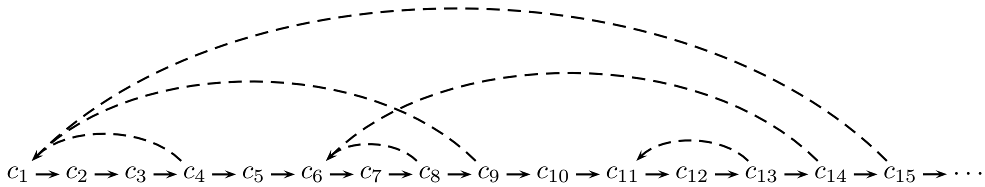
$$\langle p_0, [[\bot]_1]_2\rangle \longrightarrow \langle p_1, [[a\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[a\ \bot]_1[a\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[\bot]_1[a\ \bot]_1]_2\rangle \longrightarrow$$

$$\langle p_0, [[a\ \bot]_1]_2\rangle \longrightarrow \langle p_1, [[aa\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[aa\ \bot]_1[aa\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[a\ \bot]_1[aa\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[\bot]_1[aa\ \bot]_1]_2\rangle \longrightarrow$$

$$\langle p_0, [[aa\ \bot]_1]_2\rangle \longrightarrow \langle p_1, [[aaa\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[aaa\ \bot]_1[aaa\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[aa\ \bot]_1[aaa\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[a\ \bot]_1[aaa\ \bot]_1]_2\rangle \longrightarrow \langle p_2, [[\bot]_1[aaa\ \bot]_1]_2\rangle \longrightarrow$$

$$\langle p_0, [[aaa\ \bot]_1]_2\rangle \longrightarrow \cdots\ .$$

That is, beginning at $\langle p_0, \bot_2\rangle$ the CPDS pushes an a character, copies the stack with a $copy_2$ and removes all as. After all as are removed, it performs pop_2 the obtain the stack below containing only a. It pushes another a onto the stack and repeats this process. After each pop_2 it adds one more a character, performs a $copy_2$, pops all as and so on. This produces the graph shown below with $\mathcal{E}_1$ represented with solid lines, and $\mathcal{E}_2$ with dashed lines. Furthermore, nodes from which an a is pushed are the target of a dashed arrow, and nodes reached by popping an a are the sources of dashed arrows.



In this graph we can interpret the infinite half-grid. We restrict the graph to nodes that are the source of a dashed arrow. We define horizontal and vertical edges to obtain the grid below.

$$\begin{tikzcd}
 & & \vdots & \\
 & & c_{13} \arrow[u] \arrow[r] & \cdots \\
 & c_8 \arrow[r] & c_{14} \arrow[u] \arrow[r] & \cdots \\
c_4 \arrow[r] & c_9 \arrow[u] \arrow[r] & c_{15} \arrow[u] \arrow[r] & \cdots
\end{tikzcd}$$

There is a vertical edge from c to c' whenever $(c', c) \in \mathcal{E}_1$. There is a horizontal edge from c to c' whenever we have c'' such that

1. $(c'', c) \in \mathcal{E}_2$ and $(c'', c') \in \mathcal{E}_2$, and
2. there is a path in $\mathcal{E}_1$ from c to c', and
3. there is no c''' on the above path with $(c'', c''') \in \mathcal{E}_2$.

Thus, we can MSO-interpret the infinite half-grid, and hence MSO is undecidable over this graph.

This naive encoding contains basic matching information about pushes and pops. It remains an interesting open problem to obtain an encoding of CPDS that is amenable to MSO based frameworks that give positive decidability results for concurrent behaviours.

B Definition of The Saturation Function

We first introduce two more short-hand notation for sets of transitions.

The first is a variant on the long-form transitions. E.g. for the run in Section 2 we can write $q_3 \xrightarrow{q_1} (Q_2, Q_3)$ to represent the use of $q_3 \xrightarrow{q_2} Q_3$ and $q_2 \xrightarrow{q_1} Q_2$ as the first two transitions in the run. That is, for a sequence $q \xrightarrow{q_{k-1}} Q_k, q_{k-1} \xrightarrow{q_{k-2}} Q_{k-1}, \ldots, q_{k'} \xrightarrow{q_{k'-1}} Q_{k'}$ in Δ_k to $\Delta_{k'}$ respectively, we write $q \xrightarrow{q_{k'-1}} (Q_{k'}, \ldots, Q_k)$.

The second notation represents sets of long-form transitions. We write $Q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k)$ if there is a set $\{t_1, \ldots, t_\ell\}$ of long-form transitions such that $Q = \{q_1, \ldots, q_\ell\}$ and for all $1 \leq i \leq \ell$ we have $t_i = q_i \xrightarrow[Q^i_{col}]{a} \left(Q^i_1, \ldots, Q^i_k\right)$ and $Q_{col} = \bigcup_{1 \leq i \leq \ell} Q^i_{col} \subseteq \mathbb{Q}_{k'}$ for some k', and for all k', $Q_{k'} = \bigcup_{1 \leq i \leq \ell} Q^i_{k'}$.

Definition B.1 (The Auxiliary Saturation Function Π_r) *For a consuming CPDS rule $r = (p, a, o, p')$ we define for a given stack automaton A, the set $\Pi_r(A)$ to be the smallest set such that, when*

1. *$o = pop_k$, for each $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \dots, Q_n)$ in A, the set $\Pi_r(A)$ contains the transition $q_p \xrightarrow[\emptyset]{a} (\emptyset, \dots, \emptyset, \{q_k\}, Q_{k+1}, \dots, Q_n)$,*

2. *$o = collapse_k$, when $k = n$, the set $\Pi_r(A)$ contains $q_p \xrightarrow[\{q_{p'}\}]{a} (\emptyset, \dots, \emptyset)$, and when $k < n$, for each transition $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \dots, Q_n)$ in A, the set $\Pi_r(A)$ contains the transition $q_p \xrightarrow[\{q_k\}]{a} (\emptyset, \dots, \emptyset, Q_{k+1}, \dots, Q_n)$,*

For a generating CPDS rule $r = (p, a, o, p')$ we define for a given stack automaton A and long-form transition t of A, the set $\Pi_r(t, A)$ to be the smallest set such that, when

1. *$o = copy_k$, $t = q_{p'} \xrightarrow[Q_{col}]{a} (Q_1, \dots, Q_k, \dots, Q_n)$ and $Q_k \xrightarrow[Q'_{col}]{a} (Q'_1, \dots, Q'_k)$ is in A, the set $\Pi_r(t, A)$ contains the transition*
$$q_p \xrightarrow[Q_{col} \cup Q'_{col}]{a} \left(Q_1 \cup Q'_1, \dots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \dots, Q_n\right) ,$$

2. *$o = push^k_b$, for all transitions $t = q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \dots, Q_n)$ and $Q_1 \xrightarrow[Q'_{col}]{a} Q'_1$ is in A with $Q_{col} \subseteq \mathbb{Q}_k$, the set $\Pi_r(t, A)$ contains the transition*
$$q_p \xrightarrow[Q'_{col}]{a} (Q'_1, Q_2, \dots, Q_{k-1}, Q_k \cup Q_{col}, Q_{k+1}, \dots, Q_n) ,$$

3. *$o = rew_b$ or $o = noop$, $t = q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \dots, Q_n)$ the set $\Pi_r(t, A)$ contains the transition $q_p \xrightarrow[Q_{col}]{a} (Q_1, \dots, Q_n)$ (where $b = a$ if $o = noop$).*

As a remark, omitted from the main body of the paper, during saturation, we add transitions $q_n \xrightarrow[Q_{col}]{a} (Q_1, \dots, Q_n)$ to the automaton. Recall this represents a sequence of transitions $q \xrightarrow{q_{k-1}} Q_k \in \Delta_k, q_{k-1} \xrightarrow{q_{k-2}} Q_{k-1} \in \Delta_{k-1}, \dots, q_1 \xrightarrow[Q_{col}]{a} Q_1 \in \Delta_1$. Hence, we first, for each $n \geq k > 1$, add $q_k \xrightarrow{q_{k-1}} Q_k$ to Δ_k if it does not already exist. Then, we add $q_1 \xrightarrow[Q_{col}]{a} Q_1$ to Δ_1. Note, in particular, we only add *at most one* q' with $(q, q', Q) \in \Delta_k$ for all q and Q. This ensures termination.

Also, we say a state is *initial* if it is of the form $q_p \in Q_n$ for some control state p or if it is a state $q_k \in Q_k$ for $k < n$ such that there exists a transition $q_{k+1} \xrightarrow{q_k} Q_{k+1}$ in Δ_{k+1}. A pre-condition (that does not sacrifice generality) of the saturation technique is that there are no incoming transitions to initial states.

C Proofs for Extended CPDS

We provide the proof of Theorem 3.1 (Global Reachability of ECPDS). The proof is via the two lemmas in the sections that follow. A large part of the proof is identical to ICALP 2012 and hence not repeated here.

C.1 Completeness of Saturation for ECPDS

Lemma C.1 (Completeness of Π) *Given an extended CPDS $\mathcal{C}$ and an order-n stack automaton A_0, the automaton A constructed by saturation with Π is such that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ implies $w \in \mathcal{L}_{q_p}(A)$.*

Proof. We begin with a definition of $Pre^*_{\mathcal{C}}(A_0)$ that permits an inductive proof of completeness. Thus, let $Pre^*_{\mathcal{C}}(A_0) = \bigcup_{\alpha<\omega} Pre^{\alpha}_{\mathcal{C}}(A_0)$ where

$$\begin{array}{rcl} Pre^0_{\mathcal{C}}(A_0) & = & \left\{\langle p, w\rangle \;\middle|\; w \in \mathcal{L}_{q_p}(A_0)\right\} \\ Pre^{\alpha+1}_{\mathcal{C}}(A_0) & = & \left\{\langle p, w\rangle \;\middle|\; \exists\langle p, w\rangle \longrightarrow \langle p', w'\rangle \in Pre^{\alpha}_{\mathcal{C}}(A_0)\right\} \end{array}$$

The proof is by induction over α. In the base case, we have $w \in \mathcal{L}_{q_p}(A_0)$ and the existence of a run of A_0, and thus a run in A comes directly from the run of A_0. Now, inductively assume $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ and an accepting run of w' from $q_{p'}$ of A.

There are two cases depending on the rule used in the transition above. Here we consider the case where the rule is of the form $(p, top_1(w), \mathcal{L}_g, p')$. The case where the rule is a standard CPDS rule is identical to ICALP 2012 and hence we do not repeat it here (although a variation of the proof appears in the proof of Lemma G.2).

Take the rule $(p, top_1(w), \mathcal{L}_g, p')$ and the sequence $(p_0, a_1, o_1, p_1)\ldots,(p_{\ell-1}, a_\ell, o_\ell, p_\ell) \in \mathcal{L}_g$ that witnessed the transition, observing that $p_0 = p$ and $p_\ell = p'$. Now, let $w_i = o_\ell(\cdots o_{i+1}(w'))$ for all $0 \leq i \leq \ell$. Note, $w = w_0$ and $w' = w_\ell$.

