Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems[⋆]

M. Hague C.-H. L. Ong

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK, OX1 3QD

Abstract. Higher-order pushdown systems (PDSs) generalise pushdown systems through the use of higher-order stacks, that is, a nested "stack of stacks" structure. We further generalise higher-order PDSs to higher-order Alternating PDSs (APDSs) and consider the backwards reachability problem over these systems. We prove that given an order-n APDS, the set of configurations from which a given regular set of configurations is reachable is itself regular and computable in n-EXPTIME. We show that the result has several useful applications in the verification of higher-order PDSs such as LTL model checking, alternation-free μ-calculus model checking, and the computation of winning regions of reachability games.

1 Introduction

Pushdown automata are an extension of finite state automata. In addition to a finite set of control states, a pushdown automaton has a stack which can be manipulated with the usual push and pop operations. *Higher-order pushdown automata* (PDA) generalise pushdown automata through the use of higher-order stacks. Whereas a stack in the sense of a pushdown automaton is a first-order stack — that is, a stack of characters — a second-order stack is a stack of first-order stacks. Similarly, a third-order stack is a stack of second-order stacks, and so on.

Higher-order PDA were originally introduced by Maslov [17] in the 1970s as generators of (a hierarchy of) finite word languages. *Higher-order pushdown systems* (PDSs) are higher-order PDA viewed as generators of infinite trees or graphs. These systems provide a natural infinite-state model for higher-order programs with recursive function calls and are therefore useful in software verification. Several notable advances in recent years have sparked off a resurgence of interest in higher-order PDA/PDSs in the Verification community. E.g. Knapik *et al.* [23] have shown that the ranked and ordered trees generated by deterministic order-n PDSs are exactly those that are generated by order-n recursion schemes satisfying the *safety* constraint; Carayol and Wöhrle [5] have shown that the ϵ-closure of the configuration graphs of higher-order PDSs exactly constitute

⋆ The full version [12] of this work is downloadable from the first author's web page.

Caucal's graph hierarchy [9]. Remarkably these infinite trees and graphs have decidable monadic second-order (MSO) theories [10, 5, 23].

These MSO decidability results, though powerful, only allow us to check that a property holds from a given configuration. We may wish to compute the set of configurations that satisfy a given property, especially since there may be an infinite number of such configurations. In this paper, we consider a closely-related problem:

> *Backwards Reachability*: Given a set of configurations C_{Init}, compute the set $Pre^*(C_{Init})$ of configurations that can, via any number of transitions, reach a configuration in C_{Init}.

This is an important verification problem in its own right, since safety properties (i.e. undesirable program states – such as deadlock – are never reached) feature largely in practice.

The backwards reachability problem was solved for order-one PDSs by Bouajjani *et al.* [2]. In particular, they gave a method for computing the (regular) set of configurations $Pre^*(C_{Init})$ that could reach a given regular set of configurations C_{Init}. Regular sets of configurations are represented symbolically in the form of a finite multi-automaton. That is, a finite automaton that accepts finite words (representing stacks) with an initial state for each control state of the PDS. A configuration is accepted if the stack (viewed as a word) is accepted from the appropriate initial state. The set $Pre^*(C_{Init})$ is computed by the repeated addition of a number of transitions – determined by the transition relation of the PDS – to the automaton accepting C_{Init}, until a fixed point is reached. A fixed point is guaranteed since no states are added and the alphabet is finite: eventually the automaton will become *saturated*.

The approach was extended by Bouajjani and Meyer [1] to the case of *higher-order context-free pushdown systems*, which are higher-order PDSs with a single control state. A key innovation in their work was the introduction of a new class of (finite-state) automata called *nested store automata*, which captures an intuitive notion of regular sets of n-stores. An order-one nested store automaton is simply a finite automaton over words. An order-n nested store automaton is a finite automaton whose transitions are labelled by order-$(n-1)$ nested store automata.

Our paper is concerned with the non-trivial problem[1] of extending the backwards reachability result of Bouajjani and Meyer to the general case of higher-order PDSs (i.e. taking into account a set of control sets). In fact, we consider (and solve) the backwards reachability problem for the slightly more general case of higher-order *alternating* pushdown systems (APDSs). Though slightly unwieldy, an advantage of the alternating framework is that it conveniently lends itself to a number of higher-order PDS verification problems. We show that the winning region of a reachability game played over a higher-order PDS can be

[1] "This does not seem to be technically trivial, and naïve extensions of our construction lead to procedures which are not guaranteed to terminate." [1, p. 145]

computed by a reduction to the backwards reachability problem of an appropriate APDS. We also generalise results due to Bouajjani *et al.* [2] and Cachat [21] to give a method for computing the set of configurations of a higher-order PDS that satisfy a given formula of the alternation-free μ-calculus or a linear-time temporal logic.

Related Work. Prompted by the fact that the set of configurations reachable from a given configuration of a higher-order PDS is not regular in the sense of Bouajjani and Meyer (the stack contents cannot be represented by a finite automaton over words), Carayol [4] has proposed an alternative definition of regularity for higher-order stacks, which we shall call *C-regularity.* Our notion of regularity coincides with that of Bouajjani and Meyer, which, when confusion may arise, we will refer to as *BM-regularity.*

A set of order-n stacks is *C-regular* if it is constructible from the empty n-stack by a regular sequence of order-n stack operations. Carayol shows that C-regularity coincides with MSO definability over the canonical structure Δ_2^n associated with order-n stacks. This implies, for instance, that the winning region of a parity game over an order-n pushdown graph is also C-regular, as it can be defined as an MSO formula [21].

In this paper we solve the backwards reachability problem for higher-order PDSs and apply the solution to reachability games and model-checking. In this sense we give a weaker kind of result that uses a more "practical" notion of regularity. However our solution is direct and has a lower (though still large) complexity bound. Because C-regularity does not imply BM-regularity[2], our result is not subsumed by the work of Carayol.

The definition of higher-order PDSs may be extended to higher-order pushdown games. In the order-one case, the problem of determining whether a configuration is winning for Eloise with a parity winning condition was solved by Walukiewicz in 1996 [13]. The order-one backwards reachability algorithm of Bouajjani *et al.* was adapted by Cachat to compute the winning regions of order-one reachability and Büchi games [21]. Results for pushdown games have been extended to a number of winning conditions [22, 3, 11, 19, 8] including parity conditions [21, 18]. In the higher-order case with a parity winning condition, a method for deciding whether a configuration is winning has been provided by Cachat [21].

Higher-order recursion schemes (HORS) represent a further area of related work. MSO decidability for trees generated by arbitrary (i.e. not necessarily safe) HORS has been shown by Ong [20]. A variant kind of higher-order PDSs called *collapsable pushdown automata* (extending *panic automata* [24, 15] to all finite orders) has been shown to be equi-expressive with HORS (for generating ranked and ordered trees) by Murawski and Ong [7]. These new automata are conjectured to enrich the class of higher-order systems and provide many new avenues of research.

