HAL
open science

A Saturation Method for Collapsible Pushdown Systems

Christopher Broadbent, Arnaud Carayol, Matthew Hague, Olivier Serre

To cite this version:

Christopher Broadbent, Arnaud Carayol, Matthew Hague, Olivier Serre. A Saturation Method for Collapsible Pushdown Systems. Automata, Languages and Programming, 39th International Colloquium, ICALP 2012, 2012, Warwick, United Kingdom. pp.165-176, 10.1007/978-3-642-31585-5_18 . hal-00694991

HAL Id: hal-00694991

https://hal.science/hal-00694991

Submitted on 7 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Saturation Method for Collapsible Pushdown Systems*

C. Broadbent[1], A. Carayol[2], M. Hague[1,2], and O. Serre[1]

[1] LIAFA, Université Paris Diderot – Paris 7 & CNRS
[2] LIGM, Université Paris-Est & CNRS

Abstract. We introduce a natural extension of collapsible pushdown systems called annotated pushdown systems that replaces collapse links with stack annotations. We believe this new model has many advantages. We present a saturation method for global backwards reachability analysis of these models that can also be used to analyse collapsible pushdown systems. Beginning with an automaton representing a set of configurations, we build an automaton accepting all configurations that can reach this set. We also improve upon previous saturation techniques for higher-order pushdown systems by significantly reducing the size of the automaton constructed and simplifying the algorithm and proofs.

1 Introduction

Via languages such as C++, Haskell, Javascript, Python, or Scala, modern day programming increasingly embraces higher-order procedure calls. This is a challenge for software verification, which usually does not model recursion accurately, or models only first-order calls (e.g. SLAM [2] and Moped [29]). Collapsible pushdown systems (collapsible PDS) are an automaton model of (higher-order recursion) schemes [11, 24], which allow reasoning about higher-order recursion.

Collapsible pushdown systems are a generalisation of higher-order pushdown systems (higher-order PDS). Higher-order PDS provide a model of schemes subject to a technical constraint called *safety* [23, 19] and are closely related to the Caucal hierarchy [9]. These systems extend the stack of a pushdown system to allow a nested "stack-of-stacks" structure. Recently it has been shown by Parys that safety is a genuine constraint on definable traces [26]. Hence, to model higher-order recursion fully, we require collapsible PDS, which — using an idea from *panic automata* [20] — add additional *collapse* links to the stack structure. These links allow the automaton to return to the context in which a character was added to the stack.

These formalisms are known to have good model-checking properties. For example, it is decidable whether a given μ-calculus formula holds on the execution graph of a scheme [24] (or collapsible PDS [14]). Although, the complexity of

* This work was supported by the Fondation Sciences Mathématiques de Paris and by the following projects: AMIS (ANR 2010 JCJC 0203 01 AMIS), FREC (ANR 2010 BLAN 0202 02 FREC) and VAPF (Région Île de France).

such analyses is high — for an order-n collapsible PDS, reachability checking is complete for $(n-1)$-EXPTIME, while μ-calculus is complete for n-EXPTIME — the problem becomes PTIME if the arity of the recursion scheme, and the number of alternations in the formula, is bounded. The same holds true for collapsible PDS when the number of control states is bounded. Furthermore, when translating from a scheme to a collapsible PDS, it is the arity that determines the number of control states [14]. It has been shown by Kobayashi [21] that these analyses can be performed in practice. For example, resource usage properties of programs of orders up to five can be verified in a matter of seconds.

Kobayashi *et al.*'s approach uses *intersection types*. In the order-1 case, an alternative approach called *saturation* has been successfully implemented by tools such as Moped [29] and PDSolver [16]. Saturation techniques begin with a small automaton — representing a set of configurations — and add new transitions as they become necessary until a fixed point is reached. These algorithms, then, naturally do not pay the worst case complexity immediately, and hence, represent ideal algorithms for efficient verification. Furthermore, they also provide a solution to the *global* model checking problem: that is, determining the set of all system states that satisfy a property. This is particularly useful when, for example, composing analyses. Furthermore, when testing reachability from a given initial state, we may terminate the analysis as soon as this state is found. That is, we do not need to compute the whole fixed point.

Our first contribution is a new model of higher-order execution called *annotated pushdown systems* (annotated PDS)[1], which replace the collapse links of a collapsible PDS with annotations containing the stack the link pointed to. In addition to allowing a more straightforward definition of regularity and greatly simplifying the proofs of the paper, this model provides a more natural handling of collapse links, highlighting their connection with closures. In addition, configuration graphs of this model are isomorphic to those of collapsible PDS when restricted to configurations reachable from the initial configuration.

Our second contribution is a saturation method for backwards reachability analysis of annotated pushdown systems that can also be applied *as-is* to collapsible PDS. This is a global model-checking algorithm that is based on saturation techniques for higher-order pushdown automata [5, 15, 30]. Our algorithm handles alternating (or "two-player") as well as non-alternating systems.

In addition to the extension to annotated pushdown systems, the algorithm improves on Hague and Ong's construction for higher-order pushdown systems [15] since the number of states introduced by the construction is no longer multiplied by the number of iterations it takes to reach a fixed point, potentially leading to a large reduction in the size of the automata constructed. In addition, both the presentation and the proofs of correctness are much less involved.

Related Work In addition to the works mentioned above, solutions to global model checking problems have been proposed by Broadbent *et al.* [6]. Additionally, Salvati and Walukiewicz provide a global analysis technique for μ-calculus

[1] Kartzow and Parys have independently introduced a similar model [18].

properties using a Krivine machine model of schemes [28]. However, there are currently no versions of these algorithms available that do not pay immediately the exponential blow up.

Extensions of schemes with pattern matching have also been considered by Ong and Ramsay [25]. A recent algorithm by Kobayashi speeds up his techniques using an over-approximating least fixed point computation to give an initial input to a greatest fixed point computation [22]. Like saturation this is a 'bottom-up' approach and it would be interesting to see whether there are connections. Extensions of higher-order PDS to concurrent settings have also been considered by Seth [31].

The saturation technique has proved popular in the literature. It was introduced by Bouajjani *et al.* [4] and Finkel *et al.* [13] and based on a string rewriting algorithm by Benois [3]. It has since been extended to Büchi games [7], parity and μ-calculus conditions [16], and concurrent systems [32, 1], as well as weighted pushdown systems [27]. In addition to various implementations, efficient versions of these algorithms have also been developed [12, 33].

2 Preliminaries

2.1 Annotated Pushdown Systems

We define annotated stacks, their operations, and annotated pushdown systems.

Annotated stacks Let Σ be a set of stack symbols. We define a notion of annotated higher-order stack. Intuitively, an annotated stack of order-n is an order-n stack in which stack symbols have attached annotated stacks of order at most n. For the rest of the formal definitions, 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_k^n of stacks of order-k whose symbols are annotated by stacks of order at most n. Note, we use subscripts to indicate the order of a stack.

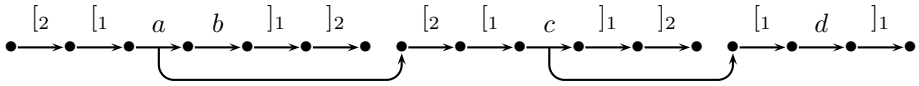
Definition 1 (Annotated Stacks). *The family of sets* $(\mathrm{Stacks}_k^n)_{1 \leq k \leq n}$ *is the smallest family (for point-wise inclusion) such that:*

- *for all* $2 \leq k \leq n$, Stacks_k^n *is the set of all (possibly empty) sequences* $[w_1 \ldots w_\ell]_k$ *with* $w_1, \ldots, w_\ell \in \mathrm{Stacks}_{k-1}^n$.
- Stacks_1^n *is all sequences* $[{a_1}^{w_1} \ldots {a_\ell}^{w_\ell}]_1$ *with* $\ell \geq 0$ *and for all* $1 \leq i \leq \ell$, a_i *is a stack symbol in* Σ *and* w_i *is an annotated stack in* $\bigcup_{1 \leq k \leq n} \mathrm{Stacks}_k^n$.

Observe that the above definition uses a least fixed-point. This ensures that all stacks are finite; in particular a stack cannot contain itself as an annotation. When the maximal order n is clear, we simply write Stacks_k instead of Stacks_k^n. We also write order-k stack to designate an annotated stack in Stacks_k^n.

An order-n stack can be represented naturally as an edge-labelled tree over the alphabet $\{[_{n-1}, \ldots, [_1,]_1, \ldots,]_{n-1}\} \uplus \Sigma$, with Σ-labelled edges having a second target to the tree representing the annotation. For technical convenience, a

tree representing an order-k stack does not use $[_k$ or $]_k$ symbols (these appear uniquely at the beginning and end of the stack). An example order-3 stack is given below, with only a few annotations shown. 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

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

noting that $top_k(w)$ is undefined if $top_{k'}(w)$ is empty 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 $k' > k$, $u :_k v = [u :_k v_1, \dots, v_\ell]_{k'}$. This composition associates to the right. For example, the order-3 stack above can be written $[[[a^w b]_1]_2]_3$ and also $u :_3 v$ 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 Annotated Stacks The following operations can be performed on an order-n stack. We say $o \in \mathcal{O}_n$ is of order-k when k is minimal such that $o \in \mathcal{O}_k$. For example, $push_k$ is of order k.

$$\begin{aligned} \mathcal{O}_n = \{pop_1, \dots, pop_n\} \ \cup \ \{push_2, \dots, push_n\} \ \cup \ \{collapse_2, \dots, collapse_n\} \ \cup \\ \left\{ \ push_a^1, \dots, push_a^n, rew_a \ \mid a \in \Sigma \ \right\} \end{aligned}$$

We define each stack operation for an order-n stack w. Annotations are created by $push_a^k$, which add a character to the top of a given stack w annotated by $top_{k+1}(pop_k(w))$. This gives a access to the context in which it was created. In Section 3.2 we give several examples of these operations.

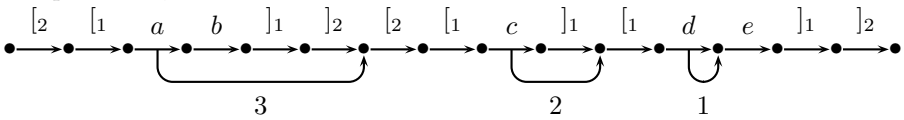
1. We set $pop_k(u :_k v) = v$.
2. We set $push_k(u :_k v) = u :_k u :_k v$.
3. We set $collapse_k\Big(a^{u'} :_1 u :_{(k+1)} v\Big) = u' :_{(k+1)} v$ when u is order-k and $n > k \geq 1$; and $collapse_n(a^u :_1 v) = u$ when u is order-n.
4. We set $push_b^k(w) = b^u :_1 w$ where $u = top_{k+1}(pop_k(w))$.
5. We set $rew_b(a^u :_1 v) = b^u :_1 v$.

Annotated Pushdown Systems We are now ready to define annotated PDS.

Definition 2 (Annotated Pushdown Systems). *An order-n* alternating annotated pushdown system (annotated PDS) *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 $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P}) \cup \big(\mathcal{P} \times 2^{\mathcal{P}}\big)$ is a set of rules.*

We write *configurations* of an annotated PDS as a pair $\langle p, w\rangle$ where $p \in \mathcal{P}$ and $w \in Stacks_n$. We write $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ to denote a transition from a rule (p, a, o, p') with $top_1(w) = a$ and $w' = o(w)$. Furthermore, we have a transition $\langle p, w\rangle \longrightarrow \{\ \langle p', w\rangle \mid p' \in P\ \}$ whenever we have a rule $p \to P$. A non-alternating annotated PDS has no rules of this second form. We write C to denote a set of configurations.

Collapsible Pushdown Systems Annotated pushdown systems are based on collapsible PDS. In this model, stacks do not contain order-k annotations, rather they have order-k links to an order-k stack occurring lower down in the top-most order-$(k+1)$ stack. We define the model formally in the appendix. We give an example below, where links are marked with their order.



The set $\mathcal{O}_n$ is the same as in the annotated version. Collapse links are created by the $push_a^k$ operation, which augments a with a link to pop_k of the stack being pushed onto. A $collapse_k$ returns to the stack that is the target of the link.