Take $t' = q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ to be the first transition on the accepting run of $\langle p', w'\rangle$. Beginning with $t_\ell = t'$, we are going to show that there is a run of $\langle p_i, w_i\rangle$ beginning with t_i and thereafter only using transitions appearing in A. Since, by the definition of Π, we add $t_0 = t$ to A, we will obtain an accepting run of A for $\langle p_0, w_0\rangle = \langle p, w\rangle$ as required. We will induct from ℓ down to 0.

The base case $i = \ell$ is trivial, since $t_\ell = t'$ and we already have an accepting run of A over $\langle p_\ell, w_\ell\rangle$ beginning with t_ℓ. Now, assume the case for $\langle p_i, w_i\rangle$ and t_i. We show the case for $i - 1$. Take (p_{i-1}, a_i, o_i, p_i), we do a case split on o_i. A reader familiar with the saturation method for CPDS will observe that the arguments below are very similar to the arguments for ordinary CPDS rules.

1. When $o_i = copy_k$, let $w_{i-1} = u_{k-1} :_k \cdots :_n u_n$. We know

$$w_i = u_{k-1} :_k u_{k-1} :_k u_k :_{(k+1)} \cdots :_n u_n \ .$$

   Let $t_i = q_{p_i} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k, \ldots Q_n)$ and $Q_k \xrightarrow[Q'_{col}]{a} (Q'_1, \ldots, Q'_k)$ be the initial transitions used on the run of w_i (where the transition from Q_k reads the second copy of u_{k-1}).

   From the construction of $\mathcal{T}^A_{t,t'}$ we have have a transition $t_{i-1} \xrightarrow{(p_{i-1},a_i,o_i,p_i)} t_i$ where

$$t_{i-1} = q_{p_{i-1}} \xrightarrow[Q_{col}\cup Q'_{col}]{a} \left(Q_1 \cup Q'_1, \ldots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \ldots, Q_n\right) \ .$$

   Since we know $u_k :_{(k+1)} \cdots :_n u_n$ is accepted from Q'_k via $Q_{k+1}, \ldots, Q_n$, and we know that u_{k-1} is accepted from $Q_1, \ldots, Q_{k-1}$ and $Q'_1, \ldots, Q'_{k-1}$ via a-transitions labelling annotations with Q_{col} and Q'_{col} respectively, we obtain an accepting run of w_{i-1}.

2. When $o_i = push^k_c$, let $w_{i-1} = u_{k-1} :_k u_k :_{k+1} \cdots :_n u_n$. We know $w_i = push^k_c(w_{i-1})$ is

$$c^{u_k} :_1 u_{k-1} :_k \cdots :_n u_n \ .$$

   Let $t_i = q_{p_i} \xrightarrow[Q_{col}]{c} (Q_1, \ldots, Q_n)$ and $Q_1 \xrightarrow[Q'_{col}]{a} Q'_1$ be the first transitions used on the accepting run of w_i. The construction of $\mathcal{T}^A_{t,t'}$ means we have a transition $t_{i-1} \xrightarrow{(p_{i-1}, a_i, o_i, p_i)} t_i$ where $t_{i-1} = q_{p_{i-1}} \xrightarrow[Q'_{col}]{a} (Q'_1, Q_2, \ldots, Q_k \cup Q_{col}, \ldots, Q_n)$. Thus we can construct an accepting run of w_{i-1} (which is w_i without the first c on top of the top order-1 stack). A run from $Q_k \cup Q_{col}$ exists since u_k is also the stack annotating c.

3. When $o_i = rew_c$ let $q_{p_i} \xrightarrow[Q_{col}]{c} (Q_1, \ldots, Q_n)$ be the first transition on the accepting run of $w_i = c^u :_1 v$ for some v and u. From the construction of $\mathcal{T}^A_{t,t'}$ we know we have a transition $t_{i-1} \xrightarrow{(p_{i-1}, a_i, o_i, p_i)} t_i$ where $t_{i-1} = q_{p_{i-1}} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$, from which we get an accepting run of $w_{i-1} = a^u :_1 v$ as required.

4. When $o_i = noop$ let $q_{p_i} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ be the first transition on the accepting run of $w_i = a^u :_1 v$ for some v and u. From the construction of $\mathcal{T}^A_{t,t'}$ we know we have a transition $t_{i-1} \xrightarrow{(p_{i-1}, a_i, o_i, p_i)} t_i$ where $t_{i-1} = q_{p_{i-1}} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$, from which we get an accepting run of $w_{i-1} = a^u :_1 v$ as required.

Hence, for every $\langle p, w \rangle \in Pre^*_{\mathcal{C}}(A_0)$ we have $w \in \mathcal{L}_{q_p}(A)$. $\square$

C.2 Soundness of Saturation for ECPDS

As in the previous section, the soundness argument repeats a large part of the proof given in ICALP 2012. We first recall the machinery used for soundness, before giving the soundness proof.

First, assume all stack automata are such that their initial states are not final. This is assumed for the automaton A_0 in and preserved by the saturation function Γ.

We assign a "meaning" to each state of the automaton. For this, we define what it means for an order-k stack w to satisfy a state $q \in \mathbb{Q}_k$, which is denoted $w \models q$.

Definition C.1 ($w \models q$) *For any $Q \subseteq \mathbb{Q}_k$ and any order-k stack w, we write $w \models Q$ if $w \models q$ for all $q \in Q$, and we define $w \models q$ by a case distinction on q.*

1. *q is an initial state in $\mathbb{Q}_n$. Then for any order-n stack w, we say that $w \models q$ if $\langle q, w \rangle \in Pre^*_{\mathcal{C}}(A_0)$.*

2. *q is an initial state in $\mathbb{Q}_k$, labeling a transition $q_{k+1} \xrightarrow{q} Q_{k+1} \in \Delta_{k+1}$. Then for any order-$k$ stack w, we say that $w \models q$ if for all order-$(k+1)$ stacks s.t. $v \models Q_{k+1}$, then $w :_{(k+1)} v \models q_{k+1}$.*

3. *q is a non-initial state in $\mathbb{Q}_k$. Then for any order-k stack w, we say that $w \models q$ if A_0 accepts w from q.*

By unfolding the definition, we have that an order-k stack w_k satisfies an initial state $q_k \in \mathbb{Q}_k$ with $q \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ if for any order-$(k+1)$ stack $w_{k+1} \models Q_{k+1}, \ldots$, and any order-$n$ stack $w_n \models Q_n$, we have $w_k :_{(k+1)} \cdots :_n w_n \models q$.

Definition C.2 (Soundness of transitions) *A transition $q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k)$ is sound if for any order-1 stack $w_1 \models Q_1, \ldots,$ and any order-k stack $w_k \models Q_k$ and any stack $u \models Q_{col}$, we have $a^u :_1 w_1 :_2 \cdots :_k w_k \models q$.*

The proof of the following lemma can be found in ICALP 2012 [8].

Lemma C.2 ([8]) *If $q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ is sound, then any transition $q_k \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k)$ contained within the transition from q_p is sound.*

Definition C.3 (Soundness of stack automata) *A stack automaton A is sound if the following holds.*

- *A is obtained from A_0 by adding new initial states of order $< n$ and transitions starting in an initial state.*
- *In A, any transition $q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k)$ for $k \leq n$ is sound.*

Unsurprisingly, if some order-n stack w is accepted by a *sound* stack automaton A from a state q_p then $\langle p, w \rangle$ belongs to $Pre^*_{\mathcal{C}}(A_0)$. More generally, we have the following lemma whose proof can be found in ICALP 2012.

Lemma C.3 ([8]) *Let A be a sound stack automaton A and let w be an order-k stack. If A accepts w from a state $q \in \mathbb{Q}_k$ then $w \models q$. In particular, if A accepts an order-n stack w from a state $q_p \in \mathbb{Q}_n$ then $\langle p, w \rangle$ belongs to $Pre^*_{\mathcal{C}}(A_0)$.*

We also recall that the initial automaton A_0 is sound.

Lemma C.4 (Soundness of A_0 [8]) *The automaton A_0 is sound.*

We are now ready to prove that the soundness of saturation for extended CPDS.

Lemma C.5 (Soundness of Π) *The automaton A constructed by saturation with Π and $\mathcal{C}$ from A_0 is sound.*

Proof. The proof is by induction on the number of iterations of Π. The base case is the automaton A_0 and the result was established in Lemma C.4. As in the completeness case, the argument for the ordinary CPDS rules is identical to ICALP 2012 and not repeated here (although the arguments appear in the proof of Lemma G.3).

We argue the case for those transitions added because of extended rules $(p, a, \mathcal{L}_g, p')$.

Hence, we consider the inductive step for transitions introduced by extended rules of the form $(p, c, \mathcal{L}_g, p')$. Take the t, t' and $(p_0, a_1, o_1, p_1)(p_1, a_2, o_2, p_2) \ldots (p_{\ell-1}, a_\ell, o_\ell, p_\ell) \in \mathcal{L}_g \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right)$ with t' being a transition of A_i that led to the introduction of t. Note $p = p_0$ and $p' = p_\ell$.

Let $t_0, \ldots, t_\ell$ be the sequence of states on the accepting run of $\mathcal{T}^{A_i}_{t,t'}$. In particular $t_0 = t$ and $t_\ell = t'$. We will prove by induction from $i = \ell$ to $i = 0$ that for each t_i, letting

$$t_i = q_{p_i} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n) \ ,$$

and for all $u \models Q_{col}$, $w_1 \models Q_1$, $\ldots$, $w_n \models Q_n$ that for $w^i = a^u :_1 w_1 :_2 \cdots :_n w_n$ we have $o_\ell\big(\cdots o_{i+1}\big(w^i\big)\big) \models q_{p'}$. Thus, at $t_0 = t$, we have $o_\ell\big(\cdots o_1\big(w^0\big)\big) \models q_{p'}$ and thus $\langle p', o_\ell\big(\cdots o_1\big(w^0\big)\big)\rangle \in Pre^*_{\mathcal{C}}(A_0)$. Since the above sequence

$$(p_0, a_1, o_1, p_1)\,(p_1, a_2, o_2, p_2) \ldots (p_{\ell-1}, a_\ell, o_\ell, p_\ell)$$

is in $\mathcal{L}_g$, we have $\langle p_0, w^0\rangle \in Pre^*_{\mathcal{C}}(A_0)$ and thus $w^0 \models q_p$, giving soundness of the new transition t_0.

The base case is $t_\ell = t'$. Since t' appears in A_i, we know it is sound. That gives us that $w^\ell \models q_{p'}$ as required.

Now assume that t_i satisfies the hypothesis. We prove that t_{i-1} does also. Take the transition $t_{i-1} \xrightarrow{(p_{i-1}, a_i, o_i, p_i)} t_i$. We perform a case split on o_i. Readers familiar with ICALP 2012 will notice that the arguments here very much follow the soundness proof for ordinary rules.