[2] For example $(push_a)^*; push_2$ defines all stacks of the form $[[a^n][a^n]]$.

2 Preliminaries

In the sequel we will introduce several kinds of alternating automata. For convenience, we will use a non-standard definition of alternating automata that is equivalent to the standard definitions of Brzozowski and Leiss [14] and Chandra, Kozen and Stockmeyer [6]. Similar definitions have been used for the analysis of pushdown systems by Bouajjani *et al.* [2] and Cachat [21]. The alternating transition relation $\Delta \subseteq \mathcal{Q} \times \Gamma \times 2^{\mathcal{Q}}$ — where Γ is a kind of alphabet and $\mathcal{Q}$ is a state-set — is given in disjunctive normal form. That is, the image $\Delta(q, \gamma)$ of $q \in \mathcal{Q}$ and $\gamma \in \Gamma$ is a set $\{Q_1, \ldots, Q_m\}$ with $Q_i \in 2^{\mathcal{Q}}$ for $i \in \{1, \ldots, m\}$. When the automaton is viewed as a game, Eloise — the existential player — chooses a set $Q \in \Delta(q, \gamma)$; Abelard — the universal player — then chooses a state $q \in Q$.

2.1 (Alternating) Higher-Order Pushdown Systems

We begin by defining higher-order stores and their operations. We will then define higher-order PDSs and APDSs in full.

The set C_1^{Σ} of 1-stores over an alphabet Σ is the set of words of the form $[a_1, \ldots, a_m]$ with $m \geq 0$ and $a_i \in \Sigma$ for all $i \in \{1, \ldots, m\}$, $[\notin \Sigma$ and $] \notin \Sigma$. For $n > 1$, $C_n^{\Sigma} = [w_1, \ldots, w_m]$ with $m \geq 1$ and $w_i \in C_{n-1}^{\Sigma}$ for all $i \in \{1, \ldots, m\}$. There are three types of operations applicable to n-stores: *push*, *pop* and *top*. These are defined inductively. Over a 1-store, we have (for all $w \in \Sigma^*$),

$$\begin{aligned} push_w[a_1 \ldots a_m] &= [w a_2 \ldots a_m] \\ top_1[a_1 \ldots a_m] &= a_1 \end{aligned}$$

We may define the abbreviation $pop_1 = push_\varepsilon$. When $n > 1$, we have

$$\begin{aligned} push_w[\gamma_1 \ldots \gamma_m] &= [push_w(\gamma_1)\gamma_2 \ldots \gamma_m] && \\ push_l[\gamma_1 \ldots \gamma_m] &= [push_l(\gamma_1)\gamma_2 \ldots \gamma_m] && \text{if } 2 \leq l < n \\ push_n[\gamma_1 \ldots \gamma_m] &= [\gamma_1\gamma_1\gamma_2 \ldots \gamma_m] && \\ pop_l[\gamma_1 \ldots \gamma_m] &= [pop_l(\gamma_1)\gamma_2 \ldots \gamma_m] && \text{if } 1 \leq l < n \\ pop_n[\gamma_1 \ldots \gamma_m] &= [\gamma_2 \ldots \gamma_m] && \text{if } m > 1 \\ top_l[\gamma_1 \ldots \gamma_m] &= top_l(\gamma_1) && \text{if } 1 \leq l < n \\ top_n[\gamma_1 \ldots \gamma_m] &= \gamma_1 && \end{aligned}$$

Note that we assume wlog $\Sigma \cap \mathcal{N} = \emptyset$, where $\mathcal{N}$ is the set of natural numbers. Further, observe that when $m = 1$, pop_n is undefined. We define $\mathcal{O}_n = \{\ push_w \mid w \in \Sigma^*\ \} \cup \{\ push_l, pop_l \mid 1 < l \leq n\ \}$.

Definition 1. An *order-n pushdown system* (PDS) is a tuple $(\mathcal{P}, \mathcal{D}, \Sigma)$ where $\mathcal{P}$ is a finite set of control states p^j, $\mathcal{D} \subseteq \mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{Q}$ is a finite set of commands d, and Σ is a finite alphabet.

A configuration of an order-n PDS is a pair $\langle p, \gamma \rangle$ where $p \in \mathcal{P}$ and γ is an n-store. We have a transition $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle$ iff we have $(p, a, o, p') \in \mathcal{D}$, $top_1(\gamma) = a$ and $\gamma' = o(\gamma)$.

Definition 2. An *order-n alternating pushdown system* (APDS) is a tuple $(\mathcal{P}, \mathcal{D}, \Sigma)$ where $\mathcal{P}$ is a finite set of control states p^j, $\mathcal{D} \subseteq \mathcal{P} \times \Sigma \times 2^{\mathcal{O}_n \times \mathcal{Q}}$ is a finite set of commands d, and Σ is a finite alphabet.

A configuration of an order-n APDS is a pair $\langle p, \gamma \rangle$ where $p \in \mathcal{P}$ and γ is an n-store. We have a transition $\langle p, \gamma \rangle \hookrightarrow C$ iff we have $(p, a, OP) \in \mathcal{D}$, $top_1(\gamma) = a$, and

$$\begin{aligned} C = \{\ \langle p', \gamma' \rangle \mid (o, p') \in OP \ \wedge \ \gamma' = o(\gamma)\ \} \\ \cup \{\ \langle p, \triangledown \rangle \mid \text{if } (o, p') \in OP \text{ and } o(\gamma) \text{ is not defined }\} \end{aligned}$$

The transition relation generalises to sets of configurations via the following rule:

$$\frac{\langle p, \gamma \rangle \ \hookrightarrow \ C}{C' \cup \langle p, \gamma \rangle \ \hookrightarrow \ C' \cup C} \quad \langle p, \gamma \rangle \notin C'$$

In both cases, we define $\overset{*}{\hookrightarrow}$ to be the transitive closure of $\hookrightarrow$. For a set of configurations C_{Init} we define $Pre^*(C_{Init})$ as the set of configurations $\langle p, \gamma \rangle$ such that $\langle p, \gamma \rangle \overset{*}{\hookrightarrow} c$ and $c \in C_{Init}$ or $\langle p, \gamma \rangle \overset{*}{\hookrightarrow} C$ and $C \subseteq C_{Init}$ respectively.

Observe that since no transitions are possible from an "undefined" configuration $\langle p, \triangledown \rangle$ we can reduce the reachability problem for higher-order PDSs to the reachability problem over higher-order APDSs in a straightforward manner.

In the sequel, to ease the presentation, we assume $n > 1$. The case $n = 1$ was investigated by Bouajjani *et al.* [2].

2.2 *n*-Store Multi-Automata

To represent sets of configurations we will use *n-store multi-automata*. These are alternating automata whose transitions are labelled by $(n-1)$*-store automata*, which are also alternating. A set of configurations is said to be *regular* if it is accepted by an n-store multi-automaton.

Definition 3.