Collapsible vs. Annotated To an order-n stack w with links, we associate a canonical annotated stack $[\![\, w \,]\!]$ where each link is replaced by the annotated version of the link's target. We inductively and simultaneously define $[\![\, w \,]\!]_k$ which is the annotated stack representing $top_k(w)$.

$$\begin{cases} [\![\, [\,]_k :_{k+1} v \,]\!]_k = [\,]_k & \\ [\![\, u :_{k'+1} v \,]\!]_k = [\![\, u \,]\!]_{k'} :_{k'+1} [\![\, v \,]\!]_k & \text{where } 0 < k' < k \\ [\![\, a^* :_1 v \,]\!]_k = a^{[\![\, collapse_{k'}(a^* :_1 v) \,]\!]_{k'}} :_1 [\![\, v \,]\!]_k & \text{where } a^* \text{ is } a \text{ with an order-}k' \text{ link} \end{cases}$$

For example, the order-3 stack above becomes $[[[a^{w_1} b]_1]_2 [[c^{w_2}]_1 [d^{w_3} e]_1]_2]_3$ where $w_1 = [[[c^{w_2}]_1 [d^{w_3} e]_1]_2]_3$, $w_2 = [[d^{w_3} e]_1]_2$ and $w_3 = [e]_1$.

Note that some annotated stacks such as $[[a^w]_1]_2$ with $w = [[b^{[]_1}]_1]_2$ do not correspond to any stacks with links. However for all order-n stacks with links w and for any operation o of order at most n, we have $[\![\, o(w) \,]\!] = o([\![\, w \,]\!])$.

Remark 1. The configuration graphs of annotated pushdown systems of order-n are isomorphic to their collapsible counter-part when restricted to configurations reachable from the initial configuration. This implies annotated pushdown automata generate the same trees as higher-order recursion schemes, as in [6].

2.2 Regularity of Annotated Stacks

We will present an algorithm that operates on sets of configurations. For this we use order-n stack automata, thus defining a notion of regular sets of stacks. These have a nested structure based on a similar automata model by Bouajjani and Meyer [5]. The handling of annotations is similar to automata introduced by Broadbent *et al.* [6], except we read stacks top-down rather than bottom-up.

Definition 3 (Order-n Stack Automata). *An* order-n stack automaton

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

is a tuple where Σ is a finite stack alphabet, and

1. *for all $n \geq k \geq 2$, we have $\mathcal{Q}_k$ is a finite set of states, $\Delta_k \subseteq \mathcal{Q}_k \times \mathcal{Q}_{k-1} \times 2^{\mathcal{Q}_k}$ is a transition relation, and $\mathcal{F}_k \subseteq \mathcal{Q}_k$ is a set of accepting states, and*
2. *$\mathcal{Q}_1$ is a finite set of states, $\Delta_1 \subseteq \bigcup_{2 \leq k \leq n} \left(\mathcal{Q}_1 \times \Sigma \times 2^{\mathcal{Q}_k} \times 2^{\mathcal{Q}_1}\right)$ a transition relation, and $\mathcal{F}_1 \subseteq \mathcal{Q}_1$ a set of accepting states.*

Stack automata are alternating automata that read the stack in a nested fashion. Order-k stacks are recognised from states in $\mathcal{Q}_k$. A transition $(q, q', Q) \in \Delta_k$ from q to Q for some $k > 1$ can be fired when the top_{k-1} stack is accepted from $q' \in \mathcal{Q}_{(k-1)}$. The remainder of the stack must be accepted from all states in Q. At order-1, a transition (q, a, Q_{br}, Q) is a standard alternating a-transition with the additional requirement that the stack annotating a is accepted from all states in Q_{br}. A stack is accepted if a subset of $\mathcal{F}_k$ is reached at the end of each order-k stack. In the appendix, we formally define the runs of a stack automaton. We write $w \in \mathcal{L}_q(A)$ whenever w is accepted from a state q.

A (partial) run is pictured below, using $q_3 \xrightarrow{q_2} Q_3 \in \Delta_3, q_2 \xrightarrow{q_1} Q_2 \in \Delta_2$ and $q_1 \xrightarrow[Q_{br}]{a} Q_1 \in \Delta_1$. The node labelled Q_{br} begins a run on the stack annotating a.

$$q_3 \xrightarrow{[_2} q_2 \xrightarrow{[_1} q_1 \xrightarrow{a} Q_1 \xrightarrow{\cdots} \cdots \xrightarrow{]_1} Q_2 \xrightarrow{\cdots} \cdots \xrightarrow{]_2} Q_3 \xrightarrow{\cdots} \cdots \qquad Q_{br} \xrightarrow{\cdots} \cdots$$

Remark 2. In the appendix, we show several results on stack automata: membership testing is linear time; emptiness is PSPACE-complete; the sets of stacks accepted by these automata form an effective Boolean algebra (note that complementation causes a blow-up in the size of the automaton); and they accept the same family of collapsible stacks as the automata used by Broadbent *et al.* [6].

3 Algorithm

Given an annotated PDS $\mathcal{C}$ and a stack automaton A_0 with a state $q_p \in \mathcal{Q}_n$ for each control state p in $\mathcal{C}$, we define $Pre^*_{\mathcal{C}}(A_0)$ as the smallest set such that $Pre^*_{\mathcal{C}}(A_0) \supseteq \{ \langle p, w \rangle \mid w \in \mathcal{L}_{q_p}(A_0) \}$, and

$$Pre^*_{\mathcal{C}}(A_0) \supseteq \left\{ \langle p, w \rangle \;\middle|\; \begin{array}{l} \exists \langle p, w \rangle \longrightarrow \langle p', w' \rangle \text{ with } \langle p', w' \rangle \in Pre^*_{\mathcal{C}}(A_0) \vee \\ \exists \langle p, w \rangle \longrightarrow C \text{ and } C \subseteq Pre^*_{\mathcal{C}}(A_0) \end{array} \right\}$$

recalling that C denotes a set of configurations. We build a stack automaton recognising $Pre^*_{\mathcal{C}}(A_0)$. We begin with A_0 and iterate a saturation function denoted Γ — which adds new transitions to A_0 — until a 'fixed point' has been reached. That is, we iterate $A_{i+1} = \Gamma(A_i)$ until $A_{i+1} = A_i$. As the number of states is bounded, we eventually obtain this, giving us the following theorem.

Theorem 1. *Given an alternating annotated pushdown system $\mathcal{C}$ and a stack automaton A_0, we can construct an automaton A accepting $Pre^*_{\mathcal{C}}(A_0)$.*

The construction runs in n-EXPTIME for both alternating annotated PDS and collapsible PDS — which is optimal — and can be improved to $(n-1)$-EXPTIME for non-alternating collapsible PDS when the initial automaton satisfies a certain notion of *non-alternation*, again optimal. Correctness and complexity are discussed in subsequent sections.

3.1 Notation and Conventions

Number of Transitions We assume for all $q \in \mathcal{Q}_k$ and $Q \subseteq \mathcal{Q}_k$ that there is at most one transition of the form $q \xrightarrow{q'} Q \in \Delta_k$. This condition can easily be ensured on A_0 by replacing pairs of transitions $q \xrightarrow{q_1} Q$ and $q \xrightarrow{q_2} Q$ with a single transition $q \xrightarrow{q'} Q$, where q' accepts the union of the languages of stacks accepted from q_1 and q_2. The construction maintains this condition.

Short-form Notation We introduce some short-form notation for runs. Consider the example run in Section 2.2. In this case, we write $q_3 \xrightarrow[Q_{br}]{a} (Q_1, Q_2, Q_3)$, $q_3 \xrightarrow{q_1} (Q_2, Q_3)$, and $q_3 \xrightarrow{q_2} (Q_3)$. In general, we write

$$q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k) \text{ and } q \xrightarrow{q'} (Q_{k'+1}, \ldots, Q_k)\,.$$

In the first case, $q \in \mathcal{Q}_k$ and there exist $q_{k-1}, \ldots, q_1$ such that $q \xrightarrow{q_{k-1}} Q_k \in \Delta_k, q_{k-1} \xrightarrow{q_{k-2}} Q_{k-1} \in \Delta_{k-1}, \ldots, q_1 \xrightarrow[Q_{br}]{a} Q_1 \in \Delta_1$. Thus, we capture nested sequences of initial transitions from q. Since we assume at most one transition between any state and set of states, the intermediate states $q_{k-1}, \ldots, q_1$ are uniquely determined by q, a, Q_{br} and $Q_1, \ldots, Q_k$.

In the second case $q \in \mathcal{Q}_k$, $q' \in \mathcal{Q}_{k'}$, and there exist $q_{k-1}, \ldots, q_{k'+1}$ with $q \xrightarrow{q_{k-1}} Q_k \in \Delta_k, q_{k-1} \xrightarrow{q_{k-2}} Q_{k-1} \in \Delta_{k-1}, \ldots, q_{k'+2} \xrightarrow{q_{k'+1}} Q_{k'+2} \in \Delta_{k'+2}$ and $q_{k'+1} \xrightarrow{q'} Q_{k'+1} \in \Delta_{k'+1}$.

We lift the short-form transition notation to transitions from sets of states. We assume that state-sets $\mathcal{Q}_n, \ldots, \mathcal{Q}_1$ are disjoint. Suppose $Q = \{q_1, \ldots, q_\ell\}$ and for all $1 \leq i \leq \ell$ we have $q_i \xrightarrow[Q^i_{br}]{a} \left(Q^i_1, \ldots, Q^i_k\right)$. Then we have $Q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$ where $Q_{br} = \bigcup_{1 \leq i \leq \ell} Q^i_{br}$ and for all k, $Q_k = \bigcup_{1 \leq i \leq \ell} Q^i_k$. Because an annotation can only be of one order, we insist that $Q_{br} \subseteq \mathcal{Q}_k$ for some k.

Finally, we remark that a transition to the empty set is distinct from having no transition.

Initial States 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}. We make the assumption that all initial states do not have any incoming transitions and that they are not final[2]

Adding Transitions Finally, when we add a transition $q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$ to the automaton, then for each $n \geq k > 1$, we add $q_k \xrightarrow{q_{k-1}} Q_k$ to Δ_k (if a transition between q_k and Q_k does not already exist, otherwise we use the existing transition and state q_{k-1}) and add $q_1 \xrightarrow[Q_{br}]{a} Q_1$ to Δ_1.

3.2 The Saturation Function

Given an annotated PDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R})$, we define the saturation function. Examples can be found below.

Definition 4 (The Saturation Function Γ). *Given an order-n stack automaton A we define $A' = \Gamma(A)$ such that A' is A plus, for each $(p, a, o, p') \in \mathcal{R}$,*

1. *when $o = pop_k$, for each $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ in A, add to A'*
$$q_p \xrightarrow[\emptyset]{a} (\emptyset, \ldots, \emptyset, \{q_k\}, Q_{k+1}, \ldots, Q_n) \ ,$$
2. *when $o = push_k$, for all transitions $q_{p'} \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k, \ldots, Q_n)$ and $Q_k \xrightarrow[Q'_{br}]{a} (Q'_1, \ldots, Q'_k)$ in A, add to A' the transition*
$$q_p \xrightarrow[Q_{br} \cup Q'_{br}]{a} \left(Q_1 \cup Q'_1, \ldots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \ldots, Q_n\right) \ ,$$
3. *when $o = collapse_k$, when $k = n$, add $q_p \xrightarrow[\{q_{p'}\}]{a} (\emptyset, \ldots, \emptyset)$, and when $k < n$, for each transition $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ in A, add to A' the transition*
$$q_p \xrightarrow[\{q_k\}]{a} (\emptyset, \ldots, \emptyset, Q_{k+1}, \ldots, Q_n) \ ,$$
4. *when $o = push_b^k$ for all transitions $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$ and $Q_1 \xrightarrow[Q'_{br}]{a} Q'_1$ in A with $Q_{br} \subseteq \mathcal{Q}_k$, add to A' the transition*
$$q_p \xrightarrow[Q'_{br}]{a} (Q'_1, Q_2, \ldots, Q_k \cup Q_{br}, \ldots, Q_n) \ ,$$
5. *when $o = rew_b$ for each transition $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$ in A, add to A' the transition $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$.*

Finally, for every rule $p \to P$, let $Q = \{ \ q_{p'} \ | \ p' \in P \ \}$, then, for each $Q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$, add a transition $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$. For convenience, the state-sets of A' are defined implicitly from the states used in the transition relations.

[2] Hence automata cannot accept empty stacks from initial states. This can be overcome by introducing a bottom-of-stack symbol.