1. Assume that $o_i = copy_k$, that we had

   $$t_i = q_{p_i} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n) \quad \text{and} \quad Q_k \xrightarrow[Q'_{col}]{a} (Q'_1, \ldots, Q'_k)$$

   where the latter set of transition are in A_i and therefore sound, and that

   $$t_{i-1} = q_{p_{i-1}} \xrightarrow[Q_{col} \cup Q'_{col}]{a} \big(Q_1 \cup Q'_1, \ldots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \ldots, Q_n\big) \ .$$

   To establish the property for this latter transition, we have to prove that for any $w_1 \models Q_1 \cup Q'_1, \ldots$, any $w_{k-1} \models Q_{k-1} \cup Q'_{k-1}$, any $w_k \models Q'_k$, any $w_{k+1} \models Q_{k+1}, \ldots$, any $w_n \models Q_n$ and any $u \models Q_{col} \cup Q'_{col}$, we have for $w^{i-1} = a^u :_1 w_1 :_2 \cdots :_n w_n$ that $o_\ell\big(\cdots o_i\big(w^{i-1}\big)\big) \models q_{p'}$.

   Let $v = top_k\big(w^{i-1}\big) = a^u :_1 w_1 :_2 \cdots :_{(k-1)} w_{k-1}$.

   From the soundness of $Q_k \xrightarrow[Q'_{col}]{a} (Q'_1, \ldots, Q'_k)$ and as $u \models Q'_{col}, w_1 \models Q'_1, \ldots, w_k \models Q'_k$, we have $v :_k w_k \models Q_k$.

   Then, from $w_1 \models Q_1, \ldots, w_{k-1} \models Q_{k-1}$, and $v :_k w_k \models Q_k$, and $w_{k+1} \models Q_{k+1}, \ldots, w_n \models Q_n$ and $u \models Q_{col}$ and the induction hypothesis for $t_i = q_{p_i} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ we get

   $$o_\ell(\cdots o_{i+1}(copy_k(w))) = o_\ell\big(\cdots o_{i+1}\big(v :_k v :_k w_k :_{(k+1)} \cdots :_n w_n\big)\big) \models q_{p'}$$

   as required.

2. Assume that $o_i = push^k_b$, that we have

   $$t_i = q_{p_i} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n) \quad \text{and} \quad Q_1 \xrightarrow[Q'_{col}]{a} (Q'_1)$$

   where the latter set of transitions is sound, and that we have

   $$t_{i-1} = q_{p_{i-1}} \xrightarrow[Q'_{col}]{a} (Q'_1, Q_2, \ldots, Q_k \cup Q_{col}, \ldots, Q_n) \ .$$

   To prove the induction hypothesis for the latter transition, we have to prove that for any $w_1 \models Q'_1$, any $w_2 \models Q_2, \ldots$, any $w_{k-1} \models Q_{k-1}$, any $w_k \models Q_k \cup Q_{col}$, any

$w_{k+1} \models Q_{k+1}, \ldots$, any $w_n \models Q_n$ and any $u \models Q'_{col}$, that we have for $w^{i-1} = a^u :_1 w_1 :_2 \cdots :_n w_n$ that $o_\ell\big(\cdots o_i\big(w^{i-1}\big)\big) \models q_{p'}$.

From the soundness of $Q_1 \xrightarrow[Q'_{col}]{a} (Q'_1)$ and as $u \models Q'_{col}$ and $w_1 \models Q'_1$ we have $a^u :_1 w_1 \models Q_1$.

Then, from $a^u :_1 w_1 \models Q_1, w_2 \models Q_2, \ldots, w_n \models Q_n$, and $top_{k+1}(pop_k(w)) = w_k \models Q_{col}$, and induction for $t_i = q_{p_i} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$, we get

$$o_\ell\big(\cdots o_{i+1}\big(push_b^k\big(w^{i-1}\big)\big)\big) = o_\ell(\cdots o_{i+1}(b^{w_k} :_1 a^u :_1 w_1 :_2 \cdots :_n w_n)) \models q_{p'}$$

as required.

3. Assume that $o = rew_b$, that we have $t_i = q_{p_i} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ and that

$$t_{i-1} = q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n) \ .$$

To prove the hypothesis for this later transition, we have to prove that for any $w_1 \models Q_1, \ldots$, for any $w_n \models Q_n$ and any $u \models Q_{col}$, we have that for $w^{i-1} = a^u :_1 w_1 :_2 \cdots :_n w_n$ we have $o_\ell\big(\cdots o_i\big(w^{i-1}\big)\big) \models q_{p'}$.

From $w_1 \models Q_1, \ldots, w_n \models Q_n$, and $u \models Q_{col}$, and the hypothesis for $t_i = q_{p_i} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$, we get

$$o_\ell\big(\cdots o_{i+1}\big(rew_b\big(w^{i-1}\big)\big)\big) = o_\ell(\cdots o_{i+1}(b^u :_1 w_1 :_2 \cdots :_n w_n)) \models q_{p'}$$

as required.

4. Assume that $o = noop$, that we have $t_i = q_{p_i} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ and that

$$t_{i-1} = q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n) \ .$$

To prove the hypothesis for this later transition, we have to prove that for any $w_1 \models Q_1, \ldots$, for any $w_n \models Q_n$ and any $u \models Q_{col}$, we have that for $w^{i-1} = a^u :_1 w_1 :_2 \cdots :_n w_n$ we have $o_\ell\big(\cdots o_i\big(w^{i-1}\big)\big) \models q_{p'}$.

From $w_1 \models Q_1, \ldots, w_n \models Q_n$, and $u \models Q_{col}$, and the hypothesis for $t_i = q_{p_i} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$, we get

$$o_\ell\big(\cdots o_{i+1}\big(rew_a\big(w^{i-1}\big)\big)\big) = o_\ell(\cdots o_{i+1}(a^u :_1 w_1 :_2 \cdots :_n w_n)) \models q_{p'}$$

as required.

This completes the proof. □

C.3 Complexity of Saturation for ECPDS

We argue that saturation for ECPDS maintains the same complexity as saturation for CPDS.

Proposition C.1 *The saturation construction for an order-n CPDS $\mathcal{C}$ and an order-n stack automaton A_0 runs in n-EXPTIME.*

Proof. The number of states of A is bounded by $2 \uparrow_{(n-1)} (\ell)$ where ℓ is the size of $\mathcal{C}$ and A_0: each state in $\mathbb{Q}_k$ was either in A_0 or comes from a transition in Δ_{k+1}. Since the automata are alternating, there is an exponential blow up at each order except at order-n. Each iteration of the algorithm adds at least one new transition. Only $2 \uparrow_n (\ell)$ transitions can be added. $\square$

The complexity can be reduced by a single exponential when runs of the stack automata are "non-alternating at order-n". In this case an exponential is avoided by only adding a transition $q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ when Q_n contains at most one element.

We refer the reader to ICALP 2012 for a full discussion of non-alternation since it relies on the original notion of collapsible pushdown system that we have not defined here. ICALP 2012 describes the connection between our notion of CPDS (using annotations) and the original notion, as well as defining non-alternation at order-n and arguing completeness for the restricted saturation step. It is straightforward to extend this proof to include ECPDS as in the proof of Lemma C.1 (Completeness of Π) above.

D Definitions and Proofs for Multi-Stack CPDS

D.1 Multi-Stack Collapsible Pushdown Automata

We formally define mutli-stack collapsible pushdown automata.

Definition D.1 (Multi-Stack Collapsible Pushdown Automata) *An order-n* multi-stack collapsible pushdown automaton (n-OCPDA) *over input alphabet Γ is a tuple $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ where $\mathcal{P}$ is a finite set of control states, Σ is a finite stack alphabet, Γ is a finite set of output symbols, and for each $1 \leq i \leq m$ we have a set of rules $\mathcal{R}_i \subseteq \mathcal{P} \times \Sigma \times \Gamma \times \mathcal{O}_n \times \mathcal{P}$.*

Configurations of an OCPDA are defined identically to configurations for OCPDS. We have a transition

$$\langle p, w_1, \ldots, w_m \rangle \xrightarrow{\gamma} \langle p', w_1, \ldots, w_{i-1}, w'_i, w_{i+1}, \ldots, w_m \rangle$$

whenever $r = (p, a, \gamma, o, p') \in \mathcal{R}_i$ with $a = top_1(w)$, $w'_i = o(w_i)$.

D.2 Regular Sets of Configurations

We prove several properties about Definition 4.4 (Regular Set of Configurations).

Property D.1 *Regular sets of configurations of a multi-stack CPDS*

1. *form an effective boolean algebra,*
2. *the emptiness problem is decidable in PSPACE,*
3. *the membership problem is decidable in linear time.*

Proof. We first prove (1). We recall from [8] that stack automata form an effective boolean algebra. Given two regular sets χ_1 and χ_2, we can form $\chi = \chi_1 \cup \chi_2$ as the simple union of the two sets of tuples. We obtain the intersection of χ_1 and χ_2 by defining $\chi = \chi_1 \cap \chi_2$ via a product construction. That is,

$$\chi = \left\{ (p, A_1 \cap A'_1, \ldots, A_m \cap A'_m) \;\middle|\; \begin{array}{c} (p, A_1, \ldots, A_m) \in \chi_1 \wedge \\ (p, A'_1, \ldots, A'_m) \in \chi_2 \end{array} \right\} .$$

It remains to define the complement $\overline{\chi}$ of a set χ. Let $\chi = \chi_1 \cup \cdots \cup \chi_\ell$ where each χ_i is a singleton set of tuples. Observe that $\overline{\chi} = \overline{\chi_1} \cap \cdots \cap \overline{\chi_\ell}$. Hence, we define for a singleton χ_i its complement $\overline{\chi_i}$. Let A be a stack automaton accepting all stacks. Furthermore, let χ_i contain only $(p, A_1, \ldots, A_\ell)$. We define

$$\begin{array}{rcl} \overline{\chi_i} & = & \{ (p', A, \ldots, A) \mid p \neq p' \in \mathcal{P} \} \cup \\ & & \left\{ \left(p, A, \ldots, A, \overline{A_j}, A, \ldots, A\right) \mid 1 \leq j \leq m \right\} . \end{array}$$

That is, either the control state does not match, or at least one of the m stacks does not match.

We now prove (2). We know from [8] that the emptiness problem for a stack automaton is PSPACE. By checking all tuples to find some tuple $(p, A_1, \ldots, A_m)$ such that A_i is non-empty for all i, we have a PSPACE algorithm for determining the emptiness of a regular set χ.

Finally, we show (3), recalling from [8] that the membership problem for stack automata is linear time. To check whether $\langle p, w_1, \ldots, w_m \rangle$ is contained in χ we check each tuple $(p, A_1, \ldots, A_m) \in \chi$ to see if w_i is contained in A_i for all i. This requires linear time. □

E Proofs for Ordered CPDS

E.1 Proofs for Simulation by $\mathcal{C}^R$

We prove Lemma 5.1 ($\mathcal{C}^R$ simulates $\mathcal{C}$) via Lemma E.1 and Lemma E.2 below.