1. A *1-store automaton* is a tuple $(\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{Q}_f)$ where $\mathcal{Q}$ is a finite set of states, Σ is a finite alphabet, q_0 is the initial state and $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states. $\Delta \subseteq \mathcal{Q} \times \Sigma \times 2^{\mathcal{Q}}$ is a finite transition relation.
2. Let $\mathfrak{B}^{\Sigma}_{n-1}$ be the (infinite) set of all $(n-1)$-store automata over the alphabet Σ. An *n-store automaton* over the alphabet Σ is a tuple $(\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{Q}_f)$ where $\mathcal{Q}$ is a finite set of states, $q_0 \notin \mathcal{Q}_f$ is the initial state, $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states, and $\Delta \subseteq \mathcal{Q} \times \mathfrak{B}^{\Sigma}_{n-1} \times 2^{\mathcal{Q}}$ is a *finite* transition relation.
3. An *n-store multi-automaton* over the alphabet Σ is a tuple

$$(\mathcal{Q}, \Sigma, \Delta, \{q^1, \ldots, q^z\}, \mathcal{Q}_f)$$

where $\mathcal{Q}$ is a finite set of states, Σ is a finite alphabet, $q_i \notin \mathcal{Q}_f$ for $i \in \{1, \ldots, z\}$ are separate initial states and $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states, and

$$\Delta \subseteq (\mathcal{Q} \times \mathfrak{B}^{\Sigma}_{n-1} \times 2^{\mathcal{Q}}) \cup (\{q^1, \ldots, q^z\} \times \{\triangledown\} \times \{q^{\varepsilon}_f\})$$

is a *finite* transition relation where $q^{\varepsilon}_f \in \mathcal{Q}_f$ has no outgoing transitions.

To indicate a transition $(q, B, \{q_1, \ldots, q_m\}) \in \Delta$, we write,

$$q \xrightarrow{B} \{q_1, \ldots, q_m\}$$

Runs of the automata take the form,

$$q \xrightarrow{\widetilde{B}_0} \{q_1^1, \ldots, q_{m_1}^1\} \xrightarrow{\widetilde{B}_1} \ldots \xrightarrow{\widetilde{B}_m} \{q_1^m, \ldots, q_{m_l}^m\}$$

where transitions between configurations $\{q_1^x, \ldots, q_{m_x}^x\} \xrightarrow{\widetilde{B}_x} \{q_1^{x+1}, \ldots, q_{m_{x+1}}^{x+1}\}$ are such that we have $q_y^x \xrightarrow{B_y} Q_y$ for all $y \in \{1, \ldots, m_x\}$ and $\bigcup_{y \in \{1,\ldots,m_x\}} Q_y = \{q_1^{x+1}, \ldots, q_{m_{x+1}}^{x+1}\}$ and $\bigcup_{y \in \{1,\ldots,m_x\}} \{B_y\} = \widetilde{B}_x$. Observe that $\widetilde{B}_0$ is necessarily a singleton set.

We will, by abuse of notation, abbreviate a run over the word w to

$$q \xrightarrow{w} \{q_1, \ldots, q_m\}.$$

Further, when a run occurs in an automaton forming part of a sequence indexed by i, we may write $\longrightarrow_i$ to indicate which automaton the run belongs to.

A 1-store $[a_1 \ldots a_m]$ is accepted by a 1-store automaton A (that is $[a_1 \ldots a_m] \in \mathcal{L}(A)$) iff we have a run $q_0 \xrightarrow{a_1 \ldots a_m} Q$ in A with $Q \subseteq \mathcal{Q}_f$. For a given n-store automaton $A = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{Q}_f)$ we define

$$\mathcal{L}(A) = \{\ [\gamma_1 \ldots \gamma_m] \mid q_0 \xrightarrow{\widetilde{B}_0} \ldots \xrightarrow{\widetilde{B}_m} Q \ \wedge\ Q \subseteq \mathcal{Q}_f \ \wedge\ \forall 0 \leq i \leq m . \gamma_i \in \mathcal{L}(\widetilde{B}_i)\ \}$$

where $\gamma \in \mathcal{L}(\widetilde{B})$ iff $\gamma \in \mathcal{L}(B)$ for all $B \in \widetilde{B}$.

For an n-store multi-automaton $A = (Q, \Sigma, \Delta, \{q^1, \ldots, q^z\}, \mathcal{Q}_f)$ we define

$$\begin{aligned} \mathcal{L}(A^{q^j}) = \{\ [\gamma_1 \ldots \gamma_m] \mid q^j \xrightarrow{\widetilde{B}_0} \ldots \xrightarrow{\widetilde{B}_m} Q \\ \wedge\ Q \subseteq \mathcal{Q}_f \ \wedge\ \forall 0 \leq i \leq m . \gamma_i \in \mathcal{L}(\widetilde{B}_i)\ \} \\ \cup \{\ \triangledown \mid q^j \xrightarrow{\triangledown} q_f^\varepsilon\ \} \\ \mathcal{L}(A) = \{\ \langle p^j, \gamma \rangle \mid j \in \{1, \ldots, z\} \wedge \gamma \in \mathcal{L}(A^{q^j})\ \} \end{aligned}$$

Finally, we define the automata B_l^a and X_l^a for all $1 \leq l \leq n$ and $a \in \Sigma$ and the notation q^θ. B_l^a is the l-store automaton that accepts any l-store γ such that $top_1(\gamma) = a$. X_l^a is the $(n-1)$-store automaton accepting all $(n-1)$-stores such that $top_1(\gamma) = a$ and $top_{l+1}(\gamma) = [[w']]$ for some w'. That is, $pop_l(\gamma)$ is undefined. If θ represents an automaton, the state q^θ refers to the initial state of the automaton represented by θ.

3 Backwards Reachability

Theorem 1. *Given an n-store multi-automaton A_0 accepting the regular set of configurations C_{Init} of an order-n APDS, we can construct in n-EXPTIME an n-store multi-automaton A_* accepting the set $Pre^*(C_{Init})$. Thus, $Pre^*(C_{Init})$ is regular.*

Fix an order-n APDS. We begin by showing how to generate an infinite sequence of automata $A_0, A_1, \ldots$, where A_0 is given such that $\mathcal{L}(A_0) = C_{Init}$. This sequence is increasing in the sense that $\mathcal{L}(A_i) \subseteq \mathcal{L}(A_{i+1})$ for all i, and sound and complete with respect to $Pre^*(C_{Init})$, that is $\bigcup_{i \geq 0} \mathcal{L}(A_i) = Pre^*(C_{Init})$. We assume wlog that all initial states in A_0 have no incoming transitions and there exist in A_0 a state q_f^* from which all valid n-stores are accepted and a state $q_f^\varepsilon \in \mathcal{Q}_f$ that has no outgoing transitions.

An intuitive explanation of the algorithm. To simplify the description, we assume that the transition relations of the automata are not alternating. Thus, abstracting over the automata labelling the transitions of A_i, we have a directed graph with labelled edges. The construction of A_{i+1} from A_i is determined by the edges of A_i and the commands of the order-n (A)PDA. This construction step may involve the addition of a number of edges to A_i. Further, because commands may have an effect at all orders of the automaton, the automata labelling the edges of A_i need to be updated to define A_{i+1}.