Examples All examples except one use the order-2 stack w', labelled by a run of a stack automaton, pictured below, where the sub-script indicates states in $\mathcal{Q}_1$ or $\mathcal{Q}_2$. Recall that the first transition of the run can be written $q_{p'} \xrightarrow[Q^2_{br}]{a} \big(Q^1_1, Q^1_2\big)$.

$$q_{p'} \xrightarrow{[_1} q_1 \xrightarrow{a} Q^1_1 \xrightarrow{]_1} Q^1_2 \xrightarrow{[_1} Q^3_1 \xrightarrow{a} Q^4_1 \xrightarrow{]_1} Q^2_2 \qquad Q^1_{br} \xrightarrow{[_1} \cdots \qquad Q^2_{br} \xrightarrow{[_1} \cdots$$

Example of (p, a, pop_2, p') Consider the stack w pictured below with $pop_2(w) = w'$. By the construction, we add a transition $q_p \xrightarrow[\emptyset]{a} (\emptyset, \{q_{p'}\})$ giving the run below labelling w, where q'_1 is the state labelling the new transition $q_{p'} \xrightarrow{q'_1} \{q_{p'}\}$.

$$q_p \xrightarrow{[_1} q'_1 \xrightarrow{a} \emptyset \xrightarrow{c} \emptyset \xrightarrow{]_1} q_{p'} \xrightarrow{[_1} q_1 \xrightarrow{a} Q^1_1 \xrightarrow{]_1} Q^1_2 \xrightarrow{[_1} \cdots \quad Q^1_{br} \xrightarrow{[_1} \cdots$$

Example of $(p, a, push_2, p')$ Consider the stack w below with $push_2(w) = w'$. Take $Q^1_2 \xrightarrow[Q^1_{br}]{a} \big(Q^4_1, Q^2_2\big)$ from the node labelled Q^1_2 in the run over w'. By the construction, we add $q_p \xrightarrow[Q^1_{br} \cup Q^2_{br}]{a} \big(Q^1_1 \cup Q^4_1, Q^2_2\big)$ and obtain a run over w

$$q_p \xrightarrow{[_1} q'_1 \xrightarrow{a} Q^1_1 \cup Q^4_1 \xrightarrow{]_1} Q^2_2 \qquad Q^1_{br} \cup Q^2_{br} \xrightarrow{[_1} \cdots \quad .$$

where q'_1 is the state used by the new transition. This run combines the runs over the top two order-1 stacks of w', ensuring any stack accepted could appear twice on top of a stack already accepted. That is, $push_2(w) = w'$ is in $Pre^*_{\mathcal{C}}(A_0)$.

Example of $(p, a, collapse_2, p')$ Consider the stack w below with $collapse_2(w) = w'$. By the construction, we add a transition $q_p \xrightarrow[\{q_{p'}\}]{a} (\emptyset, \emptyset)$; hence, we have the run below, where q'_1 is the state labelling the new transition $q_{p'} \xrightarrow{q'_1} \{\emptyset\}$.

$$q_p \xrightarrow{[_1} q'_1 \xrightarrow{a} \emptyset \xrightarrow{]_1} \emptyset \xrightarrow{[_1} \emptyset \xrightarrow{c} \emptyset \xrightarrow{]_1} \emptyset \qquad q_{p'} \xrightarrow{[_1} q_1 \xrightarrow{a} Q^1_1 \xrightarrow{]_1} Q^1_2 \xrightarrow{[_1} \cdots \quad Q^1_{br} \xrightarrow{[_1} \cdots$$

Example of $\big(p, a, push^2_b, p'\big)$ The stack of our running example is not constructible via a $push^2_b$ operation. Hence, we use the following stack and run for w'

$$q_{p'} \xrightarrow{[_1} q_1 \xrightarrow{b} Q^1_1 \xrightarrow{a} Q^2_1 \xrightarrow{]_1} Q^1_2 \xrightarrow{[_1} Q^3_1 \xrightarrow{a} Q^4_1 \xrightarrow{]_1} Q^2_2 \qquad Q_{br} \xrightarrow{[_1} Q^5_1 \xrightarrow{a} Q^6_1 \xrightarrow{]_1} Q^3_2$$

with $q_{p'} \xrightarrow[Q_{br}]{b} \big(Q^1_1, Q^1_2\big)$ and $Q^1_1 \xrightarrow[\emptyset]{a} Q^2_1$. The construction adds $q_p \xrightarrow[\emptyset]{a} \big(Q^2_1, Q^1_2 \cup Q_{br}\big)$. This gives us a run on the stack w such that $push^2_b(w) = w'$, where q'_1 is the order-1 state labelling the new order-2 transition.

$$q_{p'} \xrightarrow{[_1} q'_1 \xrightarrow{a} Q^2_1 \xrightarrow{]_1} Q^1_2 \cup Q_{br} \xrightarrow{[_1} Q^3_1 \cup Q^5_1 \xrightarrow{a} Q^4_1 \cup Q^6_1 \xrightarrow{]_1} Q^2_2 \cup Q^3_2$$

4 Correctness and Complexity

Theorem 2. *For a given $\mathcal{C}$ and A_0, let $A = A_i$ where i is the least index such that $A_{i+1} = \Gamma(A_i)$. We have $w \in \mathcal{L}_{q_p}(A)$ iff $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$.*

The proof is in the appendix. Completeness is by a straightforward induction over the "distance" to A_0. Soundness is the key technical challenge. The idea is to 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 $q \in \mathcal{Q}_k$, which is denoted $w \models q$.

Definition 5 ($w \models q$)**.** *For any $Q \subseteq \mathcal{Q}_k$ 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 $\mathcal{Q}_n$. 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 $\mathcal{Q}_k$, 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 v s.t. $v \models Q_{k+1}$, then $w :_{(k+1)} v \models q_{k+1}$.*
3. *q is a non-initial state in $\mathcal{Q}_k$. Then for any order-k stack w, we say that $w \models q$ if A_0 accepts w from q.*

We show the automaton constructed is sound with respect to this meaning. That is, for all $q_k \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$, we can place a^u, for any $u \models Q_{br}$, on top of any stack satisfying $Q_1, \ldots, Q_k$ and obtain a stack that satisfies q_k. By induction over the length of the stack, this property extends to complete stacks. That is, a stack is accepted from a state only if it is in its meaning. Since states q_p are assigned their meaning in $Pre^*_{\mathcal{C}}(A_0)$, we obtain soundness of the construction.

The construction is also sound for collapsible stacks. That is, $\langle p, w\rangle$ belongs to $Pre^*_{\mathcal{C}}(A_0)$ where $\mathcal{C}$ is a collapsible PDS and A_0 accepts collapsible stacks iff $\langle p, [\![w]\!]_n\rangle$ belongs to $Pre^*_{\mathcal{C}}(A_0)$ where $\mathcal{C}$ and A_0 are interpreted over annotated stacks. This is due to the commutativity of $[\![o(w)]\!] = o([\![w]\!])$.

Proposition 1. *The saturation construction for an alternating order-n annotated PDS $\mathcal{C}$ and an order-n stack automaton A_0 runs in n-EXPTIME, which is optimal.*

Proof. Let $2 \uparrow_0 \ell = \ell$ and $2 \uparrow_{i+1} \ell = 2^{2\uparrow_i \ell}$. 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 $\mathcal{Q}_k$ 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. Since the reachability problem for alternating higher-order pushdown systems is complete for n-EXPTIME [15], our algorithm is optimal.

It is known that the complexity of reachability for non-alternating collapsible PDS is in $(n-1)$-EXPTIME. The cause of the additional exponential blow up is in the alternation of the stack automata. In the appendix we show that, for a

suitable notion of *non-alternating* stack automata, our algorithm can be adapted to run in $(n-1)$-EXPTIME, when the collapsible PDS is also non-alternating.

Furthermore, the algorithm is PTIME for a fixed order and number of control states. If we obtained $\mathcal{C}$ from a scheme, the number of control states is given by the arity of the scheme [14]. Since the arity and order are expected to be small, we are hopeful that our algorithm will perform well in practice.

5 Perspectives

There are several avenues of future work. First, we intend to generalise our saturation technique to computing winning regions of *parity* conditions, based on the order-1 case [17]. This will permit verification of more general specifications. We also plan to design a prototype tool to test the algorithm in practice

An important direction is that of counter example generation. When checking safety property, it is desirable to provide a trace witnessing a violation of the property. This can be used to repair the bug and as part of a *counter-example guided abstraction refinement (CEGAR)* loop enabling efficient verification algorithms. However, finding *shortest* counter examples — due to its tight connection with pumping lemmas — will present a challenging and interesting problem.

Saturation techniques have been extended to concurrent order-1 pushdown systems [32, 1]; concurrency at higher-orders would be interesting.

It will also be interesting to study notions of regularity of annotated stacks. In our notion of regularity, the forwards reachability set is not regular, due to the copy operation $push_k$. This problem was addressed by Carayol for higher-order stacks [8]; adapting these techniques to annotated PDS is a challenging problem.

References

1. M. F. Atig. Global model checking of ordered multi-pushdown systems. In *FSTTCS*, pages 216–227, 2010.
2. T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *POPL*, pages 1–3, 2002.
3. M. Benois *Parties rationnelles du groupe libre. Comptes-Rendus de l'Acamdémie des Sciences de Paris*, Série A:1188–1190, 1969.
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 and A. Meyer. Symbolic Reachability Analysis of Higher-Order Context-Free Processes. In *FSTTCS*, pages 135–147 ,2004.
6. C. H. Broadbent, A. Carayol, C.-H. Luke Ong, and O. Serre. Recursion schemes and logical reflection. In *LiCS*, pages 120–129, 2010.
7. T. Cachat. *Games on Pushdown Graphs and Extensions.* PhD thesis, RWTH Aachen, 2003.
8. A. Carayol. Regular sets of higher-order pushdown stacks. In *MFCS*, pages 168–179, 2005.
9. A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, pages 112–123, 2003.
10. Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

11. W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.
12. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV*, pages 232–247, 2000.
13. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Electr. Notes Theor. Comput. Sci.* 9: 27-37, 1997.
14. M. Hague, A. S. Murawski, C.-H. Luke Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LiCS*, pages 452–461, 2008.
15. M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. *Logical Methods in Computer Science*, 4(4), 2008.
16. M. Hague and C.-H. L. Ong. Analysing mu-calculus properties of pushdown systems. In *SPIN*, pages 187–192, 2010.
17. M. Hague and C.-H. L. Ong. A saturation method for the modal μ-calculus over pushdown systems. *Inf. Comput.*, 209(5):799–821, 2010.
18. A. Kartzow and P. Parys Strictness of the Collapsible Pushdown Hierarchy. arXiv:1201.3250v1 [cs.FL], 2012.
19. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, pages 205–222, 2002.
20. T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, pages 1450–1461, 2005.
21. N. Kobayashi. Higher-order model checking: From theory to practice. In *LiCS*, pages 219–224, 2011.
22. N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *FoSSaCS*, pages 260–274, 2011.
23. A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 15:1170–1174, 1976.
24. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LiCS*, pages 81–90, 2006.
25. C.-H. L. Ong and S. J. Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *POPL*, pages 587–598, 2011.
26. P. Parys. Collapse operation increases expressive power of deterministic higher order pushdown automata. In *STACS*, pages 603–614, 2011.
27. T. W. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.*, 58(1-2):206–263, 2005.
28. S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP (2)*, pages 162–173, 2011.
29. S. Schwoon. *Model-checking Pushdown Systems.* PhD thesis, Technical University of Munich, 2002.
30. A. Seth. An alternative construction in symbolic reachability analysis of second order pushdown systems. In *RP* pages 80–95, 2007.
31. A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, pages 203–216, 2009.
32. D. Suwimonteerabuth, J. Esparza, and S. Schwoon. Symbolic context-bounded analysis of multithreaded java programs. In *SPIN*, pages 270–287, 2008.
33. D. Suwimonteerabuth, S. Schwoon, and J. Esparza. Efficient algorithms for alternating pushdown systems with an application to the computation of certificate chains. In *ATVA*, pages 141–153, 2006.

A Collapsible Pushdown Systems

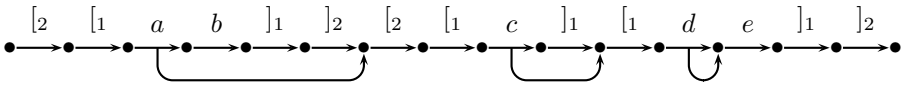
We give the definition of higher-order collapsible stacks and their operations, before giving the definition of collapsible pushdown systems.

A.1 Higher-Order Collapsible Stacks

Higher-order collapsible stacks are built from a stack alphabet Σ and form a nested "stack-of-stacks" structure. Furthermore, each stack character contains a pointer — called a "link" — to a position lower down in the stack. When building stacks from the stack operations defined below, copies of sub-stacks will be made. The link is intuitively a pointer to the context in which the stack character was first created. We define collapse links formally below.