Lemma E.1 *Given an n-OCPDS $\mathcal{C}$ and control states p_{in}, p_{out}, we have*

$$\langle p_{in}, \perp_n, \ldots, \perp_n, w \rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, \perp_n, \ldots, \perp_n \rangle .$$

only if $\langle p_{in}, w \rangle \in Pre^*_{\mathcal{C}^R}(A)$, where A is the $\mathcal{P}$-stack automaton accepting only the configuration $\langle p_{out}, \perp_n \rangle$.

Proof. Take such a run

$$\langle p_{\mathrm{in}}, \perp_n, \ldots, \perp_n, w \rangle \longrightarrow \cdots \longrightarrow \langle p_{\mathrm{out}}, \perp_n, \ldots, \perp_n \rangle$$

of $\mathcal{C}$. Observe that the run can be partitioned into $\tau_0 \sigma_1 \tau_1 \ldots \sigma_\ell \tau_\ell$ where during each τ_i, the first $(m-1)$ stacks are $\perp_n$, and, during each σ_i, there is at least one stack in the first $(m-1)$ stacks that is not $\perp_n$. Let p^1_i be the control state of the first configuration of τ_i, p^2_i be the control state in the final configuration of τ_i, p^3_i be the control state at the beginning of each σ_i, and p^4_i be the control state at the end of each σ_i. Note, $p^4_\ell = p_{\mathrm{out}}$ and $p^1_1 = p_{\mathrm{in}}$. Next, let r_i be the rule fired between the final configuration of τ_{i-1} and the first configuration of σ_i (if it exists). Finally, let w_i be the contents of stack m in the final configuration of each τ_i. Note $w_\ell = w$.

We proceed by backwards induction from $i = \ell$ down to $i = 0$. Trivially it is the case that $\langle p^4_\ell, w_\ell \rangle \in Pre^*_{\mathcal{C}^R}(A)$.

In the inductive step, first assume $\langle p^4_i, w_i \rangle \in Pre^*_{\mathcal{C}^R}(A)$. We have the final configuration of τ_i is $\langle p^4_i, \bot_n, \ldots, \bot_n, w_i \rangle$. Let $\langle p^3_i, \bot_n, \ldots, \bot_n, w' \rangle$ be the first configuration of τ_i. Note, since we assume all rules of the form $(p_1, \bot, o, p_2)$ have $o = push^n_a$ for some a, and during τ_i the first $(m-1)$ stacks are empty, we know that no rule from $\mathcal{R}_1, \ldots, \mathcal{R}_{m-1}$ was used during τ_i. Thus, τ_i is a run of $\mathcal{C}^R$ using only rules from $\mathcal{R}_m$. Hence, we have $\langle p^3_i, w' \rangle \in Pre^*_{\mathcal{C}^R}(A)$.

Now consider σ_i with $\langle p^3_i, \bot_n, \ldots, \bot_n, w' \rangle$ appended to the end. Suppose we have that $r_{i-1} = \left(p^4_{i-1}, \bot, push^n_b, p^1_i\right) \in \mathcal{R}_j$. We thus have a run

$$\langle p^1_i, w'_1, \ldots, w'_{m-1}, w_{i-1} \rangle \xrightarrow{r^1} \cdots \xrightarrow{r^{\ell-1}} \langle p^2_i, w''_1, \ldots, w''_m \rangle \xrightarrow{r^\ell} \langle p^3_i, \bot_n, \ldots, \bot_n, w' \rangle$$

where $w'_j = push^n_b(\bot_n)$ and $w'_{j'} = \bot_n$ for all $j' \neq j$. Since it is not the case that the first $(m-1)$ stacks are empty, we know that only generating rules from $\mathcal{R}_m$ can be used during this run. Let $top_1(w_{i-1}) = a$. From this run we can immediately project a sequence $\left(p^0, a^1, o^1, p^1\right)\left(p^1, a^2, o^2, p^2\right) \ldots \left(p^{\ell'-1}, a^{\ell}, o^{\ell'}, p^{\ell'}\right) \in \mathcal{L}^{b,j}_{p^1_i, a, p^3_i}\left(\mathcal{C}^L\right)$ such that we have $w' = o^{\ell'}\left(\cdots o^1(w_{i-1})\right)$, $p^0 = p^1_i$ and $p^{\ell'} = p^3_i$. Since we have $\langle p^3_i, w' \rangle \in Pre^*_{\mathcal{C}^R}(A)$ and a rule $\left(p^4_{i-1}, a, \mathcal{L}^{b,j}_{p^1_i, a, p^3_i}\left(\mathcal{C}^L\right), p^3_i\right)$ in $\mathcal{C}^R$, we thus have $\langle p^4_{i-1}, w_{i-1} \rangle \in Pre^*_{\mathcal{C}^R}(A)$ as required.

Hence, when $i = 0$, we have $\langle p_{\text{in}}, w \rangle \in Pre^*_{\mathcal{C}^R}(A)$, completing the proof. □

Lemma E.2 *Given an n-OCPDS $\mathcal{C}$ and control states p_{in}, p_{out}, we have*

$$\langle p_{in}, \bot_n, \ldots, \bot_n, w \rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, \bot_n, \ldots, \bot_n \rangle \ .$$

whenever $\langle p_{in}, w \rangle \in Pre^*_{\mathcal{C}^R}(A)$, where A is the $\mathcal{P}$-stack automaton accepting only the configuration $\langle p_{out}, \bot_n \rangle$.

Proof. Since $\langle p_{\text{in}}, w \rangle \in Pre^*_{\mathcal{C}^R}(A)$ we have a run of $\mathcal{C}^R$ of the form $\sigma_1 \ldots \sigma_\ell$ where the rules used to connect the last configuration of σ_i to σ_{i+1} are of the form $(p'_i, a, \mathcal{L}_g, p_{i+1})$ and no other rules of this form are used otherwise. Thus, let p'_i denote the control state at the end of σ_i and p_i denote the control state in the first configuration of σ_i. Similarly, let w'_i denote the stack contents at the end of σ_i and w_i the stack contents at the beginning.

We proceed by induction from $i = \ell$ down to $i = 1$. In the base case, we immediately have a run from $\langle p_\ell, \bot_n, \ldots, \bot_n, w_\ell \rangle$ to $\langle p'_\ell, \bot_n, \ldots, \bot_n \rangle$. Now, assume the we have a run from $\langle p'_i, \bot_n, \ldots, \bot_n, w'_i \rangle$ to the final configuration. Since we have a run to this configuration from $\langle p_i, w_i \rangle$ to $\langle p'_i, w'_i \rangle$ in $\mathcal{C}^R$ that uses only ordinary rules, we can execute the same run from $\langle p_i, \bot_n, \ldots, \bot_n, w_i \rangle$ to reach $\langle p'_i, \bot_n, \ldots, \bot_n, w'_i \rangle$.

Now consider the rule $\left(p'_{i-1}, a, \mathcal{L}_g, p_i\right)$ that connects σ_{i-1} and σ_i. We have $\mathcal{L}_g = \mathcal{L}^{b,j}_{p^1_i, a, p_i}\left(\mathcal{C}^L\right)$ for some p^1_i, b and j, and there is a rule $\left(p'_{i-1}, \bot, push^n_b, p^1_i\right) \in \mathcal{R}_j$ of $\mathcal{C}$. Furthermore, there is a sequence $\left(p^0, a^1, o^1, p^1\right)\left(p^1, a^2, o^2, p^2\right) \ldots \left(p^{\ell'-1}, a^\ell, o^{\ell'}, p^{\ell'}\right) \in \mathcal{L}_g$ such that $w_i = o^{\ell'}\left(\cdots o^1\left(w'_{i-1}\right)\right)$, $p^0 = p^1_i$, and $p^{\ell'} = p_i$.

From the definition of $\mathcal{C}^L$, this sequence immediately describes a run

$$\begin{array}{lcl} \langle p'_{i-1}, \bot_n, \ldots, \bot_n, w'_{i-1} \rangle & \longrightarrow & \langle p^1_i, \bot_n, \ldots, push^n_b(\bot_n), \ldots, \bot_n, w'_{i-1} \rangle \\ & \longrightarrow & \cdots \\ & \longrightarrow & \langle p_i, \bot_n, \ldots, \bot_n, w_i \rangle \end{array}$$

of $\mathcal{C}$. Thus we have a run from $\langle p'_{i-1}, \bot_n, \ldots, \bot_n, w'_{i-1} \rangle$ to the final configuration, to complete the inductive case.

Finally, when $i = 1$, we repeat the first half of the argument above to obtain a run from $\langle p_1, \bot_n, \ldots, \bot_n, w_1 \rangle$, and since $p_1 = p_{\text{in}}$ and $w_1 = w$ we have a run of $\mathcal{C}$ as required. $\square$

E.2 Proofs for Language Emptiness for OCPDS

We prove Lemma 5.2 (Language Emptiness for OCPDS) below.

Proof. By standard product construction arguments, a run of $\mathcal{C}_\emptyset$ can be projected into runs of $\mathcal{C}^L$ and $\mathcal{T}^{A_i}_{t,t'}$ and vice-versa. We need only note that in any control state (p, t_1) of $\mathcal{C}_\emptyset$, the corresponding state in $\mathcal{C}^L$ is always $(p, top_1(t_1))$. $\square$

E.3 Global Reachability

We provide an inductive proof of global reachability for ordered CPDS.

Proof. Take $A_m = Pre^*_{\mathcal{C}^R}(A)$ from Lemma 5.1 ($\mathcal{C}^R$ simulates $\mathcal{C}$). Furthermore, let $A_\bot$ be the stack automaton accepting only $\bot_n$ from its initial state. For each control state p, we have that $(p, A_\bot, \ldots, A_\bot, A_m)$ represents all configurations $\langle p, \bot_n, \ldots, \bot_n, w_m \rangle$ for which there is a run to $\langle p_{\text{out}}, \bot_n, \ldots, \bot_n \rangle$ when A_m is restricted to have initial state q_p.

Hence, inductively assume for $i+1$ that we have a finite set of tuples χ such that for each configuration $\langle p, \bot_n, \ldots, \bot_n, w_{i+1}, \ldots, w_m \rangle$ for which there is a run to $\langle p_{\text{out}}, \bot_n, \ldots, \bot_n \rangle$ there is a tuple $(p, A_\bot, \ldots, A_\bot, A_{i+1}, \ldots, A_m)$ such that w_j is accepted by A_j for each j.

Now consider any configuration $\langle p, \bot_n, \ldots, \bot_n, w_i, \ldots, w_m \rangle$ that can reach the final configuration. We know the run goes via some $\langle p', \bot_n, \ldots, \bot_n, w'_{i+1}, \ldots, w'_m \rangle$ accepted by some tuple $(p', A_\bot, \ldots, A_\bot, A_{i+1}, \ldots, A_m) \in \chi$. Furthermore, we know from the proof of correctness of the extended saturation algorithm, that there is a run of the i stack OCPDS $\mathcal{C}_\emptyset$ from $\langle (p, t_{i+1}, \ldots, t_m), \bot_n, \ldots, \bot_n, w_i \rangle$ to $\langle (p', t'_{i+1}, \ldots, t'_m), \bot_n, \ldots, \bot_n \rangle$ where

1. t'_j is the initial transition of A_j accepting w'_j, and
2. the sequence of stack operations to the jth stack $o_1, \ldots, o_\ell$ connected to this run give $w'_j = o_\ell(\cdots o_1(w_j))$, and
3. w_j can be accepted by first taking transition t_j and thereafter only transitions in A_j.