Because an update to a single automaton for a single command will refer to any number of the $(n-1)$-store automata (which are edge-labels) in A_i, we consider all $(n-1)$-store automata simultaneously. That is, we view them as one large automaton with a set of initial states built from the initial states of the automaton's constituent parts. This can be thought of as a multi-automaton. During each update we add a fresh initial state for each automaton represented, and add transitions from these new states that move only to the previous state-set.

The sequence $A_0, A_1, \ldots$ does permit a finite representation A_*. We shall give an informal explanation in the following. Viewing A_i as a graph, the (order-n) vertex-set of A_i is fixed as i ranges over the natural numbers, and the algorithm adds at most one labelled edge (transition) from q_1 to q_2 for each pair of states q_1 and q_2. The order-$(n-1)$ automaton labelling this edge will be updated during the construction of A_{i+1} from A_i, but another edge from q_1 to q_2 will never be added. Hence, ignoring the labels, the edge relation at order-n of the A_i's will reach a fixed point. Because the updates to the order-$(n-1)$ automata are determined by the edge relation, they become repetitive, in such a way that – again ignoring edge labels – the new states added to the multi-automaton representing the $(n-1)$-store automata, together with their outgoing edges, will form an infinite chain with an unvarying pattern of edges. This chain can be collapsed into a single *finite* set of new states with edges between themselves as well as the previous state-set. This is illustrated in figure 1.

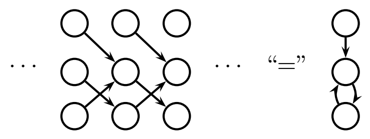


Fig. 1. Collapsing a repetitive chain of new states.

Because we consider one large order-$(n-1)$ automaton with several initial states, we essentially have an $(n-1)$-store multi-automaton. Moreover, the use of "self-referential edges" has allowed us to fix the state-set. Ignoring the edge labels, the edge relation of this automaton will eventually become saturated: we will reach a fixed point as in the order-n case. Inductively in this way the fixed points cascade to order-1 part of the automaton. Once the vertex set at order 1 has been fixed, a fixed point of the entire process is guaranteed because the alphabet is finite.

3.1 Preliminaries

To aid in the construction of an automaton representing $Pre^*(C_{Init})$, we introduce a new kind of transition to the n-store automata. These new transitions are labelled with place-holders that will eventually be converted into $(n-1)$-store automata.

New transitions are introduced during the processing of the order-n APDS commands. Between any state q_1 and set of states Q_2 we add at most one transition. We associate this transition with an identifier $\widetilde{G}_{(q_1,Q_2)}$. To describe our algorithm we will define several sequences of automata, indexed by i. The identifier $\widetilde{G}^i_{(q_1,Q_2)}$ is associated with a set that acts as a recipe for updating the $(n-1)$-store automaton described by $\widetilde{G}^{i-1}_{(q_1,Q_2)}$ or creating a new automaton if $\widetilde{G}^{i-1}_{(q_1,Q_2)}$ does not exist. Ultimately, the constructed $(n-1)$-store automaton will label the new transition.

The sets are in a kind of disjunctive normal form. A set $\{S_1, \dots, S_m\}$ represents an automaton that accepts the "disjunction" of the automata described by $S_1, \dots, S_m$. Each set $S \in \{S_1, \dots, S_m\}$ corresponds to a possible effect of a command d at the current order of the automaton. The automaton described by S is the "conjunction" of the automata described by its elements. An element that is an automaton B refers directly to the automaton B. Similarly, an identifier $\widetilde{G}$ refers to its corresponding automaton. Finally, an element of the form (a, o, θ) refers to an automaton capturing the effect of applying the inverse of the command o to the stacks accepted by the automaton represented by θ; moreover, the top_1 character of the stacks accepted by the new automaton will be a. It is a consequence of construction that for any S added during the algorithm, if $(a, o, \theta) \in S$ and $(a', o', \theta') \in S$ then $a = a'$.

Formally, to each $\widetilde{G}^i_{(q_1,Q_2)}$ at order-l we attach a subset of

$$2^{\mathcal{B}_{l-1} \cup \widetilde{\mathcal{G}}^{i-1}_{l-1} \cup (\Sigma \times \mathcal{O}_l \times (\mathcal{B}_{l-1} \cup \widetilde{\mathcal{G}}^{i-1}_{l-1}))}$$

where $\mathcal{B}_{l-1}$ is the union of the set of all $(l-1)$-store automata occurring in A_0 and all automata of the form B^a_{l-1} or X^a_{l-1}. Further, we denote the set of all order-$(l-1)$ identifiers $\widetilde{G}^i_{(\dots)}$ in A_i as $\widetilde{\mathcal{G}}^i_{l-1}$. The sets $\mathcal{B}_{l-1}$ and $\mathcal{D}$ are finite by definition. If the state-set at order-l is fixed, there is a finite bound on the size of the set $\widetilde{\mathcal{G}}^i_{l-1}$ for any i.

Given $\widetilde{\mathcal{G}}^i_l$ for some l, we build the automata for all $\widetilde{G}^i_{(\ldots)} \in \widetilde{\mathcal{G}}^i_l$ simultaneously. That is, we create a single automaton $\mathcal{G}^i_l$ associated with the set $\widetilde{\mathcal{G}}^i_l$. This automaton has a state $g^i_{(q_1,Q_2)}$ for each $\widetilde{G}^i_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^i_l$. The automaton $G^i_{(q_1,Q_2)}$ labelling the transition $q_1 \longrightarrow_i Q_2$ is the automaton $\mathcal{G}^i_l$ with $g^i_{(q_1,Q_2)}$ as its initial state.

The automaton $\mathcal{G}^i_l$ is built inductively. We set $\mathcal{G}^0_l$ to be the disjoint union of all automata in $\mathcal{B}_l$. We define $\mathcal{G}^{i+1}_l = T_{\widetilde{\mathcal{G}}^{i+1}_l}(\mathcal{G}^i_l)$ where $T_{\widetilde{\mathcal{G}}^j_l}(\mathcal{G}^i_l)$ is given in Definition 4. The reason for the distinction between $(i+1)$ and j will become clear in Section 3.3.

Definition 4. For $j > 0$, an automaton $\mathcal{G}^i_l$ and a set of identifiers $\widetilde{\mathcal{G}}^j_l$, we define the automaton $T_{\widetilde{\mathcal{G}}^j_l}(\mathcal{G}^i_l) = \mathcal{G}^{i+1}_l$. Initially, we set $\mathcal{G}^{i+1}_l = \mathcal{G}^i_l$. For each $\widetilde{G}^j_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^j_l$ we add a state $g^j_{(q_1,Q_2)}$ to $\mathcal{G}^{i+1}_l$ if it does not exist. There are now two cases depending on l:

Case $l = 1$: Each state $g^j_{(q_1,Q_2)}$ inherits transitions from $g^{j-1}_{(q_1,Q_2)}$ if it exists. That is, for each transition $g^{j-1}_{(q_1,Q_2)} \xrightarrow{b} Q$ in $\mathcal{G}^i_1$ we add the transition $g^j_{(q_1,Q_2)} \xrightarrow{b} Q$ to $\mathcal{G}^{i+1}_1$.