Definition 6 (Order-n Collapsible Stacks). *Given a finite set of stack characters Σ, an* order-0 stack *is simply a character $a \in \Sigma$. An* order-n stack *is a sequence $w = [w_1 \dots w_\ell]_n$ such that each w_i is an order-$(n-1)$ stack and each character on the stack is augmented with a collapse link. Let $Stacks_n$ denote the set of order-n stacks.*

An order-n stack can be represented naturally as an edge-labelled word-graph over the alphabet $\{[_{n-1}, \dots, [_1,]_1, \dots,]_{n-1}\} \uplus \Sigma$, with additional collapse-links pointing from a stack character in Σ to the beginning of the graph representing the target of the link. An example order-3 stack is given below, with only a few collapse links shown. The collapse links range from order-3 to order-1 respectively.



Given an order-n stack $[w_1 \dots w_\ell]_n$, we define

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

noting that $top_k(w)$ is undefined if $top_{k'}(w)$ is empty for any $k' > k$. We also remove the top portion of a top_k stack using

$$\begin{aligned} bot^i_n([w_1 \dots w_\ell]_n) &= [w_{\ell-i+1} \dots w_\ell]_n && \text{when } i \leq \ell \text{ and } \ell > 0 \\ bot^i_k([w_1 \dots w_\ell]_n) &= [bot^i_k(w_1) w_2 \dots w_\ell]_n && \text{when } k < n \text{ and } \ell > 0 \ . \end{aligned}$$

Formally, then, a collapse link is a pair (k, i) where $1 \leq k \leq n$ and $i > 0$. For $top_1(w) = a$ where a has the link (k, i), the destination of the link is $bot^i_k(w)$. We disallow collapse links where bot^i_k does not lead to a valid stack. The example stack above is thus $[[[a^{(3,1)}b]_1]_2[[c^{(2,1)}]_1]_2[[d^{(1,1)}e]_1]_2]_3$, where collapse links are denoted as superscripts. Often, we will omit these superscripts for readability.

A.2 Operations on Order-n Collapsible Stacks

The following operations may be performed on an order-n collapsible stack.

$$\begin{aligned}\mathcal{O}_n = \{&pop_1, \ldots, pop_n\} \cup \\ &\{push_2, \ldots, push_n\} \cup \\ &\{collapse_2, \ldots, collapse_n\} \cup \\ &\{\ push^1_a, \ldots, push^n_a, rew_a \mid a \in \Sigma\ \}\end{aligned}$$

We say $o \in \mathcal{O}_n$ is of order-k when k is minimal such that $o \in \mathcal{O}_k$. For example, $push_k$ is of order k.

The $collapse_k$ operation is non-standard in the sense of Hague *et al.* [14] and has the semantics of a normal collapse, with the additional constraint that the top character has an order-k link. The standard version of collapse can be simulated with a non-deterministic choice on the order of the stack link. In the other direction, we can store in the stack alphabet the order of the collapse link attached to each character on the stack.

We define each stack operation in turn for an order-n stack w. Collapse links are created by the $push^k_a$ operations, which add a character to the top of a given stack w with a link pointing to $pop_k(w)$.

1. We set $pop_k(w) = v$ when w decomposes into $u :_k v$ for a non-empty u.
2. We set $push_k(w) = u :_k u :_k v$ when $w = u :_k v$.
3. We set $collapse_k(w) = bot^i_k(w)$ where $top_1(w) = a^{(k,i)}$ for some i.
4. We set $push^k_b(w) = b^{(k,\ell-1)} :_1 w$ where $top_{k+1}(w) = [w_1 \ldots w_\ell]_{k+1}$.
5. We set $rew_b(w) = b^{(k,i)} :_1 v$ where $w = a^{(k,i)} :_1 v$.

Note that, for a $push_k$ operation, links outside of $u = top_k(w)$ point to the same destination in both copies of u, while links pointing within u point within the respective copies of u. For full introduction, we refer the reader to Hague *et al.* [14]. In Section 3.2 we give several example stacks and show how the stack operations affect them.

A.3 Collapsible Pushdown Systems

We are now ready to define alternating collapsible pushdown systems.

Definition 7 (Collapsible Pushdown Systems). *An alternating order-n* collapsible pushdown system (collapsible PDS) *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 $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P}) \cup \left(\mathcal{P} \times 2^{\mathcal{P}}\right)$ is a set of rules.*

We write *configurations* of a collapsible PDS as a pair $\langle p, w\rangle$ where $p \in \mathcal{P}$ and $w \in Stacks_n$. We write $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ to denote a transition from a rule (p, a, o, p') with $top_1(w) = a$ and $w' = o(w)$. Furthermore, we have a transition $\langle p, w\rangle \longrightarrow \{\ \langle p', w\rangle \mid p' \in P\ \}$ whenever we have a rule $p \to P$. A non-alternating collapsible PDS has no rules of this second form. We write C to denote a set of configurations.

B Stack Automata

We present omitted proofs and definitions for stack automata. We remark that both the definition of a run, and the test for membership, can easily be adapted to collapsible stacks by simply following the collapse links rather than the annotations.

B.1 Additional Notation

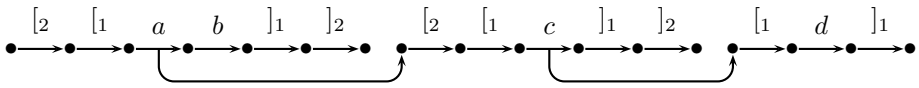
For $n \geq k > 1$, we write $Q_1 \xrightarrow{Q'} Q_2$ to denote an order-k transition from a set of states whenever $Q_1 = \{q_1, \ldots, q_\ell\}$ and for each $1 \leq i \leq \ell$ we have $q_i \xrightarrow{q'_i} Q_i$ and $Q' = \{q'_1, \ldots, q'_\ell\}$ and $Q_2 = \bigcup_{1 \leq i \leq \ell} Q_i$. The analogous notation at order-1 is a special case of the short-form notation defined in Section 3.1.

B.2 Runs of Stack Automata

Formally, fix a stack automaton

$$A = (\mathcal{Q}_n, \ldots, \mathcal{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1) \ .$$

We say a node *contains* a character if its exiting edge is labelled by the character. Recall the tree view of an annotated stack, an example of which is given below.



Some stack (tree) w is accepted by A from states $Q_0 \subseteq \mathcal{Q}_k$ — written $w \in \mathcal{L}_{Q_0}(A)$ — whenever the nodes of the tree can be labelled by elements of $\bigcup_{1 \leq k' \leq n} 2^{\mathcal{Q}_{k'}}$ such that

1. Q_0 is a subset of the label of the node containing the first $[_{k-1}$ character of the word, or if $k = 1$, the first character $a \in \Sigma$, and
2. for any node containing a character $[_{k'}$ labelled by Q, then for all $q_1 \in Q$, there exists some transition $(q_1, q_2, Q_1) \in \Delta_{k'+1}$ such that q_2 appears in the label of the succeeding node and Q_1 is a subset of the label of the node succeeding the matching $]_{k'}$ character, and
3. for any node containing a character $]_{k'}$, the label Q is a subset of $\mathcal{F}_{k'}$, and the final node of an order-k stack is labelled by $Q \subseteq \mathcal{F}_k$, and
4. for any node containing a character $a \in \Sigma$, labelled by Q, for all $q' \in Q$, there exists some transition $(q', a, Q_{br}, Q') \in \Delta_1$ such that Q_{br} is a subset of the label of the node annotating a, and Q' is a subset of the label of the succeeding node.

That is, a stack automaton is essentially a stack- and annotation-aware alternating automaton, where annotations are treated as special cases of the alternation.

Proposition 2 (Stack Automata Membership). *Membership of order-n stack automata can be tested in linear time in the size of the input stack and stack automaton.*

Proof. Take a stack w and let

$$A = (\mathcal{Q}_n, \ldots, \mathcal{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1) \ .$$

The membership algorithm iterates from the bottom (end) of the stack and its annotations to the top (beginning). Take the graph representing w. We start by labelling the final node with the set $\mathcal{F}_n$. It is easy to verify at all stages that if the label of a node contains a state q, then the stack from that node is in $\mathcal{L}_q(A)$. Now, suppose we have labelled up to a given node. For convenience, we will refer to nodes by their labelled state set, and we show, by cases, how to label the preceding node with a state-set Q_0.

In the first case, we have node Q_1 connected to Q_0 by a $]_k$ character. That is

$$Q_0 \xrightarrow{]_k} Q_1 \qquad \cdots$$

where Q_0 is the set $\mathcal{F}_k$. This is because Q_0 labels the end of an order-k stack.

In the next case the connection is by a character a with an annotation and we have

$$Q_0 \xrightarrow{a} Q_1 \qquad \cdots \qquad Q_{br} \qquad \cdots \ .$$

where $Q_0 = \left\{ q \;\middle|\; q \xrightarrow[Q'_{br}]{a} Q'_1 \in \Delta_1 \wedge Q'_1 \subseteq Q_1 \wedge Q'_{br} \subseteq Q_{br} \right\}$. Thus, any state in Q_0 has a transition to states from which the remainder of the stack is accepted.

In the final case we have

$$Q_0 \xrightarrow{[_k} Q_1 \qquad \cdots \xrightarrow{]_k} Q_2 \qquad \cdots$$

where $]_k$ matches $[_k$. We define $Q_0 = \left\{ q \;\middle|\; q \xrightarrow{q'} Q \in \Delta_{k+1} \wedge q' \in Q_1 \wedge Q \subseteq Q_2 \right\}$.

Thus, there is a state $q \in Q_0$ whenever there is a transition $q \xrightarrow{q'} Q$ such that the next order-k stack is accepted from q' and the remainder of the stack is accepted from Q.

Thus, after this labelling, we can test whether $w \in \mathcal{L}_{Q_0}(A)$ for $Q_0 \subseteq \mathcal{Q}_k$ by checking whether $Q_0 \subseteq Q$ where Q labels the node containing the first $[_{k-1}$ character of the word, or if $k = 1$, the first character $a \in \Sigma$.

We now show that emptiness checking is PSPACE-complete.

Proposition 3 (Emptiness of Stack Automata). *Given an order-n stack automaton A and a state q of A, deciding if there exists some annotated stack $w \in \mathcal{L}_q(A)$ is PSPACE-complete.*

Proof. The problem is already PSPACE-hard for alternating word automata which correspond to the order-1 case [10]. We now establish the upperbound.

Let $A = (\mathcal{Q}_n, \ldots, \mathcal{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1)$ be a stack automaton. We use a fixed point to compute the set $\mathcal{Q}_{ac} := \{\, q \in \mathcal{Q}_k \mid \mathcal{L}_q(A) \neq \emptyset \,\}$.

For this, we take Q_0 to be the set of all final states. For all $i \geq 0$, we define Q_{i+1} by adding to Q_i all states $q \in \mathcal{Q}_k$ such that:

- for $k > 1$, there exists a transition $q \xrightarrow{q'} Q \in \Delta_k$ with $Q \subseteq Q_i$ and $q' \in Q_i$.
- for $k = 1$, there exists a transition $q \xrightarrow[Q_{br}]{a} Q \in \Delta_1$ with $Q, Q_{br} \subseteq Q_i$.

As the sequence of the Q_i is increasing (for inclusion), we have $Q_{j+1} = Q_j$ for some index $j \geq 0$. We claim that Q_j is the set Q_{ac}. A straightforward induction shows that $Q_i \subseteq Q_{ac}$ and hence $Q_j \subseteq Q_{ac}$. Assume toward a contradiction, that the converse inclusion does not hold. Let w be smallest stack accepted by a state $q \in \mathcal{Q}_k \setminus Q_j$. If $k = 1$, then $w = a^u :_1 w$ and there exists a transition $q \xrightarrow[Q_{br}]{a} Q \in \Delta_1$ with w' accepted from Q and u is accepted from Q_{br}. As q does not belongs to Q_j, then either u or w' is accepted from a state not in Q_j. As u and w' are both smaller than w, this contradictes the definition of w. The case $k > 1$ is similar.

The algorithm to test emptiness from a given state $p \in \mathcal{Q}_n$ consists of computing Q_j by iteratively computing the Q_i and then checking if p belongs to Q_j.

B.3 Comparison with Broadbent *et al.* [6]

We show that our notion of stack automata, when viewed as acceptors of collapsible stacks, accept the same family of stacks as the automata introduced by Broadbent *et al.* in LICS 2010 [6]. We begin by defining these models, which for the purposes of comparison, we will call *bottom-up stack automata.* Let $\Sigma' = \{[_{n-1}, \ldots, [_1,]_1, \ldots,]_{n-1}\} \uplus \Sigma$.