Thus, let A_i be $Pre^*_{\mathcal{C}_\emptyset}(A)$ where A accepts $\langle (p', t'_{i+1}, \ldots, t'_m), \bot_n \rangle$. Restrict A_i to have initial state $q_{(p, t_{i+1}, \ldots, t_m)}$ and let $A_j^{t_j}$ be the automaton A_j with the transition t_j added from a new state, which is designated as the initial state. Thus, for each configuration $\langle p, \bot_n, \ldots, \bot_n, w_i, \ldots, w_m \rangle$, there is a tuple $\left(p, A_\bot, \ldots, A_\bot, A_i, A_{i+1}^{t_{i+1}}, \ldots, A_m^{t_m}\right)$ such that w_i is accepted by A_i and w_j is accepted by $A_j^{t_j}$ for all $j > i$. This results in a finite set of tuples χ' satisfying the induction hypothesis.

Thus, after $i = 1$ we obtain a finite set of tuples χ of the form $(p, A_1, \ldots, A_m)$ representing all configurations that can reach $\langle p_{\text{out}}, \bot_n, \ldots, \bot_n \rangle$, as required. $\square$

E.4 Complexity

Assume $n > 1$. Our control state reachability algorithm requires $2 \uparrow_{m(n-1)} (\ell)$ time, where ℓ is polynomial in the size of the OCPDS. Beginning with stack m, the saturation algorithm can add at most $\mathcal{O}(2 \uparrow_{n-1} (\ell))$ transitions over the same number of iterations. Each of these iterations may require analysis of some $\mathcal{C}_\emptyset$ which has $\mathcal{O}(2 \uparrow_{n-1} (\ell))$ control states and thus the stack-automaton constructed by saturation over $\mathcal{C}_\emptyset$ may have up to $\mathcal{O}\big(2\uparrow_{2(n-1)}(\ell)\big)$ transitions. By continuing in this way, we have at most $\mathcal{O}\big(2\uparrow_{(m-1)(n-1)}(\ell)\big)$ control states when there is only one stack remaining, and thus the number of transitions, and the total running time of the algorithm is $\mathcal{O}\big(2\uparrow_{m(n-1)}(\ell)\big)$. This also gives us at most $\mathcal{O}(2\uparrow_{mn}(\ell))$ tuples in the solution to the global reachability problem.

F Phase-Bounded CPDS

Phase-bounding [29] for multi-stack pushdown systems is a restriction where each computation can be split into a fixed number of phases. During each phase, characters can only be removed from one stack, but push actions may occur on any stack.

Definition F.1 (Phase-Bounded CPDS) *Given a fixed number ζ of phases, an order-n* phase-bounded CPDS *(n-PBCPDS) is an n-MCPDS with the restriction that each run σ can be partitioned into $\sigma_1 \dots \sigma_\zeta$ and for all i, if some transition in σ_i by $r \in \mathcal{R}_j$ on stack j for some j is consuming, then all consuming transitions in σ_i are by some $r' \in \mathcal{R}_j$ on stack j.*

We give a direct[1] algorithm for deciding the reachability problem over phase-bounded CPDSs. We remark that Seth [28] presented a saturation technique for order-1 phase-bounded pushdown systems. Our algorithm was developed independently of Seth's, but our product construction can be compared with Seth's automaton T_i.

Theorem F.1 (Decidability of the Reachability Problems) *For n-PBCPDSs the control state reachability problem and the global control state reachability problem are decidable.*

In Appendix F.3 we show that our control state reachability algorithm will require $\mathcal{O}\big(2\uparrow_{m(n-1)}(\ell)\big)$ time, where ℓ is polynomial in the size of the PBCPDS, and we have at most $\mathcal{O}(2\uparrow_{mn}(\ell))$ tuples in the solution to the global reachability problem.

Control State Reachability A run of the PBCPDS will be $\sigma_1 \dots \sigma_\zeta$, assuming (w.l.o.g.) that all phases are used. We can guess (or enumerate) the sequence $p_0 p_1 \dots p_\zeta$ of control states occurring at the boundaries of each σ_i. That is, σ_i ends with control state p_i, p_ζ is the target control state, and p_0 is the initial control state. We also guess for each i, the stack ι_i that may perform consuming operations between p_{i-1} and p_i. Our algorithm iterates from $i = \zeta$ down to $i = 0$.

We begin with the stack automata $A^1_\zeta, \dots, A^m_\zeta$ which each accept $\langle p_\zeta, w\rangle$ for all stacks w. Note we can vary these automata to accept any regular set of stacks we wish.

Thus, $A^1_i, \dots, A^m_i$ will characterise a possible set of stack contents at the end of phase i. We show below how to construct $A^1_{i-1}, \dots, A^m_{i-1}$ given $A^1_i, \dots, A^m_i$. This is repeated until we have $A^1_0, \dots, A^m_0$. We then check, for each j, that $\langle p_0, \bot_n\rangle$ is accepted by A^j_0. This is the case iff we have a positive instance of the reachability problem.

We construct $A^1_{i-1}, \dots, A^m_{i-1}$ from $A^1_i, \dots, A^m_i$. For each $j \neq \iota_i$ we build A^j_{i-1} by adding to A^j_i a brand new set of initial states q_p and a guessed transition $t_j = q_{p_{i-1}} \xrightarrow[Q_{col}]{a} (Q_1, \dots, Q_n)$ with $Q_{col}, Q_1, \dots, Q_n$ being states of A^j_i and $q_{p_{i-1}}$ being one of the new states. The idea is t_j will be the initial transition accepting $\langle p_{i-1}, w\rangle$ where w is stack j at the

[1] For PDS, phase-bounded reachability can be reduced to ordered PDS. We do not know if this holds for CPDS, and prefer instead to give a direct algorithm.

beginning of phase i. By guessing an accompanying t'_j of A_i^j we can build $\mathcal{T}^{A_i^j}_{t_j,t'_j}$ (by instantiating Definition 3.2 (Transition Automata) with $A = A_i^j$, $t = t_j$ and $t' = t'_j$) for which there will be an accepting run if the updates to stack j during phase i are concordant with the introduction of transition t_j.

Thus, for each $j \neq \iota_i$ we have A_{i-1}^j and $\mathcal{T}^{A_i^j}_{t_j,t'_j}$. We now consider the ι_ith stack. We build a CPDS $\mathcal{C}_i$ that accurately models stack ι_i and tracks each $\mathcal{T}^{A_i^j}_{t_j,t'_j}$ in its control state. We ensure that $\mathcal{C}_i$ has a run from $\langle p_{i-1}, w\rangle$ to $\langle p_i, w'\rangle$ for some w and w' iff there is a corresponding run over the ι_ith stack of $\mathcal{C}$ that updates the remaining stacks j in concordance with each guessed t_j. Thus, we define $A_{i-1}^{\iota_i}$ to be the automaton recognising $Pre^*_{\mathcal{C}_i}(A_i^{\iota_i})$ constructed by saturation. The construction of $\mathcal{C}_i$ (given below) follows the standard product construction of a CPDS with several finite-state automata.

Note $\mathcal{C}_i$ is looking for a run from p_{i-1} to p_i concordant with runs of t_j to t'_j for each j. To let $\mathcal{C}_i$ start in p_{i-1} and finish in p_i, we have an initial transition from p_{i-1} to $(p_{i-1}, t_1, \ldots, t_m)$. Thereafter, the components are updated as in a standard product construction. When $(p_i, t'_1, \ldots, t'_m)$ is reached, there is a final transition to p_i. To ease notation, we use dummy variables $t_{\iota_i} = t'_{\iota_i} = t^{\iota_i} = t_1^{\iota_i}$ for the transition automaton component of the ι_ith stack (for which we do not have a t and t' to track).

In the definition below, the first line of the definition of $\mathcal{R}^i$ gives the initial and final transitions, the second line models rules operating on stack ι_i, and the final line models generating operations occurring on the jth stack for $j \neq \iota_i$.

Definition F.2 **($\mathcal{C}_i$)** *Given for all $1 \leq j \neq \iota_i \leq m$ a transition automaton $\mathcal{T}_j = \mathcal{T}^{A_i^j}_{t_j,t'_j}$ and a phase-bounded CPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ and control states p_{i-1}, p_i, we define the CPDS $\mathcal{C}_i = \left(\{p_{i-1}, p_i\} \cup \mathcal{P}^i, \mathcal{R}^i, \Sigma\right)$ where, letting $t_{\iota_i} = t'_{\iota_i} = t^{\iota_i} = t_1^{\iota_i}$ be dummy transitions for technical convenience, and letting t^j for all $j \neq \iota_i$ range over all states of $\mathcal{T}_j$, we have*

- *$\mathcal{P}^i$ contains all states $\left(p, t^1, \ldots, t^m\right)$ where $p \in \mathcal{P}$, and*
- *the rules $\mathcal{R}^i$ of $\mathcal{C}_i$ are*

$$
\begin{array}{l}
\{(p_{i-1}, a, noop, (p_{i-1}, t_1, \ldots, t_m)), ((p_i, t'_1, \ldots, t'_m), a, noop, p_i) \mid a \in \Sigma\} \cup \\
\left\{ \left(\left(p, t^1, \ldots, t^m\right), a, o, \left(p', t_1^1, \ldots, t_1^m\right)\right) \;\middle|\; \begin{array}{c} (p, a, o, p') \in \mathcal{R}_{\iota_i} \\ \forall j' \neq j \;.\; t^{j'} \xrightarrow{(p,\_,noop,p')} t_1^{j'} \end{array} \right\} \cup \\
\left\{ (p_1, a, o, p_2) \;\middle|\; \begin{array}{c} p_1 = \left(p, t^1, \ldots, t^j, \ldots t^m\right) \wedge p_2 = \left(p', t_1^1, \ldots, t_1^j, \ldots t_1^m\right) \\ \wedge\;\; (p, b, o, p') \in \mathcal{R}_j \wedge t^j \xrightarrow{(p,b,o,p')} t_1^j \;\wedge \\ \forall j' \neq j \;.\; t^{j'} \xrightarrow{(p,\_,noop,p')} t_1^{j'} \end{array} \right\} .
\end{array}
$$

We state the correctness of our reduction, deferring the proof to Appendix F.2.

Lemma F.1 (Simulation of a PBCPDS) *Given a phase-bounded CPDS $\mathcal{C}$ control states p_0 and p_ζ, there is a run of $\mathcal{C}$ from $\langle p_0, w_1, \ldots, w_m\rangle$ to $\langle p_\zeta, w'_1, \ldots, w'_m\rangle$ iff for each $1 \leq j \leq m$, we have that $\langle p_0, w_j\rangle$ is accepted by A_0^j.*

F.1 Global Reachability

$A_0^1, \ldots, A_0^m$ were obtained by a finite sequence of non-deterministic choices ranging over a finite number of values. Let χ be the therefore finite set of tuples $(p_0, A_1, \ldots, A_m)$ for each sequence as above, where A_i is A_0^i with initial state q_{p_0}. From Lemma F.1, we have a regular solution to the global control state reachability problem as required.