Then for each set $\widetilde{G}^j_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^j_1$, each set $\{\alpha_1, \ldots, \alpha_r\} \in \widetilde{G}^j_{(q_1,Q_2)}$, every $b \in \Sigma$ and $Q = Q_1 \cup \ldots \cup Q_r$, we add the transition,

$$g^j_{(q_1,Q_2)} \xrightarrow{b} Q$$

to $\mathcal{G}^{i+1}_l$ when for each $t \in \{1, \ldots, r\}$ we have

- If α_t is an automaton or identifier, then there exists the run $q^{\alpha_t} \xrightarrow{b} Q_t$ in $\mathcal{G}^i_1$.
- If $\alpha_t = (a, push_w, \theta_1)$, then $b = a$ and we have a path $q^{\theta_1} \xrightarrow{w} Q_t$ in $\mathcal{G}^i_1$.

Case $l > 1$: Initially we set $\widetilde{G}^{i+1}_{(\ldots)} = \emptyset$ for all $\widetilde{G}^i_{(\ldots)} \in \widetilde{\mathcal{G}}^i_{l-1}$. Each state $g^j_{(q_1,Q_2)}$ inherits transitions from $g^{j-1}_{(q_1,Q_2)}$ if it exists. That is, for each transition $g^{j-1}_{(q_1,Q_2)} \xrightarrow{B} Q$ in $\mathcal{G}^i_l$ we add,

$$g^j_{(q_1,Q_2)} \xrightarrow{\widetilde{G}^{i+1}_{(g^j_{(q_1,Q_2)},Q)}} Q$$

if it does not exist (setting $\widetilde{G}^{i+1}_{(g^j_{(q_1,Q_2)},Q)} = \emptyset$), and add the set $\{B\}$ to $\widetilde{G}^{i+1}_{(g^j_{(q_1,Q_2)},Q)}$.

Then for each set $\widetilde{G}^j_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^j_l$, each set $\{\alpha_1, \ldots, \alpha_r\} \in \widetilde{G}^j_{(q_1,Q_2)}$ and every pair of sets $S = S_1 \cup \ldots \cup S_r$ and $Q = Q_1 \cup \ldots \cup Q_r$, we add the transition,

$$g^j_{(q_1,Q_2)} \xrightarrow{\widetilde{G}^{i+1}_{(g^j_{(q_1,Q_2)},Q)}} Q$$

if it does not exist (setting $\widetilde{G}^{i+1}_{(g^j_{(q_1,Q_2)},Q)} = \emptyset$), and add S to $\widetilde{G}^{i+1}_{(g^j_{(q_1,Q_2)},Q)}$, when for each $t \in \{1, \ldots, r\}$ we have

– If α_t is an automaton or identifier, then $S_t = \{B\}$ and there exists $q^{\alpha_t} \xrightarrow{B} Q_t$ in $\mathcal{G}^i_l$.
– If $\alpha_t = (a, push_l, \theta_1)$, we have $S_t = \{B^a_{l-1}\} \cup \widetilde{B}_1 \cup \widetilde{B}_2$ and there exists a path $q^{\theta_1} \xrightarrow{\widetilde{B}_1} Q^1 \xrightarrow{\widetilde{B}_2} Q_t$ in $\mathcal{G}^i_l$.
– If $\alpha_t = (a, pop_l, \theta_1)$, we have $S_t = \{B^a_{l-1}\}$ and $Q_t = \{q^{\theta_1}\}$.
– If $\theta = (a, o, \theta_1)$ when $\ell(o) < l$, we have $S_t = \{(a, o, B)\}$ and a run $q^{\theta_1} \xrightarrow{B} Q_t$ in $\mathcal{G}^i_l$.

We have constructed an automaton with transitions labelled by sets in $\widetilde{\mathcal{G}}^{i+1}_{l-1}$. We construct $\mathcal{G}^{i+1}_{l-1}$ through a recursive call to $T_{\widetilde{\mathcal{G}}^{i+1}_{l-1}}(\mathcal{G}^i_{l-1})$.

3.2 Constructing the Sequence $A_0, A_1, \ldots$

For a given order-n APDS with commands $\mathcal{D}$ we define the operation $A_{i+1} = T_{\mathcal{D}}(A_i)$ as follows.

Definition 5. For an automaton A_i and a set of order-n APDS commands $\mathcal{D}$, we define the automaton $T_{\mathcal{D}}(A_i) = A_{i+1}$. We set $\widetilde{G}^{i+1}_{(q_1,Q_2)} = \emptyset$ for all $\widetilde{G}^i_{(q_1,Q_2)}$ in $\widetilde{\mathcal{G}}^i_{n-1}$. Then, for each $d = (p^j, a, \{(o_1, p^{k_1}), \ldots, (o_m, p^{k_m})\}) \in \mathcal{D}$, we perform the following update: for every pair of sets $S = S_1 \cup \ldots \cup S_m$ and $Q = Q_1 \cup \ldots \cup Q_m$ add the transition,

$$q^j \xrightarrow{\qquad \widetilde{G}^{i+1}_{(q^j,Q)} \qquad} Q$$

if it does not exist (setting $\widetilde{G}^{i+1}_{(q^j,Q)} = \emptyset$), and add S to $\widetilde{G}^{i+1}_{(q^j,Q)}$, when for each $t \in \{1, \ldots, m\}$ we have,

– If $o_t = push_n$, then $S_t = \{B^a_{n-1}\} \cup \widetilde{\theta}_1 \cup \widetilde{\theta}_2$ and there exists a run,

$$q^{k_t} \xrightarrow{\widetilde{\theta}_1}_i Q' \xrightarrow{\widetilde{\theta}_2}_i Q_t$$

in A_i.
– If $o_t = pop_n$, then $S_t = \{B^a_{n-1}\}$ and $Q_t = \{q^{k_t}\}$. Or, if $q^j \xrightarrow{\triangledown}_i \{q^\varepsilon_f\}$ exists in A_i, we may have $S_t = \{B^a_{n-1}\}$ and $Q_t = \{q^\varepsilon_f\}$.
– If $o_t = push_w$ or $o_t = push_l$ for $l < n$, then $S_t = \{(a, o, \theta)\}$ and there exists a transition $q^{k_t} \xrightarrow{\theta}_i Q_t$ in A_i.
– If $o_t = pop_l$ for $l < n$, then $S_t = \{(a, o, \theta)\}$ and there exists a transition $q^{k_t} \xrightarrow{\theta}_i Q_t$ in A_i. Or, if $q^j \xrightarrow{\triangledown}_i \{q^\varepsilon_f\}$ exists in A_i, we may have $S_t = \{X^a_l\}$ and $Q_t = \{q^*_f\}$.