Definition 8 (Bottom-up Stack Automata). *A* bottom-up stack automaton *B is a tuple $(\mathcal{Q}, \Sigma', q_{in}, \mathcal{F}, \Delta)$ where $\mathcal{Q}$ is a finite set of states, Σ' is a finite input alphabet, $q_{in} \in \mathcal{Q}$ is the initial state and $\Delta : (\mathcal{Q} \times \Sigma') \cup (\mathcal{Q} \times \Sigma' \times \mathcal{Q}) \to \mathcal{Q}$ is a deterministic transition function.*

Representing collapsible stacks as word graphs, a run of a bottom-up stack automaton is a labelling of the graph with states in $\mathcal{Q}$ such that

1. The rightmost (final) node is labelled by q_{in}.
2. The leftmost (initial) node is labelled by $q \in \mathcal{F}$.
3. Whenever we have for any $a \in \Sigma'$, and pair of labelled nodes with an edge

$$\cdots \quad q \xrightarrow{a} q' \quad \cdots$$

then $q = \Delta(q', a)$.

4. Whenever we have for any $a \in \Sigma'$, and triple of labelled nodes with an edge

$$\begin{tikzcd} \cdots & q \arrow[r, "a"] \arrow[rr, bend right] & q' & \cdots & q_{br} & \cdots \end{tikzcd}$$

then $q = \Delta(q', a, q_{br})$.

An example run over an order-2 stack is given below

$$\begin{tikzcd} q_f \arrow[r, "{[_1}"] & q_1 \arrow[r, "a"] \arrow[rrrrrrr, bend right] & q_1^1 \arrow[r, "b"] & q_1^2 \arrow[r, "{]_1}"] & q_2^1 \arrow[r, "{[_1}"] & q_1^3 \arrow[r, "a"] \arrow[rrrr, bend right] & q_1^4 \arrow[r, "b"] & q_1^5 \arrow[r, "{]_1}"] & q_{in} \end{tikzcd}$$

where $q_f \in \mathcal{F}$.

We prove the following proposition.

Proposition 4 (Equivalence of Stack Automata). *For every order-n stack automaton A with initial state q, there is a bottom-up stack automaton B with initial state q' such that $\mathcal{L}_q(A) = \mathcal{L}_{q'}(B)$ and vice-versa.*

There are two directions. We begin by constructing a stack automaton from a bottom-up stack automaton. Intuitively this construction implements the membership algorithm as a bottom up automaton. This labels each position of a candidate stack bottom-up (right-to-left) with sets of states from which the top-down automaton would have an accepting run. The states of the bottom-up automaton will therefore be sets of states of the top-down version, creating at worst an exponential blow-up.

Definition 9 (Bottom-Up Automata to Stack Automata). *Given a bottom-up stack automaton $B = (\mathcal{Q}, \Sigma', q_{in}, \mathcal{F}, \Delta)$, we define*

$$A_B = (\mathcal{Q}_n, \ldots, \mathcal{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1) \ .$$

First we stratify the states of $\mathcal{Q}$ into

$$\begin{array}{l} \mathcal{F}'_k = \{ \ q \in \mathcal{Q} \ \mid \exists q' \in \mathcal{Q}.\, q = \Delta(q',]_k) \ \} \text{ where } 1 \leq k < n \\ \mathcal{F}'_n = \{q_{in}\} \end{array}$$

and

$$\begin{array}{ll} \mathcal{Q}'_k = \{ \ q \in \mathcal{Q} \ \mid \exists q' \in \mathcal{Q}.\, q = \Delta(q', a, [_{k-1}) \ \} \cup \mathcal{F}'_k & \text{when } 1 < k \leq n \\ \mathcal{Q}'_1 = \{ \ q \in \mathcal{Q} \ \mid \exists q', q_{br} \in \mathcal{Q}, a \in \Sigma.\, q = \Delta(q', a, q_{br}) \ \} \cup \mathcal{F}'_1 \ . & \end{array}$$

Then we define the states of A_B to be $\mathcal{Q}_k = \mathcal{Q}'_k \times \cdots \times \mathcal{Q}'_n$ for all $1 \leq k < n$ and $\mathcal{Q}_n = \mathcal{Q}'_n \cup \{q_0\}$ where q_0 is a fresh state, with $\mathcal{F}_n = \mathcal{F}'_n \cup \{ \ q_0 \ \mid q_{in} \in \mathcal{F} \ \}$ and

$$\mathcal{F}_k = \{ \ (q_k, q_{k+1}, \ldots, q_n) \ \mid q_k = \Delta(q_{k+1},]_k) \ \}$$

for all $1 \leq k < n$. That is, we have to keep track of where to return to after the completion of a stack. We need to keep complete information when handling the collapse links.

We then have the transition relations

$$\Delta_1 = \left\{ (q_1, q_2, \ldots, q_n) \xrightarrow[\{(q_{br}, q_{k+1}, \ldots, q_n)\}]{a} \{(q'_1, q_2, \ldots, q_n)\} \;\middle|\; \begin{array}{l} q_1 = \Delta(q'_1, a, q_{br}) \\ \wedge\ q_{br} \in \mathcal{Q}'_k \end{array} \right\}$$

and when $1 < k < n$ *we have* $\Delta_k =$

$$\left\{ (q_k, q_{k+1}, \ldots, q_n) \xrightarrow{(q_{k-1}, q'_k, q_{k+1}, \ldots, q_n)} \{(q'_k, q_{k+1}, \ldots, q_n)\} \;\middle|\; \begin{array}{c} q_k = \Delta(q_{k-1}, [_{k-1}) \\ \wedge \\ q'_k \in \mathcal{Q}'_k \end{array} \right\}$$

and

$$\begin{aligned} \Delta_n = & \left\{ q_n \xrightarrow{(q_{n-1}, q'_n)} \{q'_n\} \mid q_n = \Delta(q_{n-1}, [_{n-1}) \wedge q'_n \in \mathcal{Q}'_n \right\} \cup \\ & \left\{ q_0 \xrightarrow{(q_{n-1}, q'_n)} \{q'_n\} \mid \exists q_n = \Delta(q_{n-1}, [_{n-1}) \wedge q_n \in \mathcal{F} \wedge q'_n \in \mathcal{Q}'_n \right\} \end{aligned}$$

We then have

Lemma 1 (Correctness of A_B). *For a given bottom-up stack automaton B with initial state q_{in}, for every order-n collapsible stack w we have $w \in \mathcal{L}_{q_{in}}(B)$ iff $[\![w]\!] \in \mathcal{L}_{q_0}(A_B)$.*

Proof. There are two directions: take an accepting run of B and show how to construct an accepting run of A_B, then consider the opposite direction.

1. Take the run graph $\mathcal{R}$ over any accepted stack w. We take an unlabelled graph of w which we will label with an accepting run of A_B, proceeding from left to right.
   In the case when w is empty, $\mathcal{R}$ has a single node labelled q_{in}. For this to be accepting, q_{in} must be final. Hence q_0 is final, and we build the run graph whose single node is labelled q_0.
   Otherwise, we label the leftmost node q_0 and we have in $\mathcal{R}$ the labelling $q_n \xrightarrow{[_{n-1}} q_{n-1}$ with $q_n = \Delta(q_{n-1}, [_{n-1})$ and $q_n \in \mathcal{F}$. Let q'_n be the labelling in $\mathcal{R}$ after the matching $]_{n-1}$. We use the transition $q_0 \xrightarrow{(q_{n-1}, q'_n)} \{q'_n\} \in \Delta_n$ to extend the run of A_B.
   We thus maintain the invariant that a node is labelled by $(q_k, q_{k+1}, \ldots, q_n)$ where q_k labels the corresponding node in $\mathcal{R}$ and $q_{k+1}, \ldots, q_n$ are the labels in $\mathcal{R}$ occurring immediately after the next $]_k, \ldots,]_{n-1}$ respectively.
   Suppose from the current node we have $q_k \xrightarrow{[_{k-1}} q_{k-1}$ in $\mathcal{R}$ with $q_k = \Delta(q_{k-1}, [_{k-1})$. Let q'_k be the labelling in $\mathcal{R}$ immediately after the matching $]_{k-1}$. From the invariant we extend the run being built from $(q_k, q_{k+1}, \ldots, q_n)$. We use the transition

   $$(q_k, q_{k+1}, \ldots, q_n) \xrightarrow{(q_{k-1}, q'_k, q_{k+1}, \ldots, q_n)} \{(q'_k, q_{k+1}, \ldots, q_n)\}$$

   to do so, which maintains the invariant.
   In the next case, we have in $\mathcal{R}$

$$q_1 \xrightarrow{a} q_1' \quad \cdots \quad q_{br} \quad \cdots \ .$$

and the node in the run we're constructing is labelled by $(q_1, q_2, \ldots, q_n)$. We use the transition

$$(q_1, q_2, \ldots, q_n) \xrightarrow[\{(q_{br}, q_{k+1}, \ldots, q_n)\}]{a} \{(q_1', q_2, \ldots, q_n)\}$$

which maintains the invariant.

Finally, when we have in $\mathcal{R}$ the edge $q_1 \xrightarrow{]_k} q_2$, then our current node is labelled $(q_k, q_{k+1}, \ldots, q_n)$ with $q_k = \Delta(q_{k+1},]_k)$, hence the labelling is in $\mathcal{F}_k$. This means the current stack is accepted, and from the above we know the next node is labelled by $(q_{k+1}, \ldots, q_n)$ as required to maintain the invariant. The rightmost node of the run constructed is necessarily labelled q_{in}, giving us an accepting run as required.

2. Next, we take an accepting run over any w of A_B and construct an accepting run of B. The proof relies on a key observation: the labelling of each node may consist of a single state only. We prove this by backwards induction over the run graph $\mathcal{R}$ of A_B. The rightmost node is necessarily labelled q_0 for empty w or q_{in} otherwise. Since the former case is trivial, assume the label is q_{in}. This is the *unique* labelling such that acceptance can occur.

   Now, induct right to left. Assume we have $Q \xrightarrow{]_{k-1}} (q_k, \ldots, q_n)$ where the label $(q_k, \ldots, q_n)$ is the unique label from which the rest of the run can be accepting. From the determinism of B, there is one $q_{k-1} = \Delta(q_k,]_{k-1})$. Thus we must have $Q = \{(q_{k-1}, q_k, \ldots, q_n)\}$ being the only possible labelling of the new node.

   In the next case we have in $\mathcal{R}$

$$Q \xrightarrow{a} (q_1', q_2, \ldots, q_n) \quad \cdots \quad (q_{br}, q_{k+1}, \ldots, q_n) \quad \cdots \ .$$

   noting that the labellings must be consistent in $q_{k+1}, \ldots, q_n$. (This can be checked by following this induction.) From the determinism of B, there is a unique $q_1 = \Delta(q_1', a, q_{br})$. This implies Q can only be $\{(q_1, q_2, \ldots, q_n)\}$.

   In the next case we have the case of $Q \xrightarrow{[_k} \big(q_k, q_{k+1}', q_{k+2}, \ldots, q_n\big)$. From the determinism of B we have a unique $q_{k+1} = \Delta(q_k, [_k)$. Hence, it must be the case that $Q = \{(q_{k+1}, q_{k+2}, \ldots, q_n)\}$.

   Finally, we look at the leftmost node. We have $q_0 \xrightarrow{[_{n-1}} (q_{n-1}, q_n')$ and we know there is $q_n \in \mathcal{F}$ such that $q_n = \Delta(q_{n-1}, [_{n-1})$.

   Thus, we construct an accepting run of B by taking the labelling of $\mathcal{R}$ and projecting it to the first component of each tuple, labelling the leftmost node with q_n.

Now we consider the translation from stack automata to bottom-up stack automata.

Definition 10 (Stack Automata to Bottom-Up Automata). *Given an order-n stack automaton*

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

and $q_{in} \in \mathcal{Q}_n$*, we construct a bottom-up stack automaton* $B_A = (\mathcal{Q}, \Sigma', \mathcal{F}_n, \mathcal{F}, \Delta)$ *where* $\mathcal{F} = \{\ Q \subseteq \mathcal{Q}_n \mid q_{in} \in Q\ \}$ *and*

$$\mathcal{Q} = \bigcup_{1 \leq k \leq n} \left(2^{\mathcal{Q}_k} \times \cdots \times 2^{\mathcal{Q}_n}\right) .$$

This structure is needed to flatten the nested structure of the automaton.