F.2 Proofs for Control-State Reachability

In this section we prove Lemma F.1 (Simulation of a PBCPDS) via Lemma F.2 and Lemma F.3 below.

Lemma F.2 *Given a phase-bounded CPDS $\mathcal{C}$ control states p_0 and p_ζ, there is a run of $\mathcal{C}$ from $\langle p_0, w_1, \ldots, w_m \rangle$ to $\langle p_\zeta, w'_1, \ldots, w'_m \rangle$ only if for each $1 \leq j \leq m$, we have that $\langle p_0, w_j \rangle$ is accepted by A_0^j.*

Proof. Take a run of $\mathcal{C}$ from $\langle p_0, w_0^1, \ldots, w_0^m \rangle$ to $\langle p_\zeta, w_\zeta^1, \ldots, w_\zeta^m \rangle$ and split it into phases $\sigma_1 \ldots \sigma_\zeta$. Let p_i be the control state at the end of each σ_i, and p_0 be the control state at the beginning of σ_1. Similarly, let w_i^j be the stack contents of stack j at the end of σ_i. We include, for convenience, the transition from the end of σ_i to the beginning of σ_{i+1} in σ_{i+1}. Thus, the last configuration of σ_i is also the first configuration of σ_{i+1}.

We proceed by induction from $i = \zeta$ down to $i = 1$. In the base case we know by definition that $\langle p_\zeta, w_\zeta^j \rangle$ is accepted by A_ζ^j.

Hence, assume $\langle p_{i+1}, w_{i+1}^j \rangle$ is accepted by A_{i+1}^j. We show the case for i. First consider ι_i. Take the run

$$\langle p_i, w_i^1, \ldots, w_i^m \rangle \longrightarrow \cdots \longrightarrow \langle p_{i+1}, w_{i+1}^1, \ldots, w_{i+1}^m \rangle \ .$$

We want to find a run

$$\langle p_i, w_i^{\iota_i} \rangle \longrightarrow \langle (p_i, t_1, \ldots, t_m), w_i^{\iota_i} \rangle \longrightarrow \cdots \longrightarrow \langle (p_{i+1}, t'_1, \ldots, t'_m), w_{i+1}^{\iota_i} \rangle \longrightarrow \langle p_1, w_{i+1}^{\iota_i} \rangle$$

of $\mathcal{C}_i$, giving us that $\langle p_i, w_i^{\iota_i} \rangle$ is accepted by $A_i^{\iota_i}$. This is almost by definition, except we need to prove for each $j \neq \iota_i$ that there is a sequence $t^0, \ldots, t^\ell$ that is also the projection of the run of $\mathcal{C}_i$ to the $(j+1)$th component (that is, the state of the jth transition automaton). In particular, we require $t^0 = t_j$ and $t^\ell = t'_j$. The proof proceeds in exactly the same manner as the case of $(p, a, \mathcal{L}_g, p')$ in the proof of Lemma C.1 (Completeness of Π) for ECPDS. Namely, from the sequence of operations $o^0, \ldots, o^\ell$ taken from the run $t^0, \ldots, t^\ell$, we obtain a sequence of stacks such that at each z there is an accepting run of the zth stack constructed from t^z and thereafter only transitions of A_{i+1}^j. Thus, since t_j is added to A_{i+1}^j to obtain A_i^j, we additionally get an accepting run of A_i^j over $\langle p_i, w_i^j \rangle$. We do not repeat the arguments here.

Finally, then, when i reaches 1, we repeat the arguments above to conclude $\langle p_0, w_0^j \rangle$ is accepted by A_0^j for each j, giving the required lemma. $\square$

Lemma F.3 *Given a phase-bounded CPDS $\mathcal{C}$ control states p_0 and p_ζ, there is a run of $\mathcal{C}$ from $\langle p_0, w_1, \ldots, w_m \rangle$ to $\langle p_\zeta, w'_1, \ldots, w'_m \rangle$ whenever for each $1 \leq j \leq m$, we have that $\langle p_0, w_j \rangle$ is accepted by A_0^j.*

Proof. Assume for each $1 \leq j \leq m$, we have that $\langle p_0, w_j \rangle$ is accepted by A_0^j.

Thus, we can inductively assume for each j we have $\langle p_i, w_i^j \rangle$ accepted by A_i^j and a run of $\mathcal{C}$ of the form

$$\langle p_0, w_1, \ldots, w_m \rangle \longrightarrow \cdots \longrightarrow \langle p_i, w_i^1, \ldots, w_i^m \rangle \ .$$

Taking $w_0^j = w_j$ trivially gives us the base case. We prove the case for $(i+1)$.

From the induction hypothesis, we have in particular that $\langle p_i, w_i^{\iota_i}\rangle$ is accepted by $A_i^{\iota_i}$ and hence we have a run of $\mathcal{C}_{i+1}$ of the form

$$\langle p_i, w_i^{\iota_i}\rangle \longrightarrow \langle (p_i, t_1, \ldots, t_m), w_i^{\iota_i}\rangle \longrightarrow \cdots \longrightarrow \langle (p_{i+1}, t'_1, \ldots, t'_m), w_{i+1}^{\iota_i}\rangle \longrightarrow \langle p_1, w_{i+1}^{\iota_i}\rangle$$

such that $\langle p_1, w_{i+1}^{\iota_i}\rangle$ is accepted by $A_{i+1}^{\iota_i}$. From this run, due to the definition of $\mathcal{C}_i$ we can build a run

$$\langle p_i, w_i^1, \ldots, w_i^m\rangle \longrightarrow \cdots \longrightarrow \langle p_{i+1}, w_{i+1}^1, \ldots, w_{i+1}^m\rangle$$

of $\mathcal{C}$ where for all $j \neq \iota_i$, we define $w_{i+1}^j = o^\ell\left(\cdots o^1\left(w_i^j\right)\right)$ where

$$\left(p^0, a^1, o^1, p^1\right)\left(p^1, a^2, o^2, p^2\right)\ldots\left(p^{\ell-1}, a^\ell, o^\ell, p^\ell\right)$$

is the sequence of labels on the run of $\mathcal{T}_{t_j, t'_j}^{A_i^j}$. We have to prove for all $j \neq \iota_i$ that $\langle p_{i+1}, w_{i+1}^j\rangle$ is accepted by A_{i+1}^j. For the proof observe that the introduction of t_j to A_{i+1}^j to form A_i^j followed the saturation technique for extended CPDS for a rule $(p_i, a, \mathcal{L}_g, p_{i+1})$ where $\mathcal{L}_g$ is the language of possible sequences of the form above. Thus, from the soundness of the saturation method for extended CPDS, we have that there must be the required run of A_{i+1}^j over $\langle p_{i+1}, w_{i+1}^j\rangle$ beginning with transition t'_j.

Alternatively, we can argue similarly to the proof of Lemma C.1 (Completeness of Π), but in the reverse direction. That is, we start with the observation that the accepting run of $\langle p_i, w_i^j\rangle$ uses $t_j = t^0$ for the first transition, and thereafter only transitions from A_{i+1}^j. We prove this by induction for the stack obtained by applying o^1 and t^1, then for the stack obtained by applying o^2 and t^2. This continues until we reach w_{i+1}^j, and since $t^\ell = t'_j$ with t'_j being a transition of A_{i+1}^j, we get the accepting run we need. We remark that this is how the soundness proof for the standard saturation algorithm would proceed if we were able to assume that each new transition is only used at the head of any new runs the transition introduces (but in general this is not the case because new transitions may introduce loops). We leave the construction of this proof as an exercise for the interested reader, for which they may follow the proof of the extended rule case for Lemma C.5 (Soundness of Π).

Thus, finally, by induction, we obtain a run to $\langle p_\zeta, w_1, \ldots, w_m\rangle$ such that $\langle p_\zeta, w_j\rangle$ is accepted by A_ζ^j. $\square$

F.3 Complexity

Assume $n > 1$. Our control state reachability algorithm requires $2\uparrow_{\zeta(n-1)}(\ell)$ time, where ℓ is polynomial in the size of the PBCPDS. Beginning with phase ζ, the saturation algorithm can add at most $\mathcal{O}(2\uparrow_{n-1}(\ell))$ transitions over the same number of iterations to $A_{\zeta-1}^{\iota_\zeta}$. Thus we assume each A_i^j to have at most $\mathcal{O}\big(2\uparrow_{(\zeta-i)(n-1)}(\ell)\big)$ transitions. The largest automaton A_{i-1}^j construction is when $j = \iota_i$. For this we build a CPDS with $\mathcal{O}\big(2\uparrow_{(\zeta-i)(n-1)}(\ell)\big)$ control states and thus $A_{i-1}^{\iota_i}$ has at most $\mathcal{O}\big(2\uparrow_{(\zeta-i+1)(n-1)}(\ell)\big)$ transitions. Hence, when $i = 0$, we have at most $\mathcal{O}\big(2\uparrow_{\zeta(n-1)}(\ell)\big)$ transitions, which also gives the run time of the algorithm. This also implies we have at most $\mathcal{O}(2\uparrow_{\zeta n}(\ell))$ tuples in the solution to the global reachability problem.

G Proofs for Scope-Bounded CPDS

G.1 Operations on Layer Automata

Shift of a Layer Automaton The idea behind `Shift` is that all transitions in layer i are moved up to layer $(i+1)$ and transitions involving states in layer ζ are removed. Intuitively this is because the stack elements in layer ζ will "go out of scope" when the context switch corresponding to the `Shift` occurs. In more detail, states of layer i are renamed to become states of layer $(i+1)$, with all states of layer ζ being deleted. Similarly, all transitions that involved a layer ζ state are also removed.

We define $\texttt{Shift}(A)$ of an order-n ζ-layer stack automaton

$$A = (\mathbb{Q}_n, \ldots, \mathbb{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \emptyset, \ldots, \emptyset)$$

to be

$$A' = (\mathbb{Q}'_n, \ldots, \mathbb{Q}'_1, \Sigma, \Delta'_n, \ldots, \Delta'_1, \emptyset, \ldots, \emptyset)$$

where defining

$$\texttt{Shift}(q) = \begin{cases} q & \text{if } q \in \mathbb{Q}_k,\ n > k \text{ and } q \text{ is layer } i < \zeta \\ q_p^{i+1} & \text{if } q = q_p^i \in \mathbb{Q}_n \text{ and } i < \zeta \\ \text{undefined} & \text{otherwise} \end{cases}$$

and extending `Shift` point-wise to sets of states, we have

$$\Delta'_n = \left\{ \texttt{Shift}(q) \xrightarrow{q'} \texttt{Shift}(Q) \;\middle|\; q \xrightarrow{q'} Q \in \Delta_n \text{ and } q \text{ is layer } i < \zeta \right\}$$

and for all $n > k > 1$

$$\Delta'_k = \left\{ q \xrightarrow{q'} \texttt{Shift}(Q) \;\middle|\; q \xrightarrow{q'} Q \in \Delta_k \text{ and } q \text{ is layer } i < \zeta \right\}$$

and

$$\Delta'_1 = \left\{ q \xrightarrow[\texttt{Shift}(Q_{col})]{q'} \texttt{Shift}(Q) \;\middle|\; q \xrightarrow[Q_{col}]{q'} Q \in \Delta_1 \text{ and } q \text{ is layer } i < \zeta \right\} .$$

In all cases above, transitions are only created if the applications of `Shift` result in a defined state or set of states. This operation will erase all layer ζ states, and all transitions that go to a layer ζ state. All other states will be shifted up one layer. E.g. layer 1 states become layer 2.