By repeated applications of $T_{\mathcal{D}}$ we construct the sequence $A_0, A_1, \ldots$ which can be converted into n-store multi-automata using the procedure described in the previous section. This sequence is sound and complete with respect to $Pre^*(C_{Init})$.

Property 1. For any configuration $\langle p^j, \gamma\rangle$ it is the case that $\gamma \in \mathcal{L}(A_i^{q^j})$ for some i iff $\langle p^j, \gamma\rangle \in Pre^*(C_{Init})$.

3.3 Constructing the Automaton A_*

We need to construct a finite representation of the sequence $A_0, A_1, \ldots$ in a finite amount of time. To do this we will construct an automaton A_* such that $\mathcal{L}(A_*) = \bigcup_{i\in\omega} \mathcal{L}(A_i)$. We begin by introducing some notation and a notion of subset modulo i for the sets $\widetilde{G}^i_{(q,Q')}$.

Definition 6. Given $\theta \in \mathcal{B}_l \cup \widetilde{\mathcal{G}}^i_l$ for some i and l, let

$$\theta[j/i] = \begin{cases} \theta & \text{if } \theta \in \mathcal{B}_l \\ G^j_{(q_1,Q_2)} & \text{if } \theta = G^i_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^i_l \end{cases}$$

For a set S we define $S[j/i]$ such that, $\theta \in S$ iff we have $\theta[j/i] \in S[j/i]$, and $(a, o, \theta) \in S$ iff we have $(a, o, \theta[j/i]) \in S[j/i]$.

We extend the notation $[j/i]$ point-wise to nested sets of sets structures.

1. $\widetilde{G}^i_{(q,Q')} \lesssim \widetilde{G}^j_{(q,Q')}$ iff for each $S \in \widetilde{G}^i_{(q,Q')}$ we have $S[j-1/i-1] \in \widetilde{G}^j_{(q,Q')}$.
2. $\widetilde{\mathcal{G}}^i_l \lesssim \widetilde{\mathcal{G}}^j_l$ iff for all $\widetilde{G}^i_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^i_l$ we have $\widetilde{G}^j_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^j_l$ and $\widetilde{G}^i_{(q,Q')} \lesssim \widetilde{G}^j_{(q,Q')}$.

An important result in reaching a fixed point is that the sets $\widetilde{G}^i_{(q,Q')}$ are increasing. That is, for all i and $\widetilde{G}^i_{(q,Q')}$ we have $\widetilde{\mathcal{G}}^i_{(q,Q')} \lesssim \widetilde{\mathcal{G}}^{i+1}_{(q,Q')}$. This follows because no transitions are removed.

Writing $A \simeq B$ to mean $A \lesssim B$ and $B \lesssim A$, we now show that a fixed point is reached at order-n.

Property 2. There exists $i_{n-1} > 0$ such that $\widetilde{\mathcal{G}}^i_{n-1} \simeq \widetilde{\mathcal{G}}^{i_{n-1}}_{n-1}$ for all $i \geq i_{n-1}$.

Proof. Since the state-set in A_i remains constant, there is some i_{n-1} where no more transitions are added at order-n. That $\widetilde{\mathcal{G}}^i_{n-1} \simeq \widetilde{\mathcal{G}}^{i_{n-1}}_{n-1}$ for all $i \geq i_{n-1}$ follows since the contents of any $\widetilde{G}^i_{(q,Q')}$ and $\widetilde{G}^{i_{n-1}}_{(q,Q')}$ are derived from the same transition structure.

The following lemma shows that, once a fixed point has been reached at order-$(l+1)$, we can fix the state-set at order-l.

Lemma 1. *Suppose we have a sequence of automata $\mathcal{G}^0_l, \mathcal{G}^1_l, \ldots$ and associated sets $\widetilde{\mathcal{G}}^0_l, \widetilde{\mathcal{G}}^1_l, \ldots$. Further suppose there exists an i_l such that for all $i \geq i_l$ we have $\widetilde{\mathcal{G}}^i_l \simeq \widetilde{\mathcal{G}}^{i_l}_l$. We can define a sequence of automata $\hat{\mathcal{G}}^{i_l}_l, \hat{\mathcal{G}}^{i_l+1}_l, \ldots$ such that the order-l state-set in $\hat{\mathcal{G}}^i_l$ remains constant. The following are equivalent for all w,*

1. The run $g^{i_l}_{(q,Q')} \xrightarrow{w}_i Q$ with $Q \subseteq \mathcal{Q}_f$ exists in $\hat{\mathcal{G}}^i_l$ for some i.
2. The run $g^{i'}_{(q,Q')} \xrightarrow{w}_{i'} Q''$ with $Q'' \subseteq \mathcal{Q}_f$ exists in $\mathcal{G}^{i'}_l$ for some i'.

We use $\hat{\mathcal{G}}^{i+1}_l = T_{\widetilde{\mathcal{G}}^{i_l}_l[i_l/i_l-1]}(\hat{\mathcal{G}}^i_l)$ to construct the sequence $\hat{\mathcal{G}}^{i_l}_l, \hat{\mathcal{G}}^{i_l+1}_l, \ldots$. Intuitively, since the transitions from the states introduced to define $\mathcal{G}^i_l$ for $i \geq i_l$ are derived from similar sets, we can compress the subsequent repetition into a single set of new states with "self-loops".

Once the state-set has been fixed at order-l, we will reach another fixed point. In this way the fixed points cascade. The proof is similar to Property 2.

Property 3. For a sequence of automata $\mathcal{G}^0_l, \mathcal{G}^1_l, \ldots$ such that the state-set at order-l of $\mathcal{G}^i_l$ remains constant there exists $i_{l-1} > 0$ such that $\widetilde{\mathcal{G}}^i_{l-1} \simeq \widetilde{\mathcal{G}}^{i_{l-1}}_{l-1}$ for all $i \geq i_{l-1}$.

We have the following algorithm for constructing A_*:

1. Given A_0, iterate $A_{i+1} = T_{\mathcal{D}}(A_i)$ until the fixed point $A_{i_{n-1}}$ is reached.
2. For $l = n-1$ down to $l = 1$: iterate $\mathcal{G}^{i+1}_l = T_{\widetilde{\mathcal{G}}^{i_l}_l[i_l/i_l-1]}(\mathcal{G}^i_l)$ to generate the fixed point $\mathcal{G}^{i_{l-1}}_l$ from $\mathcal{G}^{i_l}_l$.
3. Let $\mathcal{G}^*_1 = \mathcal{G}^{i_0}_1$. For $l = 2$ to $l = n-1$: construct $\mathcal{G}^*_l$ by labelling the transitions of $\mathcal{G}^{i_{l-1}}_l$ with automata derived from $\mathcal{G}^*_{l-1}$. Then, construct A_* analogously.

Property 4. There exists an automaton A_* which is sound and complete with respect to $A_0, A_1, \ldots$ and hence computes the set $Pre^*(C_{Init})$.