We define Δ *to be the smallest set containing*

1. *For all* $1 < k \leq n$ *and* $Q_k \subseteq \mathcal{Q}_k, \ldots, Q_n \subseteq \mathcal{Q}_n$ *we have the transition* $((\mathcal{F}_{k-1}, Q_k, \ldots, Q_n),]_{k-1}, (Q_k, \ldots, Q_n))$*. Note* Q_k *is stored to know which transitions to use on reading the matching* $[_{k-1}$*, and*
2. *For all* $a \in \Sigma$*,* $1 \leq k \leq n$ *and* $Q_2 \subseteq \mathcal{Q}_2, \ldots, Q_n \subseteq \mathcal{Q}_n$ *and* $Q_{br} \subseteq \mathcal{Q}_k$ *we have the transition* $\left((Q^1, Q_2, \ldots, Q_n), a, (Q_{br}, Q_{k+1}, \ldots, Q_n), (Q^2, Q_2, \ldots, Q_n)\right)$ *where* Q^1 *is maximal such that* $Q^1 \xrightarrow[Q'_{br}]{a} Q'$ *and* $Q'_{br} \subseteq Q_{br}$ *and* $Q' \subseteq Q^2$*, and*
3. *For all* $1 \leq k < n$ *and* $Q_k \subseteq \mathcal{Q}_k, \ldots, Q_n \subseteq \mathcal{Q}_n$ *we have that there is a transition* $((Q, Q_{k+2}, \ldots, Q_n), [_k, (Q_k, Q_{k+1}, Q_{k+2}, \ldots, Q_n))$ *whenever* Q *is maximal such that* $Q \xrightarrow{Q'_k} Q'_{k+1} \in \Delta_{k+1}$ *with* $Q'_k \subseteq Q_k$ *and* $Q'_{k+1} \subseteq Q_{k+1}$*.*

We then have

Lemma 2 (Correctness of B_A). *For a given order-n stack automaton* A *and state* $q_{in} \in \mathcal{Q}_n$*, for every order-n collapsible stack* w *we have* $w \in \mathcal{L}_{q_{in}}(A)$ *iff* $w \in \mathcal{L}_{\mathcal{F}_n}(A_B)$*.*

Proof. There are two directions – first we translate an accepting run of A over any w into an accepting run of B_A, then vice-versa.

1. Take any stack w and an accepting run $\mathcal{R}$ of A. From $\mathcal{R}$ we construct an accepting run of B_A. We proceed by induction from the rightmost node of $\mathcal{R}$, which we label with $\mathcal{F}_n$. We now consider the inductive cases.

   Suppose from the current node we have $Q_{k-1} \xrightarrow{]_{k-1}} Q_k$ in $\mathcal{R}$. By induction, assume the node corresponding to Q_k in the run being constructed is labelled $(Q^r_k, \ldots, Q^r_n)$ where $Q^r_k \supseteq Q_k, \ldots, Q^r_n \supseteq Q_n$, where $Q_k, \ldots, Q_n$ are the labels in $\mathcal{R}$ immediately following the next $]_{k-1}, \ldots,]_{n-1}$ characters respectively. From the construction of B_A, we have a transition $\mathcal{F}_{k-1} = \Delta((Q^r_k, \ldots, Q^r_n),]_{k-1})$ and since $\mathcal{R}$ is accepting, we know $\mathcal{F}_{k-1} \supseteq Q_{k-1}$. Hence, we label the node corresponding to Q_{k-1} with $(\mathcal{F}_{k-1}, Q^r_k, \ldots, Q^r_n)$, maintaining the induction hypothesis.

   In the next case, we have in $\mathcal{R}$

$$Q'_1 \xrightarrow{a} Q_1 \quad \cdots \quad Q_{br} \quad \cdots$$

for some order-k link and the latter two corresponding nodes in the run we're constructing are labelled by $(Q^r_1, Q^r_2, \ldots, Q^r_n)$ and $\left(Q^r_{br}, Q^r_{k+1}, \ldots, Q^r_n\right)$ with $Q^r_1 \supseteq Q_1, \ldots, Q^r_n \supseteq Q_n$where $Q_2, \ldots, Q_n$ are the labels in $\mathcal{R}$ immediately following the next $]_{k-1}, \ldots,]_{n-1}$ characters respectively and $Q^r_{br} \supseteq Q_{br}$. From the construction, we have the transition

$$\left((Q, Q^r_2, \ldots, Q^r_n), a, \left(Q^r_{br}, Q^r_{k+1}, \ldots, Q^r_n\right), (Q^r_1, Q^r_2, \ldots, Q^r_n)\right)$$

where, since Q is maximal, we have $Q \supseteq Q'_1$.

Finally, when we have in $\mathcal{R}$ the edge $Q_{k+1} \xrightarrow{[_k} Q_k$ we know by induction that the node corresponding to the latter node is labelled by $\left(Q^r_k, Q^r_{k+1}, \ldots, Q^r_n\right)$ with the superset property as in the previous cases. From the construction, we have the transition

$$\left(\left(Q, Q^r_{k+2}, \ldots, Q^r_n\right), [_k, \left(Q^r_k, Q^r_{k+1}, Q^r_{k+2}, \ldots, Q^r_n\right)\right)$$

and since Q is maximal, we have $Q \supseteq Q_{k+1}$.

Thus, when we reach the leftmost node, we have a label Q with $q_{in} \in Q$. Hence, $Q \in \mathcal{F}$ and the run is accepting.

2. Take any stack w and an accepting run $\mathcal{R}$ of B_A. For any node whose label has the first component Q, we prove for any state $q \in Q$ there is an accepting run of A from the node. The run begins with q.

   We proceed by induction. In the base case, take the rightmost node. This is labelled by $\mathcal{F}_n$, hence the result is immediate. We now consider the inductive cases.

   Suppose we have $(Q_{k-1}, Q_k, \ldots, Q_n) \xrightarrow{]_{k-1}} (Q_k, \ldots, Q_n)$ in $\mathcal{R}$. From the construction of B_A, we have $Q_{k-1} = \mathcal{F}_{k-1}$ thus an accepting run from all states in $\mathcal{F}_{k-1}$ (by definition of accepting runs from states in Q_{k-1}, there is nothing more to satisfy).

   In the next case, we have in $\mathcal{R}$

$$(Q'_1, Q_2, \ldots, Q_n) \xrightarrow{a} (Q_1, Q_2, \ldots, Q_n) \quad \cdots \quad (Q_{br}, Q_{k+1}, \ldots, Q_n) \quad \cdots$$

   for some order-k link. For every $q \in Q'_1$ we have $q \xrightarrow[Q'_{br}]{a} Q''_1 \in \Delta_1$, and accepting runs from all states in $Q'_{br} \subseteq Q_{br}$ and $Q''_1 \subseteq Q'_1$. Thus, we have an accepting run from q.

   Finally, when we have in $\mathcal{R}$ the edge

$$(Q, Q_{k+2}, \ldots, Q_n) \xrightarrow{[_k} (Q_k, Q_{k+1}, Q_{k+2}, \ldots, Q_n)$$

   we know by construction of B_A that the node occurring after the matching $]_k$ is labelled $(Q_{k+1}, \ldots, Q_n)$. Also by construction of B_A we know for all

$q \in Q$ that we have a transition $q \xrightarrow{q'} Q'$ with $q' \in Q_k$ and $Q' \subseteq Q_{k+1}$. By induction, we have accepting runs from q' and Q', giving an accepting run from q.
Thus, when we reach the leftmost node, we have a label $Q \in \mathcal{F}$, hence $q_{in} \in Q$. Hence, we have an accepting run of A from q_{in}.

B.4 Effective Boolean Algebra

Proposition 5 (Effective Boolean Algebra for Languages of Stack Automata). *The set of annotated stacks accepted by stack automata form an effective Boolean algebra.*

Proof (sketch). The automata constructions follow standard techniques. Fix a stack automaton

$$A = (\mathcal{Q}_n, \ldots, \mathcal{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1) \ .$$

There are three cases.

1. Take any two states q_1 and q_2 both elements of the same $\mathcal{Q}_k$. We can introduce a new state q that accepts the intersection of q_1 and q_2.
   At order-1, we make $q \in \mathcal{F}_1$ iff $q_1, q_2 \in \mathcal{F}_1$. Then, for each $\{q_1, q_2\} \xrightarrow[Q_{br}]{a} Q$, introduce a transition $q \xrightarrow[Q_{br}]{a} Q$.
   At order-k for $1 < k \leq n$, we can permit transitions $q \xrightarrow{Q'} Q$ by constructing a state q' accepting the intersection of the states in Q' and introducing the transition $q \xrightarrow{q'} Q$. Thus, to form the intersection of q_1 and q_2 we set $q \in \mathcal{F}_k$ iff $q_1, q_2 \in \mathcal{F}_k$ and for each transition $\{q_1, q_2\} \xrightarrow{Q'} Q$, add the transition $q \xrightarrow{Q'} Q$.
2. To form the union of q_1 and q_2, we again introduce a new state q.
   At order-1 we set $q \in \mathcal{F}_1$ iff $q_1 \in \mathcal{F}_1$ or $q_2 \in \mathcal{F}_1$, then for each transition $q_1 \xrightarrow[Q_{br}]{a} Q$ or $q_2 \xrightarrow[Q_{br}]{a} Q$, add the transition $q \xrightarrow[Q_{br}]{a} Q$.
   At order-k for $1 < k \leq n$ we set $q \in \mathcal{F}_k$ iff $q_1 \in \mathcal{F}_k$ or $q_2 \in \mathcal{F}_k$, then for each transition $q_1 \xrightarrow{q'} Q$ or $q_2 \xrightarrow{q'} Q$, add the transition $q \xrightarrow{q'} Q$.
3. Finally, to negate an automaton, we introduce for every state q it's negation $\overline{q}$. We also introduce q_k^* states which accept any stack of order-k.
   For $q \in \mathcal{Q}_1$, we set $\overline{q} \in \mathcal{F}_1$ iff $q \notin \mathcal{F}_1$. Then, for every a, let $q \xrightarrow[Q_{br}^1]{a} Q_1, \ldots, q \xrightarrow[Q_{br}^\ell]{a} Q_\ell$ be the transitions over a from q. We take each subset $I \subseteq \{1, \ldots, \ell\}$ in turn and for every Q_{br} and Q such that
   - if $i \in I$, then there is some $q_{br} \in Q_{br}^i \cap \mathcal{Q}_{k'}$ (for some k') with either $\overline{q_{br}} \in Q_{br}$ or $q_{k''}^* \in Q_{br}$ where $k'' \neq k'$, and
   - if $i \notin I$, then there is some $q' \in Q_i$ with $\overline{q'} \in Q$,

we introduce a transition $\overline{q} \xrightarrow[Q_{br}]{a} Q$ when Q_{br} is coherent with respect to the order of the annotation. That is, every transition from q either rejects by the annotation branch (by being of the wrong order, or the right order, but the wrong shape), or by the remainder of the order-1 stack.
Finally, when $q \in \mathcal{Q}_k$ for $1 < k \leq n$, we set $\overline{q} \in \mathcal{F}_k$ iff $q \notin \mathcal{F}_k$. Then, let $q \xrightarrow{q'_1} Q_1, \ldots, q \xrightarrow{q'_\ell} Q_\ell$ be the transitions from q. Next, for every subset $I \subseteq \{1, \ldots, \ell\}$ and for every Q' and Q such that
- if $i \in I$, then $\overline{q'_i} \in Q'$, and
- if $i \notin I$, then there is some $q' \in Q_i$ with $\overline{q'} \in Q$,

we introduce a transition $\overline{q} \xrightarrow{Q'} Q$. That is, every transition from q either rejects by the top order-$(k-1)$ stack, or by the remainder of the order-k stack.

C Soundness

In this section, all stack automata are such that their initial states are not final. This is assumed for the automaton A_0 in Section 3.1. This property is preserved by the saturation function Γ.

We start by assigning 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 $q \in \mathcal{Q}_k$, which is denoted $w \models q$.

Definition 11 ($w \models q$). *For any $Q \subseteq \mathcal{Q}_k$ 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 $\mathcal{Q}_n$. 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 $\mathcal{Q}_k$, 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 $\mathcal{Q}_k$. 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 $q_k \in \mathcal{Q}_k$ with $q \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ if for any order-$(k+1)$ stack $w_{k+1} \models Q_{k+1}$, ..., and any order-n stack $w_n \models Q_n$, we have $w_k :_{(k+1)} \cdots :_n w_n \models q$.