Environment Moves Given an automaton A, define $\texttt{EnvMove}(A, q, q')$ of an order-n ζ-layer stack automaton to be A' obtained from A by adding for each transition $q' \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ the transition $q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$. This operation can be thought of as a saturation rule that captures the effect of an external context, and could be considered as rules $(p, a, noop, p')$ for each $a \in \Sigma$.

Saturating a Layer Automaton Given a layer automaton A, we define $\texttt{Saturate}_j(A)$ to be the result of applying the saturation procedure with the CPDS $(\mathcal{P}, \Sigma, \mathcal{R}_j)$ and the stack automaton A with initial state-set $\left\{ q_p^1 \mid p \in \mathcal{P} \right\}$.

G.2 Size of the Reachability Graph

We define N.

Lemma G.1 *The maximum number of states in any layer automaton constructable by repeated applications of* $\texttt{Predecessor}_j$ *is* $2 \uparrow_{n-2} (f(\zeta, |\mathcal{P}|))$ *states for some computable polynomial* f.

Proof. A ζ-layer automaton may have in q_n only the states q^i_p for $1 \leq i \leq \zeta$ and $p \in \mathcal{P}$, and thus at most $\zeta\,|\mathcal{P}| = d$ states. There may be at most d transitions from any state at order-n using the restricted saturation algorithm where Q_n has cardinality 1 for any transition added, and thus at most $d \cdot d$ states at order-$(n-1)$ (noting that the shift operation deletes all states that would become non-initial if they were to remain).

Next, there may be at most $2^{d \cdot d}$ transitions from any state at order-$(n-1)$, and thus at most $d \cdot d \cdot 2^{d \cdot d}$ states at order-$(n-2)$ (noting that the shift operation deletes all states that would become non-initial if they were to remain).

Thus, we can repeat this argument down to order-1 and obtain $2 \uparrow_{n-2} (f(\zeta, |\mathcal{P}|))$ states for some computable polynomial f. $\square$

Take the automaton accepting any $\langle p_i, w \rangle$ from $q^1_{p_i}$. This automaton has order-n states of the form q^i_p, and at most a single transition from each of the layer 1 states to $\emptyset$. Each of these transitions is labelled by a state with at most one transition to $\emptyset$, and so on until order-1.

Definition G.1 (N) *Following Lemma G.1, we take* $N = 2 \uparrow_{n-2} (f(\zeta, d))$ *for some computable polynomial* f.

G.3 Proofs for Control State Reachability

In this section, we prove Lemma 6.1 (Simulation by $\mathcal{G}^{p_{\text{out}}}_{\mathcal{C}}$). The proof is split in to two directions, given in Lemma G.2 and Lemma G.3 below.

Lemma G.2 *Given a scope-bounded CPDS* $\mathcal{C}$ *and control states* p_{in} *and* p_{out}*, there is a run of* $\mathcal{C}$ *from* $\langle p_{in}, w_1, \ldots, w_m \rangle$ *to* $\langle p_{out}, w'_1, \ldots, w'_m \rangle$ *for some* $w'_1, \ldots, w'_m$ *only if there is a path in* $\mathcal{G}^{p_{out}}_{\mathcal{C}}$ *from an initial vertex to a vertex*

$$(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$$

where for all i *we have* $\langle p_{i-1}, w_i \rangle$ *accepted from the* $1st$ *layer of* A_i *and* $p_0 = p_{in}$.

Proof. Take a run of the scope-bounded CPDS from $\langle p_{\text{in}}, w_1, \ldots, w_m \rangle$ to $\langle p_{\text{out}}, w'_1, \ldots, w'_m \rangle$. We proceed by induction over the number of rounds in the run. In the following we will override the w_i and w'_i in the statement of the lemma to ease notation.

In the base case, take a single round

$$\langle p_0, w_1, \ldots, w_m \rangle \longrightarrow^* \langle p_1, w'_1, w_2, \ldots, w_m \rangle \longrightarrow^* \cdots \longrightarrow^* \langle p_m, w'_1, \ldots, w'_m \rangle$$

where p_i is the control state after the run on stack i, and w'_i is the ith stack at the end of this run. Take an initial vertex

$$(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)\ .$$

We know A_i is constructed by saturation from an automaton accepting $\langle p_i, w'_i \rangle$ and thus $\langle p_{i-1}, w_i \rangle$ is accepted by A_i from the 1st layer. This vertex then gives us a path in the reachability graph to a vertex where for all i we have $\langle p_{i-1}, w_i \rangle$ accepted from the 1st layer of A_i.

Now consider the inductive step where we have a round

$$\langle p_0, w_1, \ldots, w_m \rangle \longrightarrow^* \langle p_1, w'_1, w_2, \ldots, w_m \rangle \longrightarrow^* \cdots \longrightarrow^* \langle p_m, w'_1, \ldots, w'_m \rangle$$

and a run from $\langle p_m, w'_1, \ldots, w'_m \rangle$ to the destination control state. By induction we have a vertex in the reachability graph

$$\left(p'_0, A'_1, p'_1, \ldots, p'_{m-1}, A'_m, p'_m\right)$$

with $p_m = p'_0$ that is reachable from an initial vertex and has for all i that $\langle p'_{i-1}, w'_i \rangle$ is accepted from the 1st layer of A'_i.

By definition of the reachability graph, there exists an edge to this vertex from a vertex

$$(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m) \ .$$

such that $A_i = \mathtt{Predecessor}_i\left(A'_i, q_{p_i}, q_{p'_{i-1}}\right)$.

Since the run of $\mathcal{C}$ is scope-bounded, we know there is an accepting run of w'_i from $q^1_{p'_{i-1}}$ in A'_i that does not use any layer ζ states (by the further condition described below and since layer ζ corresponds to the round out of scope for elements of w'_i). Therefrom, we have an accepting run of w'_i from $q^2_{p'_{i-1}}$ in $\mathtt{Shift}(A'_i)$. Thus, there is an accepting run of w'_i from p^1_i after the application of $\mathtt{EnvMove}$. Since there is a run over stack i from $\langle p_{i-1}, w_i \rangle$ to $\langle p_i, w'_i \rangle$ we therefore have an accepting run of w_i from $q^1_{p_{i-1}}$ in A_i.

In addition to the above, we need a further property that reflects the scope boundedness. In particular, if no character or stack with pop- or collapse-round 0 is removed during the zth round, then there is a run over w_i that uses only transitions $q \xrightarrow{q'} Q$ to read stacks u such that no layer z state is in Q and, similarly, for characters a, the run uses only transitions $q \xrightarrow[Q_{col}]{a} Q$ to read the instance of a where no layer z state appears in Q and no layer z state appears in Q_{col}.

Note that the base case is for the automata accepting any stack, only containing transitions to the empty set, for which the property is trivial. In the inductive step, we prove this property by further induction over the length of the run from $\langle p_i, w_i \rangle$ to $\langle p_{i+1}, w'_i \rangle$. In the base case we have a run of length 0 and the property holds since, by induction, we can assume that A'_i has the property (with the round numbers shifted) and it is maintained by the $\mathtt{Shift}$ and $\mathtt{EnvMove}$. Hence, assume we have a run beginning $\langle p, w \rangle \longrightarrow \langle p', w' \rangle$ and the required run over w'. We do a case split on the stack operation o associated with the transition.

1. If $o = pop_k$ then we have $w = u :_k v$ and $w' = v$. If $z = 1$ and u has pop-round 0 (i.e. appears in w_i), then this case cannot occur because the transition we're currently analysing appears in round 1 and by assumption u is not removed in round 1. Hence, assume $z > 1$. We had a run over w' from $q^1_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ in A_i respecting the property, and by saturation we have a run over w beginning with

$$q^1_p \xrightarrow[\emptyset]{a} (\emptyset, \ldots, \emptyset, \{q_k\}, Q_{k+1}, \ldots, Q_n)$$

that also respects the property, since q_k is layer 1 and $z \neq 1$.

2. When $o = copy_k$ we have $w = u :_k v$ and $w' = u :_k u :_k v$. Let $q^1_{p'} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k, \ldots Q_n)$ and $Q_k \xrightarrow[Q'_{col}]{a} (Q'_1, \ldots, Q'_k)$ be the initial transitions used on the run of w'. We know neither these transitions, nor the runs from these transitions, pass a layer z state on any component with pop- or collapse-round 0. Furthermore, we know the first u has pop-round 1. The second u may have pop-round 0. If it does, we know Q'_k does not contain any layer z states.

   From the saturation algorithm, we have a transition

   $$q^1_p \xrightarrow[Q_{col} \cup Q'_{col}]{a} \left(Q_1 \cup Q'_1, \ldots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \ldots, Q_n\right) \ .$$

   from which we have an accepting run of w that satisfies the property.

3. If $o = collapse_k$, $w = a^{u'} :_1 u :_{(k+1)} v$ and $w' = u' :_{(k+1)} v$. When $k = n$, we have an accepting run of w' respecting the property, and from the saturation, an accepting run of w beginning with a transition $q^1_p \xrightarrow[\left\{q^1_{p'}\right\}]{a} (\emptyset, \ldots, \emptyset)$ and $w' = u'$. When $z = 1$ and a has collapse-round 0, this case cannot occur because the transition we're currently analysing appears in round 1 (similarly to the pop_k case). Otherwise $z > 1$ and we have a run over w respecting the property.

   When $k < n$, we have an accepting run of w' in beginning with $q^1_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ that respects the property. By saturation, we have an accepting run of w beginning with a transition $q^1_p \xrightarrow[\{q_k\}]{a} (\emptyset, \ldots, \emptyset, Q_{k+1}, \ldots, Q_n)$. If the collapse-round of a is 0 and $z = 1$, this case cannot occur. Otherwise, the run over w satisfies the property since the run over w' does and q_k is layer 1 and $z > 1$.