Remark 1. We claim our algorithm runs in n-EXPTIME. Intuitively, when the state-set $\mathcal{Q}$ is fixed at order-1 of the store automaton, we add at most $\mathcal{O}(2^{|\mathcal{Q}|})$ transitions (since we never remove states, it is this final stage that dominates the complexity). At orders $l > 1$ we add at most $\mathcal{O}(2^{|\mathcal{Q}|})$ new transitions, which exponentially increases the state-set at order-$(l-1)$. Hence, the algorithm runs in n-EXPTIME.

4 Applications

4.1 Model Checking Linear-Time Temporal Logics

Bouajjani *et al.* use their backwards reachability algorithm to provide a model checking algorithm for linear-time temporal logics over PDSs [2]. In this Section we show that this work permits a simple generalisation to higher-order PDSs.

We form the product of the higher-order PDS and the Büchi automaton corresponding to the negation of ϕ [26, 16, 25]. This gives us a higher-order Büchi PDS with a set $\mathcal{F}$ of accepting control states. Thus, model checking reduces to the non-emptiness problem for higher-order Büchi PDSs. Let $[{}^1a]^1$ denote the order-1 stack consisting of a single character a and $[{}^la]^l$ for $l > 1$ denote the stack consisting of a single order-$(l-1)$ stack $[{}^{(l-1)}a]^{(l-1)}$.

Proposition 1. *Let c be a configuration of an order-n Büchi PDS BP. BP has an accepting run from c iff there exist distinct configurations $\langle p^1, [^n a]^n \rangle$ and $\langle p^1, \gamma_2 \rangle$ with $top_1(\gamma_2) = a$ and configuration $\langle p^f, \gamma_1 \rangle$ such that $p^f \in \mathcal{F}$ and,*

1. *$c \overset{*}{\hookrightarrow} \langle p^1, \gamma_3 \rangle$ for some w_3 with $top_1(\gamma_3) = a$, and*
2. *$\langle p^1, [^n a]^n \rangle \overset{*}{\hookrightarrow} \langle p^f, \gamma_1 \rangle \overset{*}{\hookrightarrow} \langle p^1, \gamma_2 \rangle$*

Using this reduction, we have the following theorem and corollary.

Theorem 2. *Given a order-n Büchi PDS $BP = (\mathcal{P}, \mathcal{D}, \Sigma, \mathcal{F})$, we can calculate in n-EXPTIME the set of configurations C such that from all $c \in C$ there is an accepting run of BP.*

Corollary 1. *Given an order-n PDS $(\mathcal{P}, \mathcal{D}, \Sigma)$ and a formula ϕ of an ω-regular logic, we can calculate in $(n + 2)$-EXPTIME the set of configurations C of $(\mathcal{P}, \mathcal{D}, \Sigma)$ such that every run from each $c \in C$ satisfies ϕ.*

There is one exponential blow-up in the construction of BP and one more in a final complementation step. We can test $c \notin C$ rather than $c \in C$ for an $(n + 1)$-EXPTIME algorithm.

4.2 Reachability Games

Our algorithm may be used to compute the winning region for a player in a two-player reachability game over an order-n PDS. This generalises a result due to Cachat [21]. We call our players Eloise and Abelard.

Given an order-n PDS $(\mathcal{P}, \mathcal{D}, \Sigma)$, a *Pushdown Reachability Game* $(\mathcal{P}, \mathcal{D}, \Sigma, \mathcal{R})$ over the order-n PDS is given by a partition $\mathcal{P} = \mathcal{P}_A \uplus \mathcal{P}_E$ and a set $\mathcal{R}$ of configurations considered winning for Eloise. The *winning region* for Eloise can be characterised using an *attractor* $Attr_E(\mathcal{R})$. Conversely, the winning region for Abelard is $\overline{Attr_E(\mathcal{R})}$. We can use backwards-reachability for APDSs to calculate $Attr_E(\mathcal{R})$, and hence the winning regions of both Abelard and Eloise. This is a straightforward encoding of the pushdown reachability game as a APDS. We form $\mathcal{R}'$ from $\mathcal{R}$ by adding the regular set of all configurations from which Abelard cannot make a move (and therefore loses the game).

Theorem 3. *Given an order-n pushdown reachability game, where $\mathcal{R}$ is a regular set of configurations, and an order-n APDS as defined above, $Attr_E(\mathcal{R})$ is regular and equivalent to $Pre^*(\mathcal{R}')$ less all configurations of the form $\langle p, \triangledown \rangle$. Hence, computing the winning regions of the game is n-EXPTIME.*

4.3 Model-Checking Branching-Time Temporal Logics

Generalising a further result of Bouajjani *et al.* [2], we have that backwards-reachability for higher-order APDSs may be used to solve the model-checking problem for the alternation-free (propositional) μ-calculus over higher-order PDSs. Common logics such as CTL are sub-logics of the alternation-free μ-calculus. The algorithm uses properties of the alternation-free μ-calculus to reduce the model checking problem to a number of reachability games. We state the following theorem:

Theorem 4. *Given an order-n PDS $(\mathcal{P}, \mathcal{D}, \Sigma)$ and a formula ϕ of the alternation-free μ-calculus, we can compute the regular set of configurations satisfying ϕ in $((n_\phi \cdot n) + 1)$-EXPTIME, where n_ϕ is the length of ϕ.*

The $((n_\phi \cdot n) + 1)$-EXPTIME result is due to a final complementation step. As before, this may be avoided, yeilding an $(n_\phi \cdot n)$-EXPTIME algorithm.

5 Conclusion

Given an automaton representation of a regular set of higher-order APDS configurations C_{Init}, we have shown that the set $Pre^*(C_{Init})$ is regular and computable via automata-theoretic methods. This builds upon previous work on pushdown systems [2] and higher-order context-free pushdown processes [1]. The main innovation of this generalisation is the careful management of a complex automaton construction. This allows us to identify a sequence of cascading fixed points, resulting in a terminating algorithm.

Our result has many applications. We have shown that it can be used to provide a solution to the model checking problem for linear-time temporal logics and the alternation-free μ-calculus. In particular we compute the set of configurations of a higher-order PDS satisfying a given constraint. We also show that the winning regions can be computed for a reachability game played over an higher-order PDS.

There are several possible extensions to this work. Firstly, we intend to complete the complexity analysis with corresponding hardness results. Secondly, we plan to investigate the applications of this work to higher-order pushdown games with more general winning conditions. In his PhD thesis, Cachat adapts the reachability algorithm of Bouajjani *et al.* [2] to calculate the winning regions in Büchi games over pushdown processes [21]. It is likely that our work will permit similar extensions. Finally, we intend to generalise this work to higher-order collapsible pushdown automata, which can be used to study higher-order recursion schemes [24, 7]. This may provide the first steps into the study of games over these structures.

Acknowledgements. We thank Olivier Serre and Arnaud Carayol for helpful discussions.