Definition 12 (Soundness of transitions). *A transition $q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$ is sound if for any order-1 stack $w_1 \models Q_1$, ..., and any order-k stack $w_k \models Q_k$ and any stack $u \models Q_{br}$, we have $a^u :_1 w_1 :_2 \cdots :_k w_k \models q$.*

Lemma 3. *If $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$ is sound, then any transition $q_k \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$ contained within the transition from q_p is sound.*

Proof. The proof is by induction over k. In the base case, $k = n$, then $q_k = q_p$ and the result is immediate. Otherwise, assume $q_{k+1} \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_{k+1})$ is sound — where q_{k+1} or $q_{k+2} \xrightarrow{q_{k+1}} (Q_{k+2}, \ldots, Q_n)$ — and $q_{k+1} \xrightarrow{q_k} Q_{k+1}$. We need to show $q_k \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$ is sound. Hence, take any $w_1 \models Q_1, \ldots, w_k \models Q_k$ and $u \models Q_{br}$. By definition of $\models$, we have $a^u :_1 w_1 :_2 \cdots :_k w_k \models q_k$ if, for all $w_{k+1} \models Q_{k+1}$, we have $a^u :_1 w_1 :_2 \cdots :_{(k+1)} w_{k+1} \models q_{k+1}$. Since $w_1 \models Q_1, \ldots, w_{k+1} \models Q_{k+1}$ and $q_{k+1} \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_{k+1})$ is sound, we have $a^u :_1 w_1 :_2 \cdots :_{(k+1)} w_{k+1} \models q_{k+1}$ and hence $a^u :_1 w_1 :_2 \cdots :_k w_k \models q_k$. Thus, $q_k \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$ is sound.

Definition 13 (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 $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$ for $k < n$ 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 result.

Lemma 4. *Let A be a sound stack automaton A and let w be an order-k stack. If A accepts w from a state $q \in \mathcal{Q}_k$ then $w \models q$. In particular, if A accepts an order-n stack w from a state $q_p \in \mathcal{Q}_n$ then $\langle p, w \rangle$ belongs to $Pre^*_{\mathcal{C}}(A_0)$.*

Proof. We proceed by induction on the size of the stack (where the size of an annotated stack is defined to be the size of a tree representing the stack).

Let w be an order-k stack accepted from a state $q \in Q_k$. We assume that the property holds for any smaller stack.

If w is empty then q is a final state. Recall that by assumption final states are not initial. Hence, q is not initial and therefore q is a final state of A_0. It follows that the empty stack is accepted from q in A_0 and hence $w \models q$.

If w is a not empty stack of order-1. The stack w can be decompose as $a^u :_1 v$. As w is accepted by A from q, there exists a transition $q \xrightarrow[Q_{br}]{a} (Q)$ such that v is accepted from Q and u is accepted from Q_{br}. By induction hypothesis (both u and v are smaller than w), we have $u \models Q_{br}$ and $v \models Q_{br}$. As the transition $q \xrightarrow[Q_{br}]{a} (Q)$ is sound, we have that $w \models q$.

If w is a not empty stack of order-k. The stack w can be decompose as $u :_{k+1} v$. As w is accepted by A from q, there exists a transition $q \xrightarrow{q'} Q$ such that v is accepted from Q and u is accepted from q'. By induction hypothesis, we have $u \models q'$ and $v \models Q$. By definition of the satisfiability wrt the (initial) state q', we have $w \models q'$.

We first establish that the initial automaton A_0 is sound.

Lemma 5 (Soundness of A_0). *The automaton A_0 is sound.*

Proof. Let $q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k)$ for $k \leq n$. As we required in Section 3.1 that no initial states of A_0 has an incoming transition, the sets Q_i, $i \in [1, k]$ and the set Q_{br} do not contain any initial states.

By Lemma **??**, it is only necessary to prove the property for $k = n$ or for $k < n$ with q being non-initial.

First assume that $k < n$ and that q is not initial. For any order-1 stack $w_1 \models Q_1, \ldots,$ and any order-k stack $w_k \models Q_k$ and any stack $u \models Q_{br}$, we have to show that $w = a^u :_1 w_1 :_2 \cdots :_k w_k \models q$. As q is not initial, we need to show that A_0 accepts w from q. As for all $i \in [1, k]$, we have A_0 accepts w_i from the set of states Q_i and the stack u from the set Q_{br}, we easily derive an accepting run of A_0 from q on reading w.

Now assume that $k = n$. For any order-1 stack $w_1 \models Q_1, \ldots,$ and any order-k stack $w_k \models Q_n$ and any stack $u \models Q_{br}$, we have to show that $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q$. It suffices to show that A_0 accepts w from q (as if q is equal to q_p for some control state p, this implies that $\langle q_p, w\rangle$ belongs $Pre^*_{\mathcal{C}}(A_0)$). As for all $i \in [1, n]$, we have A_0 accepts w_i from the set of states Q_i and the stack u from the set Q_{br}, we easily derive an accepting run of A_0 from q on reading w.

We are now ready to prove that the stack automaton produced by our algorithm is sound.

Lemma 6 (Automaton Soundness). *The automaton A constructed by saturation with Γ and $\mathcal{C}$ from A_0 is sound.*

Proof. The proof is by induction on the number of iterations of Γ. The base case is the automaton A_0 and the result was established in Lemma **??**.

The inductive step is by case distinction on the rule (p, a, o, p') that led to the introduction of each new transition.

1. Assume that $o = pop_k$, that we had a transition $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ and that we added the transition

$$q_p \xrightarrow[\emptyset]{a} (\emptyset, \ldots, \emptyset, \{q_k\}, Q_{k+1}, \ldots, Q_n) \ .$$

   To establish soundness of this latter transition, we have to prove that for any $w_1 \models \emptyset, \ldots,$ any $w_{k-1} \models \emptyset$, any $w_k \models q_k$, any $w_{k+1} \models Q_{k+1}, \ldots,$ any $w_n \models Q_n$ and any $u \models \emptyset$, one has $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q_p$. For this, it suffices to show that $pop_k(w) \models q_{p'}$. Indeed, as q_p and $q_{p'}$ are initial states in $\mathcal{Q}_n$, $w \models q_p$ means that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ and $pop_k(w) \models q_{p'}$ means that $\langle p', pop_k(w)\rangle \in Pre^*_{\mathcal{C}}(A_0)$: as we have the rule (p, a, pop_k, p'), we will have the result as required.
   Since $w_k \models q_k$ and $w_{k+1} \models Q_{k+1}, \ldots, w_n \models Q_n$, we know from the definition of $\models$ and $q_{p'} \xrightarrow{q_k} (Q_{k+1} \ldots, Q_n)$ that $pop_k(w) = w_k :_{(k+1)} \cdots :_n w_n \models q_{p'}$ and we are done.

2. Assume that $o = push_k$, that we had the sound transitions

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

and that we added

$$q_p \xrightarrow[Q_{br} \cup Q'_{br}]{a} \big(Q_1 \cup Q'_1, \ldots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \ldots, Q_n\big) \ .$$

To establish soundness of 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 $u \models Q_{br} \cup Q'_{br}$, we have that $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q_p$. For this it suffices to show $push_k(w) \models q_{p'}$. Indeed, as q_p and $q_{p'}$ are initial states in $\mathcal{Q}_n$, $w \models q_p$ means that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ and $push_k(w) \models q_{p'}$ means that $\langle p', push_k(w)\rangle \in Pre^*_{\mathcal{C}}(A_0)$: as we have the rule $(p, a, push_k, p')$, we will have the result as required.
Let $v = top_k(w) = a^u :_1 w_1 :_2 \cdots :_{(k-1)} w_{k-1}$. From the soundness of $Q_k \xrightarrow[Q'_{br}]{a} (Q'_1, \ldots, Q'_k)$ and as $u \models Q'_{br}, w_1 \models Q'_1, \ldots, w_k \models Q'_k$, 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 $u \models Q_{br}$ and the soundness of $q_{p'} \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$ we get $push_k(w) = v :_k v :_k w_k :_{(k+1)} \cdots :_n w_n \models q_{p'}$ as required.

3. Assume that $o = collapse_k$, that $k < n$ and that we had a transition $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ and we added

$$q_p \xrightarrow[\{q_k\}]{a} (\emptyset, \ldots, \emptyset, Q_{k+1}, \ldots, Q_n) \ .$$

To establish soundness of this latter transition, we have to prove that for any $w_1 \models \emptyset, \ldots$, any $w_k \models \emptyset$, any $w_{k+1} \models Q_{k+1}, \ldots$, any $w_n \models Q_n$ and any $u \models q_k$, that we have $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q_p$. For this it suffices to show $collapse_k(w) \models q_{p'}$. Indeed, as q_p and $q_{p'}$ are initial states in $\mathcal{Q}_n$, $w \models q_p$ means that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ and $collapse_k(w) \models q_{p'}$ means that $\langle p', collapse_k(w)\rangle \in Pre^*_{\mathcal{C}}(A_0)$: as we have the rule $(p, a, collapse_k, p')$, we will have the result as required.
Since $u \models q_k$ and $w_{k+1} \models Q_{k+1}, \ldots, w_n \models Q_n$, we know from the definition of $\models$ and $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ that $collapse_k(w) = u :_{(k+1)} w_{k+1} :_{(k+2)} \cdots :_n w_n \models q_{p'}$ hence, we are done.

Assume now that $o = collapse_k$, that $k = n$ and that we added $q_p \xrightarrow[\{q_{p'}\}]{a} (\emptyset, \ldots, \emptyset)$. To prove that this latter transition is sound, we have to prove that for any $w_1 \models \emptyset, \ldots$, any $w_n \models \emptyset$ and any $u \models q_{p'}$, we have $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q_p$. We immediately know $collapse_k(w) = u \models q_{p'}$, and we conclude using the same argument as in the first subcase.

4. Assume that $o = push_b^k$, that we had the sound transitions

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

and that we added

$$q_p \xrightarrow[Q'_{br}]{a} (Q'_1, Q_2, \ldots, Q_k \cup Q_{br}, \ldots, Q_n) \ .$$

To prove that this later transition is sound, 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 $w_k \models Q_k \cup Q_{br}$, any $w_{k+1} \models Q_{k+1}, \ldots$, any $w_n \models Q_n$ and any $u \models Q'_{br}$, that we have $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q_p$.
For this it suffices to show $push_b^k(w) \models q_{p'}$. Indeed, as q_p and $q_{p'}$ are initial states in $\mathcal{Q}_n$, $w \models q_p$ means that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ and $push_b^k(w) \models q_{p'}$ means that $\langle p', push_b^k(w)\rangle \in Pre^*_{\mathcal{C}}(A_0)$: as we have the rule $\big(p, a, push_b^k, p'\big)$, we will have the result as required. From the soundness of $Q_1 \xrightarrow[Q'_{br}]{a} (Q'_1)$ and as $u \models Q'_{br}$ 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 $top_{k+1}(pop_k(w)) = w_k \models Q_{br}$, and the soundness of $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$, we get $push_b^k(w) = b^{w_k} :_1 a^u :_1 w_1 :_2 \cdots :_n w_n \models q_{p'}$ as required.
5. Assume that $o = rew_b$, that we had the sound transition $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$ and that we added

$$q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n) \ .$$

To prove that this later transition is sound, we have to prove that for any $w_1 \models Q_1, \ldots$, for any $w_n \models Q_n$ and any $u \models Q_{br}$, we have that $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q_p$. For this it suffices to show $rew_b(w) \models q_{p'}$. Indeed, as q_p and $q_{p'}$ are initial states in $\mathcal{Q}_n$, $w \models q_p$ means that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ and $rew_b(w) \models q_{p'}$ means that $\langle p', rew_b(w)\rangle \in Pre^*_{\mathcal{C}}(A_0)$: as we have the rule (p, a, rew_b, p'), we will have the result as required.
From $w_1 \models Q_1, \ldots, w_n \models Q_n$, and $u \models Q_{br}$, and the soundness of $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$, we get $rew_b(w) = b^u :_1 w_1 :_2 \cdots :_n w_n \models q_{p'}$ as required.

The final case is when a transition was added due to an alternating transition $p \to P$. Letting $Q = \{\ q_{p'} \ \mid p' \in P\ \}$ we added a rule $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$ from the sound transition $Q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$.

To prove that this later transition is sound, we have to prove that for any $w_1 \models Q_1, \ldots$, any $w_n \models Q_n$ and any $u \models Q_{br}$, we have that $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models q_p$. For this it suffices to show $w \models Q$. Indeed, as q_p and Q are initial states in $\mathcal{Q}_n$, $w \models q_p$ means that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ and $w \models Q$ means that $\langle p', w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ for all $p' \in P$: as we have the rule $p \to P$, we will have the result as required.

From $w_1 \models Q_1, \ldots, w_n \models Q_n$, and $u \models Q_{br}$, and the soundness of $Q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$, we immediately get $w = a^u :_1 w_1 :_2 \cdots :_n w_n \models Q$ as required.

D Completeness

We know show that the automaton constructed by Γ is complete for annotated stacks. The intuition behind the completeness proof is well illustrated by the examples in Section 3.2, hence we encourage the reader to consult these examples when reading the proof.