4. When $o = push^k_c$, let $w = u_{k-1} :_k u_k :_{k+1} \cdots :_n u_n$. We know $w' = push^k_c(w)$ is

   $$c^{u_k} :_1 u_{k-1} :_k \cdots :_n u_n \ .$$

   Let $q^1_{p'} \xrightarrow[Q_{col}]{c} (Q_1, \ldots, Q_n)$ and $Q_1 \xrightarrow[Q'_{col}]{a} Q'_1$ be the first transitions used on the accepting run of w'. If the pop-round of a is 0, we know there are no layer z states in Q'_1. Similarly if the pop-round of u_k is 0 we know that there are no layer z states in Q_{col}. The saturation algorithm means we have $q^1_p \xrightarrow[Q'_{col}]{a} (Q'_1, Q_2, \ldots, Q_k \cup Q_{col}, \ldots, Q_n)$ leading to an accepting run that respects the property.

5. If $o = rew_b$ then $w = a^u :_1 v$ and $w' = b^u :_1 v$. Note none of the pop- or collapse-rounds are changed, and the run of w' beginning $q^1_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ and satisfying the property implies a run of w beginning $q^1_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ and also satisfying the property.

6. If $o = noop$ then $w = a^u :_1 v$ and $w' = a^u :_1 v$. Note none of the pop- or collapse-rounds are changed, and the run of w' beginning $q^1_{p'} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ and satisfying the property implies a run of w beginning $q^1_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ and also satisfying the property.

Finally then, by induction over the number of rounds, we reach the first round beginning with $\langle p_0, w_1, \ldots, w_m \rangle$ and we know there is a path from an initial vertex to a vertex

$$(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$$

with $p_0 = p$ and for all i we have $\langle p_{i-1}, w_i \rangle$ accepted from the 1st layer of A_i. $\square$

Lemma G.3 *Given a scope-bounded CPDS $\mathcal{C}$ and control states p_{in} and p_{out}, there is a run of $\mathcal{C}$ from $\langle p_{in}, w_1, \ldots, w_m \rangle$ to $\langle p_{out}, w'_1, \ldots, w'_m \rangle$ for some $w'_1, \ldots, w'_m$ whenever there is a path in $\mathcal{G}_{\mathcal{C}}^{p_{out}}$ from an initial vertex to a vertex*

$$(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$$

with $p_0 = p_{in}$ and for all i we have $\langle p_{i-1}, w_i \rangle$ accepted from the 1st layer of A_i.

Proof. Note, in the following proof, we override the w_i and w'_i in the statement of the lemma. Take a path in the reachability graph. The proof goes by induction over the length of the path. When the path is of length 0 we have a single vertex $(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$. Take any configuration $\langle p_{i-1}, w_i \rangle$ accepted by A_i. We know A_i accepts all configurations that can reach $\langle p_i, w \rangle$ for some w. Therefore, from the initial configuration

$$\langle p_0, w_1, \ldots, w_m \rangle$$

we first apply the run over the 1st stack to p_1 to obtain

$$\langle p_1, w'_1, w_2, \ldots, w_m \rangle$$

for some w'_1. Then we apply the run over the 2nd stack to p_2 and so on until we reach

$$\langle p_m, w'_1, \ldots, w'_m \rangle$$

for some $w'_1, \ldots, w'_m$. This witnesses the reachability property as required.

Now consider the inductive case where we have a path beginning with an edge of the reachability graph from

$$(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$$

to

$$\left(p'_0, A'_1, p'_1, \ldots, p'_{m-1}, A'_m, p'_m\right) \ .$$

By induction we have a run from

$$\langle p_m, w'_1, \ldots, w'_m \rangle$$

to the final control state for any w'_i accepted by A'_i from $q^1_{p_{i-1}}$.

Now, similarly to the base case, take any configuration $\langle p_{i-1}, w_i \rangle$ accepted by A_i. We know A_i accepts all configurations that can reach $\langle p_i, w \rangle$ for some w accepted from $q^2_{p'_{i-1}}$ in $\mathtt{Shift}(A'_i)$ and therefore, from $q^1_{p'_{i-1}}$ in A'_i. Hence, from the initial configuration

$$\langle p_0, w_1, \ldots, w_m \rangle$$

we first apply the run over the 1st stack to p_1 to obtain

$$\langle p_1, w'_1, w_2, \ldots, w_m \rangle$$

for some w'_1. Then we apply the run over the 2nd stack to p_2 and so on until we reach

$$\langle p_m, w'_1, \ldots, w'_m \rangle$$

for some $w'_1, \ldots, w'_m$ and then, by induction, we have a run from this configuration to the target control state as required.

We need to prove a stronger property that we can in fact build a scope-bounded run. In particular, we show that, for all stacks u in w_i, if the accepting run of w_i uses only transitions $q \xrightarrow{q'} Q$ to read u such that no layer z state is in Q, then there is a run to the final control state such that u is not popped during round z. Similarly, for characters a, if the accepting run uses only transitions $q \xrightarrow[Q_{col}]{a} Q$ to read the instance of a where no layer z state appears in Q, then a is not popped in round z. Similarly, if no layer z state appears in Q_{col}, then collapse is not called on that character during round z. We observe the property is trivially true for the base case where the automata accept any stack using only transitions to $\emptyset$. The inductive case is below.

We start from $\langle p, w \rangle = \langle p_i, w_i \rangle$. First assign each stack and character in w pop- and collapse-round 0. Noting that A is obtained by saturation from A' (after a `Shift` and `EnvMove` — call this automaton B), we aim to exhibit a run from $\langle p, w \rangle$ to $\langle p_{i+1}, w_{i+1} \rangle$ (in fact we choose w_{i+1} via this procedure) such that all stacks and characters in w_{i+1} with pop- or collapse-round 0 do not pass layer z states in B. Since we have a run over w_{i+1} in A'_i that does not pass layer 1 states for parts of the stack with pop- or collapse-round 0, we know by induction we have a run from $\langle p_{i+1}, w_{i+1} \rangle$ that is scope bounded.

To generate such a run we follow the counter-example generation algorithm in [9]. We refer the reader to this paper for a precise exposition of the algorithm. Furthermore, that this routine terminates is non-trivial and requires a subtle well-founded relation over stacks, which is also shown in [9].

Beginning with the run over $\langle p_i, w_i \rangle$ that has the property of not passing layer z states, we have our base case. Now assume we have a run to $\langle p, w \rangle$ such that the run over w has no transitions to layer z states reading stacks or characters with pop- or collapse-rounds of 0. We take the first transition of such a run, which was introduced by the saturation algorithm because of a rule (p, a, o, p') and certain transitions of the partially saturated B. Let $\langle p', w' \rangle$ be the configuration reached via this rule. We do a case split on o.

1. If $o = pop_k$, then we have $w = u :_k v$ and the accepting run of w begins with

$$q^1_p \xrightarrow[\emptyset]{a} (\emptyset, \ldots, \emptyset, \{q_k\}, Q_{k+1}, \ldots, Q_n)$$

   where $q^1_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ was already in B. This gives us an accepting run of v beginning with this transition. Note that q_k is of layer 1. Thus, if u has pop-round 0 and $z = 1$, this case cannot occur. Otherwise, we have that the run of v visits a subset of the states in the run over w and thus maintains the property.

2. If $o = copy_k$, then we have $w = u :_k v$ and $w' = u :_k u :_k v$. Furthermore, we had an accepting run of w using the initial transition

$$q^1_p \xrightarrow[Q_{col} \cup Q'_{col}]{a} \left(Q_1 \cup Q'_1, \ldots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \ldots, Q_n\right)$$

   and an accepting run of B on w' using the initial transitions $q^1_{p'} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k, \ldots, Q_n)$ and $Q_k \xrightarrow[Q'_{col}]{a} (Q'_1, \ldots, Q'_k)$ from which we have an accepting run over w'. Note that, to prove the required property, we observe that for all elements of w' obtaining their pop- and collapse-rounds from w, the targets of the transitions used to read them already appear in the run of w, hence the run satisfies the property. The only new part of the run is to Q'_k after reading the new copy of u, which has pop-round 1. Thus the property is maintained.

3. If $o = collapse_k$ then we have $w = a^{u'} :_1 u :_{(k+1)} v$ and $w' = u' :_{(k+1)} v$. When $k = n$, the accepting run of w begins with a transition $q_p^1 \xrightarrow[\{q^1_{p'}\}]{a} (\emptyset, \ldots, \emptyset)$ and $w' = u'$. When $z = 1$ and a has collapse-round 0, this case cannot occur because the initial transition goes to a layer z state. Otherwise, we have a run over w' that is a subrun of that over w, and thus the property is transferred.

   When $k < n$, the accepting run of w begins with $q_p^1 \xrightarrow[\{q_k\}]{a} (\emptyset, \ldots, \emptyset, Q_{k+1}, \ldots, Q_n)$ and we have an accepting run of w' in B beginning with $q^1_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$. If the collapse-round of a is 0 and $z = 1$, this case cannot occur because q_k is layer z. Otherwise, the run over w' is a subrun of that over w and the property is transferred.

4. If $o = push_b^k$ then $w' = b^u :_1 w$ where $u = top_{k+1}(pop_k(w))$ and the collapse-round of b is the pop-round of $top_k(w)$. The run of w begins with a transition

$$q_p^1 \xrightarrow[Q'_{col}]{a} (Q'_1, Q_2, \ldots, Q_{k-1}, Q_k \cup Q_{col}, Q_{k+1}, \ldots, Q_n)$$

   and there is a run over w' in B beginning with $q^1_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ and $Q_1 \xrightarrow[Q'_{col}]{a} Q'_1$. Note that, to prove the required property, we observe that for all elements of w' obtaining their pop- and collapse-rounds from w, the targets of the transitions used to read them already appear in the run of w, hence the run satisfies the property. The only new parts of the run are to Q'_1 after reading b, which has pop-round 1, and the transition to Q_{col} on the collapse branch of b. Note, however, that b has the collapse-round equal to the pop-round of $top_k(w)$ and hence we know that Q_{col} has no layer z states if the collapse-round of b is 0. Thus the property is maintained.

5. If $o = rew_b$ then $w = a^u :_1 v$ and $w' = b^u :_1 v$. Note none of the pop- or collapse-rounds are changed, and the run of w beginning $q_p^1 \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ and satisfying the property implies a run of w' in B beginning $q^1_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ and also satisfying the property.

6. If $o = noop$ then $w = a^u :_1 v$ and $w' = a^u :_1 v$. Note none of the pop- or collapse-rounds are changed, and the run of w beginning $q_p^1 \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ and satisfying the property implies a run of w' in B beginning $q^1_{p'} \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ and also satisfying the property.

Thus we are done. □

G.4 Complexity

Solving the control state reachability problem requires finding a path in the reachability graph. Since each vertex can be stored in $\mathcal{O}(2 \uparrow_{n-1} (f(\zeta, \ell)))$ space, where f is a polynomial and ℓ the number of control states, and we require $\mathcal{O}(2 \uparrow_{n-1} (f(\zeta, \ell)))$ time to decide the edge relation, we have via Savitch's algorithm, a $\mathcal{O}(2 \uparrow_{n-1} (f(\zeta, \ell)))$ space procedure for deciding the control state reachability problem. We also observe that the solution to the global control state reachability problem may contain at most $\mathcal{O}(2 \uparrow_{n} (f(\zeta, \ell)))$ tuples.