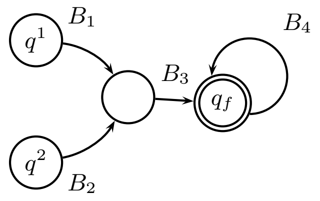
References

1. A. Bouajjani and A. Meyer. Symbolic Reachability Analysis of Higher-Order Context-Free Processes. In *Proc. FSTTCS'04*, 2004. LNCS 3328.
2. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. CONCUR '97*, pp. 135–150, 1997.
3. A. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with the unboundedness and regular conditions. In *Proc. FSTTCS'03*, pages 88–99, 2003.
4. A. Carayol. Regular sets of higher-order pushdown stacks. In *Proc. MFCS*, pages 168–179, 2005.

5. A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proc. FSTTCS*, pages 112–123, 2003.
6. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
7. A. S. Murawski and C.-H. L. Ong. Collapsable pushdown automata and recursion schemes, 2006. Unpublished.
8. C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proc. FSTTCS'04*, pages 408–420. 2004. LNCS 3328.
9. D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. MFCS'02*, pages 165–176, 2002. LNCS 2420.
10. D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
11. H. Gimbert. Parity and exploration games on infinite graphs. In *Proc. CSL'04*, pages 56–70, 2004. LNCS 3210.
12. M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. Preprint, 54 pages, `www.comlab.ox.ac.uk/oucl/work/matthew.hague/FoSSaCS07-long.pdf`, 2006.
13. I. Walukiewicz. Pushdown processes: Games and model checking. In *Proc. CAV '96*, pages 62–74. 1996.
14. J. A. Brzozowski and E. L. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theor. Comput. Sci.*, 10:19–35, 1980.
15. K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *Proc. FoSSaCS*, pages 490–504, 2005.
16. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.
17. A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 15:1170–1174, 1976.
18. O. Serre. Note on winning positions on pushdown games with ω-regular conditions. *Information Processing Letters*, 85:285–291, 2003.
19. O. Serre. Games with winning conditions of high Borel complexity. In *Proc. ICALP'04*, pages 1150–1162. Springer-Verlag, 2004. LNCS 3142.
20. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proc. LICS '06*, pages 81–90. IEEE Computer Society, 2006.
21. T. Cachat. *Games on Pushdown Graphs and Extensions.* PhD thesis, RWTH Aachen, 2003.
22. T. Cachat, J. Duparc, and W. Thomas. Solving pushdown games with a Σ_3 winning condition. In *Proc. CSL'02*, pages 322–336. Springer-Verlag, 2002. LNCS 2471.
23. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *Proc. FoSSaCS '02*, pages 205–222, London, UK, 2002. Springer-Verlag.
24. T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *Proc. ICALP '05*, pages 1450–1461, 2005.
25. M. Y. Vardi. A temporal fixpoint calculus. In *Proc. POPL '88*, pages 250–259, New York, NY, USA, 1988. ACM Press.
26. W. Thomas. Automata on infinite objects. *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 133–191, 1990.

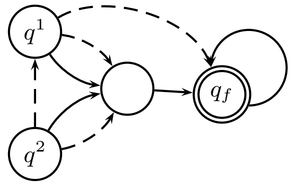
A Example

Fix the following two-state second-order PDS and 2-store multi-automaton A_0 with some B_1, B_2, B_3 and B_4.

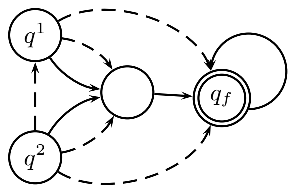


$$\begin{aligned}
d_1 &= (p^1, a, push_2, p^1) \\
d_2 &= (p^1, a, push_\varepsilon, p^1), \\
d_3 &= (p^2, a, push_w, p^1) \\
d_4 &= (p^2, a, pop_2, p^1)
\end{aligned}$$

We iterate $T_{\mathcal{D}}$ until a fixed point is reached. The dashed arrows are labelled with the appropriate $G^i_{(\ldots)}$.

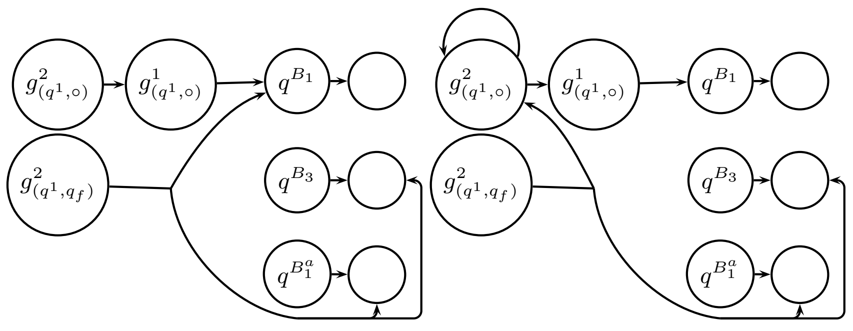


$$\begin{aligned}
\widetilde{G}^1_{(q^1,\circ)} &= \left\{\underline{\{(a, push_\varepsilon, B_1)\}}\right\} \\
\widetilde{G}^1_{(q^1,q_f)} &= \{\{B_1^a, B_1, B_3\}\} \\
\widetilde{G}^1_{(q^2,\circ)} &= \{\{(a, push_w, B_1)\}\} \\
\widetilde{G}^1_{(q^2,q^1)} &= \{\{B_1^a\}\}
\end{aligned}$$



$$\begin{aligned}
\widetilde{G}^2_{(q^1,\circ)} &= \left\{\{(a, push_\varepsilon, B_1)\}, \underline{\{(a, push_\varepsilon, \widetilde{G}^1_{(q^1,\circ)})\}}\right\} \\
\widetilde{G}^2_{(q^1,q^f)} &= \left\{\begin{array}{l} \{B_1^a, B_1, B_3\}, \{(a, push_\varepsilon, \widetilde{G}^1_{(q^1,q_f)})\}, \\ \underline{\{B_1^a, \widetilde{G}^1_{(q^1,q_f)}, B_4\}}, \{B_1^a, \widetilde{G}^1_{(q^1,\circ)}, B_3\} \end{array}\right\} \\
\widetilde{G}^2_{(q^2,\circ)} &= \left\{\{(a, push_w, B_1)\}, \{(a, push_w, \widetilde{G}^1_{(q^1,\circ)})\}\right\} \\
\widetilde{G}^2_{(q^2,q^1)} &= \{\{B_1^a\}\} \\
\widetilde{G}^2_{(q^2,q_f)} &= \left\{\{(a, push_w, \widetilde{G}^1_{(q^1,q_f)})\}\right\}
\end{aligned}$$

The following diagrams show excerpts of $\mathcal{G}_1^2$ and $\mathcal{G}_1^*$ respectively. All edges are labelled a.



The relevent S are underlined in the tables above. The new edges in the second diagram derive from $S[2/1]$ for the appropriate S. The branching edge is removed from the first diagram for clarity. The new branching edge is added after the self-loop.