Lemma 7 (Completeness of Γ). *Given an annotated PDS $\mathcal{C}$ and an order-n stack automaton A_0, the automaton A constructed by saturation with Γ 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

$$Pre^0_{\mathcal{C}}(A_0) = \left\{ \ \langle p, w\rangle \ \middle| \ w \in \mathcal{L}_{q_p}(A_0) \ \right\}$$

$$Pre^{\alpha+1}_{\mathcal{C}}(A_0) = \left\{ \ \langle p, w\rangle \ \middle| \ \begin{array}{l} \exists \langle p, w\rangle \longrightarrow \langle p', w'\rangle \in Pre^{\alpha}_{\mathcal{C}}(A_0) \ \vee \\ \exists \langle p, w\rangle \longrightarrow C \subseteq Pre^{\alpha}_{\mathcal{C}}(A_0) \end{array} \right\}$$

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.

There are two cases. In the first, inductively assume $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ and an accepting run of w' from $q_{p'}$ of A via a rule $(p, top_1(w), o, p')$. Hence $w' = o(w)$.

1. When $o = pop_k$, let $q_{p'} \xrightarrow{q_k} (Q_1, \ldots Q_{k+1})$ be the order-n to order-k part of the initial transition accepting w'. We know, from the construction, we have the transition $q_p \xrightarrow[\emptyset]{a} (\emptyset, \ldots, \emptyset, \{q_k\}, Q_{k+1}, \ldots, Q_n)$. Let $w = u_{k-1} :_k u_k :_{(k+1)} \cdots :_n u_n$. We know $w' = u_k :_{(k+1)} \cdots :_n u_n$ and the run from q_k over w' is accepting. Hence, via the transition from q_p we get an accepting run of w which labels all nodes in w' with the same nodes as the accepting run of w' from q_k.
2. When $o = push_k$, let $w = u_{k-1} :_k \cdots :_n u_n$. We know

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

   Let $q_{p'} \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k, \ldots Q_n)$ and $Q_k \xrightarrow[Q'_{br}]{a} (Q'_1, \ldots, Q'_k)$ be the initial transitions used on the run of w' (where the transition from Q_k reads the second copy of u_{k-1}).
   From the construction we the have a transition

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

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_{br} and Q'_{br} respectively, we obtain an accepting run of w.

3. When $o = collapse_k$, let

$$w = a^u :_1 u_k :_{(k+1)} \cdots :_n u_n$$

where $w' = u :_{(k+1)} u_{k+1} :_{(k+1)} \cdots :_n u_n$. Let $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ in A be the initial transition on the accepting run of w' (or take $q_k = q_{p'}$ if $k = n$). We know from the construction that we have the transition $q_p \xrightarrow[\{q_k\}]{a} (\emptyset, \ldots, \emptyset, Q_{k+1}, \ldots, Q_n)$.
We know the run from q_k via $Q_{k+1}, \ldots, Q_n$ over w' is accepting. Hence, via the transition from q_p we get an accepting run of w which labels all nodes in w' with the same nodes as the accepting run of w' from q_k (we label u_k with empty sets).
4. When $o = push_b^k$, let $w = u_{k-1} :_k u_k :_{k+1} \cdots :_n u_n$. We know $w' = push_b^k(w)$ is

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

Let $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$ and $Q_1 \xrightarrow[Q'_{br}]{a} Q'_1$ be the first transitions used on the accepting run of w'. From the construction we introduce a transition $q_p \xrightarrow[Q'_{br}]{a} (Q'_1, Q_2, \ldots, Q_k \cup Q_{br}, \ldots, Q_n)$ from which we can construct an accepting run of w (which is w' without the first b on top of the top order-1 stack). A run from $Q_k \cup Q_{br}$ exists since u_k is also the annotation of b.
5. When $o = rew_b$ let $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$ be the first transition on the accepting run of $w' = b^u :_1 v$ for some v and u. From the construction we know we have a transition $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$, from which we get an accepting run of $w = a^u :_1 v$ as required.

The second case is when we have a branching transition $\langle p, w \rangle \longrightarrow C$ (where C is a set of configurations) via a rule $p \to P$. In this case, let $Q = \{ \ q_{p'} \ | \ p' \in P \ \}$ and $Q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$ be the accumulation of all initial transitions from states in Q. By construction, we have a rule $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$ from which an accepting run over w can easily be built.

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

E Complexity of collapsible PDS Model Checking

In this section we show that the complexity of the algorithm can be improved to $(n-1)$-EXPTIME when analysing non-alternating collapsible pushdown systems, matching the known $(n-1)$-completeness of the problem. The cause of the

additional exponential blow up is the alternation in the automaton constructed. Hence, we introduce a notion of non-alternation at order-n. Note that the automata are alternating both via transitions to Q with $|Q| > 1$, and via collapse links.

Definition 14 (Non-Alternation at Order-n). *An order-n stack automaton A is* non-alternating at order-n *whenever, for all stacks $w \in \mathcal{L}_q(A)$ for some state q of A, there is an accepting run of A over the graph of w such that no node is labelled by $Q \subseteq \mathcal{Q}_n$ and $|Q| > 1$.*

For example, take a run over an example order-2 collapsible stack

$$q_{p'} \xrightarrow{[_1} q_1 \xrightarrow{a} Q_1^1 \xrightarrow{b} Q_1^2 \xrightarrow{]_1} Q_2^1 \xrightarrow{[_1} Q_1^3 \xrightarrow{a} Q_1^4 \xrightarrow{b} Q_1^5 \xrightarrow{]_1} Q_2^2 \xrightarrow{[_1} \cdots \quad .$$

This is non-alternating at order-n whenever $\left|Q_2^1\right| \leq 1$ and $\left|Q_2^2\right| \leq 1$.

Note, for example, that a stack automaton that does not follow collapse links, and has no alternating transitions in Δ_n, is trivially non-alternating at order-n. Similarly, we may allow $q \xrightarrow[Q_{br}]{a} (\emptyset, \ldots, \emptyset)$ when $Q_{br} \subseteq \mathcal{Q}_n$ and $|Q_{br}| \leq 1$.

Observe that, since annotations are always independent of each other, it is not clear how to define a similar notion for annotated PDS, since combining runs due to a $push_n$ cannot exploit that the collapse links "pointed to the same stack".

We then define Γ' to be the saturation function Γ with the additional constraint that a transition $q \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$ is not added if $|Q_n| > 1$. Clearly saturation by Γ' remains sound, since it contains a subset of the transitions of the automaton produced by saturation with Γ. Hence, we only need to prove that the automaton remains complete. We have the following lemma. Intuitively, the automaton remains correct because a collapse link at order-n can only be used once, whereas, at lower orders, a $push_k$ operation may make different copies of a link (with different targets). Hence, lower order links need alternation to keep track of the different uses of the link throughout the run.

Lemma 8 (Completeness of Γ'). *Given a non-alternating collapsible PDS $\mathcal{C}$ and an order-n stack automaton A_0 that is non-alternating at order-n, the automaton A constructed by saturation with Γ' is such that $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ implies $w \in \mathcal{L}_{q_p}(A)$.*

Proof. The proof is by induction over α such that $\langle p, w\rangle \in Pre^{\alpha}_{\mathcal{C}}(A_0)$. We prove simultaneously during the induction that all $\langle p, w\rangle \in Pre^*_{\mathcal{C}}(A_0)$ have an accepting run from q_p of A over the graph of w such that no node is labelled by $Q \subseteq \mathcal{Q}_n$ and $|Q| > 1$.

Henceforth, for convenience, we will refer to non-alternation at order-n as simply *non-alternation*.

In the base case, we have $w \in \mathcal{L}_{q_p}(A_0)$ and the existence of a non-alternating run of A_0, and thus a run in A comes directly from the non-alternating run of A_0.

Hence, inductively assume $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ and a non-alternating accepting run of w' from $q_{p'}$ of A via a rule $(p, top_1(w), o, p')$. Hence $w' = o(w)$.

1. When $o = pop_k$, let $q_{p'} \xrightarrow{q_k} (Q_1, \ldots, Q_{k+1})$ be the order-n to order-k part of the initial transition accepting w'. We know, from the construction, we have the transition $q_p \xrightarrow[\emptyset]{a} (\emptyset, \ldots, \emptyset, \{q_k\}, Q_{k+1}, \ldots, Q_n)$. Let $w = u_{k-1} :_k u_k :_{(k+1)} \cdots :_n u_n$. We know $w' = u_k :_{(k+1)} \cdots :_n u_n$ and the run from q_k over w' is accepting and non-alternating. Hence, via the transition from q_p we get an accepting run of w which labels all nodes in w' with the same nodes as the accepting run of w' from q_k. Since the transitions reading u_{k-1} are all to the empty-set, this gives us a run over w that remains non-alternating.
2. When $o = push_k$, let $w = u_{k-1} :_k \cdots :_n u_n$. We know
$$w' = u_{k-1} :_k u_{k-1} :_k u_k :_{(k+1)} \cdots :_n u_n \ .$$
Let $q_{p'} \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_k, \ldots, Q_n)$ and $Q_k \xrightarrow[Q'_{br}]{a} (Q'_1, \ldots, Q'_k)$ be the initial transitions used on the run of w' (where the transition from Q_k reads the second copy of u_{k-1}).
From the construction we the have a transition
$$q_p \xrightarrow[Q_{br} \cup Q'_{br}]{a} \left(Q_1 \cup Q'_1, \ldots, Q_{k-1} \cup Q'_{k-1}, Q'_k, Q_{k+1}, \ldots, Q_n\right) \ .$$
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 with collapse links Q_{br} and Q'_{br} respectively, we obtain an accepting run of w.
Since we know that the destinations of the collapse links that go outside of u_{k-1} (in particular, the order-n links) always point to the same stacks in both copies of u_{k-1} in the non-alternating accepting run of w', we know that the run thus defined from q_p is both accepting and non-alternating. We remark that, while this is *always* the case for order-n links, it is not always the case for order-k links with $k < n$, and hence, alternation is needed at orders lower than n.
3. When $o = collapse_k$, let
$$w = u^1_{k-1} :_k \cdots :_k u^\ell_{k-1} :_k u_k :_{(k+1)} \cdots :_n u_n$$
where $w' = u_k :_{(k+1)} \cdots :_n u_n$ is the stack pointed to by the order-k collapse link of $top_1(w)$. Let $q_{p'} \xrightarrow{q_k} (Q_{k+1}, \ldots, Q_n)$ in A be the initial transition on the accepting run of w'. We know from the construction that we have the transition $q_p \xrightarrow[\{q_k\}]{a} (\emptyset, \ldots, \emptyset, Q_{k+1}, \ldots, Q_n)$.
We know the run from q_k via $Q_{k+1}, \ldots, Q_n$ over w' is accepting and non-alternating. Hence, via the transition from q_p we get an accepting run of w which labels all nodes in w' with the same nodes as the accepting run of w' from q_k (by following the collapse link). Since the transition reading u^1_{k-1} is to the empty set, this immediately allows us to label $u^2_{k-1}, \ldots, u^\ell_{k-1}$ with empty sets, giving us a run over w that remains non-alternating.

4. When $o = push_b^k$, let $w = u_{k-1} :_k u_k :_{k+1} \cdots :_n u_n$. We know $w' = push_b^k(w)$ is
$$b^{u_k} :_1 u_{k-1} :_k \cdots :_n u_n \ .$$
Let $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$ and $Q_1 \xrightarrow[Q'_{br}]{a} Q'_1$ be the first transitions used on the accepting, non-alternating run of w'. From the construction we have $q_p \xrightarrow[Q'_{br}]{a} (Q'_1, Q_2, \ldots, Q_k \cup Q_{br}, \ldots, Q_n)$ from which we can construct an accepting run of w (which is w' without the first b on top of the top order-1 stack).
To see that the run is non-alternating, we observe that $Q_k \cup Q_{br}$ results in the same labelling on the nodes of the graph of w as in corresponding run over w'. Apart from the labelling of the initial nodes (which label the beginning of each top_k stack), the run over w is identical to the run over w', and hence, non-alternating.
5. When $o = rew_b$ let $q_{p'} \xrightarrow[Q_{br}]{b} (Q_1, \ldots, Q_n)$ be the first transition on the non-alternating accepting run of $w' = b :_1 v$ for some v. From the construction we know we have a transition $q_p \xrightarrow[Q_{br}]{a} (Q_1, \ldots, Q_n)$, from which we get an accepting, non-alternating run of $w = a :_1 v$ as required.

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

This gives us a algorithm running in $(n-1)$-EXPTIME, since by avoiding alternation at order-n, we reduce the size of the automaton by a single exponential.

Proposition 6. *The saturation construction using Γ' for a non-alternating order-n collapsible PDS $\mathcal{C}$ and an order-n stack automaton A_0 that is non-alternating at order-n terminates in $(n-1)$-EXPTIME.*