Symbolic Reachability Analysis of Higher-Order Context-Free Processes

Ahmed Bouajjani and Antoine Meyer

LIAFA, Univ. of Paris 7, Case 7014, 2 place Jussieu 75251, Paris Cedex 5, France
{abou,ameyer}@liafa.jussieu.fr

Abstract. We consider the problem of symbolic reachability analysis of higher-order context-free processes. These models are generalizations of the context-free processes (also called BPA processes) where each process manipulates a data structure which can be seen as a nested stack of stacks. Our main result is that, for any higher-order context-free process, the set of all predecessors of a given regular set of configurations is regular and effectively constructible. This result generalizes the analogous result which is known for level 1 context-free processes. We show that this result holds also in the case of backward reachability analysis under a regular constraint on configurations. As a corollary, we obtain a symbolic model checking algorithm for the temporal logic $\mathsf{E}(\mathsf{U},\mathsf{X})$ with regular atomic predicates, i.e., the fragment of CTL restricted to the EU and EX modalities.

1 Introduction

Pushdown systems and their related decision and algorithmic analysis problems (reachability analysis, model checking, games solving and control synthesis, etc) have been widely investigated in the last few years [11, 7, 22, 5, 15, 8, 2]. This recent intensive research effort is mainly motivated by the fact that pushdown systems are quite natural models for sequential programs with recursive procedure calls (see e.g., [16, 14]), and therefore they are particularly relevant for software verification and design.

Higher-order pushdown systems [13] (HPDS) are generalizations of these models in which the elements appearing in a pushdown stack are no longer single letters but stacks themselves. We call this kind of nested stack structures *higher-order stores*. Stores of level 1 are sequences of symbols in some finite alphabet (those are standard pushdown stacks), and stores of level $n+1$ are sequences of stores of level n, for any $n > 1$. The operations allowed on these structures are (1) the usual *push* and *pop* operations on the top-most level 1 store, (2) higher-order *push* and *pop* operations allowing to *duplicate* or *erase* the top-most level k store of any given level $k \leq n$.

This general model is quite powerful and has nice structural characterizations [12, 10]. It has been in particular proved in [19] that HPDS are equivalent to (safe) higher-order recursive program schemes. Interestingly, it has also been proved that the monadic second-order theory of an infinite tree generated by a

HPDS is decidable [19, 11], which generalizes the analogous result for pushdown systems proved by Muller and Schupp [20]. Also, it has been proved that parity games can be solved for HPDS [9], which generalizes the result of Walukiewicz for pushdown systems [22]. These results actually show that model checking is decidable for HPDS. However, they only allow to check that a property holds in a *single* initial configuration and they do not provide a procedure for computing a representation of the set of configurations which satisfy some given property (the satisfiability set of the property).

The basic step toward defining an algorithm which effectively computes the satisfiability sets of properties is to provide a procedure for computing the set of backward reachable configurations from a given set of configurations, i.e. their set of predecessors. In fact, the computation of forward- or backward-reachable sets is a fundamental problem in program analysis and in verification.

Since HPDS are infinite-state systems, to solve this problem we need to consider *symbolic representation structures* which (1) provide finite representations of potentially infinite sets of configurations, and (2) enjoy closure properties and decidability properties which are necessary for their use in verification. Minimal requirements in this regard are closure under union and intersection, and decidability of the emptiness and inclusion problems.

A natural class of symbolic representations for infinite-state systems is the class of finite-state automata. Recently, many works (including several papers on the so-called *regular model-checking*) have shown that finite-state automata are suitably generic representation structures, which allow to uniformly handle a wide variety of systems including pushdown systems, FIFO-channel systems, parameterized networks of processes, counter systems, etc. [5, 3, 18, 1, 23, 6, 4, 17].

In particular, for the class of pushdown systems, automata-based symbolic reachability analysis techniques have been developed and successfully applied in the context of program analysis [5, 15, 21]. Our aim in this paper is to extend this approach to a subclass of HPDS called *higher-order context-free processes* (HCFP for short). This class corresponds to the higher order extension of the well-known context-free processes (also called BPA processes). HCFP can actually be seen as HPDS with a single control state, similarly to level 1 CFP which are equivalent to level 1 PDS with a single control state. The contributions of our paper can be summarized as follows.

First, we observe that, due to the duplication operation, the set of immediate successors (i.e. the *post* image) of a given regular set of configurations is in general *not* regular, but it is always a context-sensitive set.

Then, we prove that, and this is our main result, for every HCFP of any level, the set of all predecessors (i.e. the pre^* image) of any given regular set of configurations is a regular set and effectively constructible. As a corollary of this result, we obtain a symbolic model checking algorithm (an algorithm which computes the set of all configurations satisfying a formula) for the temporal logic $\mathsf{E}(\mathsf{F}, \mathsf{X})$ with regular atomic predicates, i.e., the fragment of CTL with the modalities EF (there exist path where a property eventually holds) and EX (there exist an immediate successor satisfying some property).

Furthermore, we extend our construction of the pre^* images by showing that the set of predecessors under a regular constraint (i.e., the set of all predecessors reachable by computations which stay in some given regular set of configurations) is also regular and effectively constructible. For that, we use representation structures which can be seen as alternating finite-state automata. This result allows us to provide a symbolic model checking algorithm for the logic $\mathsf{E}(\mathsf{U}, \mathsf{X})$ with regular atomic predicates, i.e., the fragment of CTL with the operators EU (exists until) and EX (exists next).

The structure of this paper is the following. In the next two sections, we introduce higher-order stores and the model of higher-order context-free processes. We also provide a symbolic representation for (infinite) *regular* sets of stores using a certain type of finite automata. Then, for the sake of readability, we first present our algorithm for computing the unconstrained *pre* and pre^* sets of a regular set of stores (Section 4), before extending it to the case of pre^* sets constrained by a regular set C (Section 5). Due to lack of space, additional definitions and detailed proofs can be found in the full version of this paper[1].

2 Higher-order Context-free Processes

We introduce a class of models we call *higher-order context-free processes*, which generalize context-free processes (CFP) and are a subclass of higher-order pushdown systems (HPDS). They manipulate data structures called *higher-order stores.*

Definition 2.1 (Higher-order store). *The set $\mathcal{S}_1$ of* level 1 stores *(or* 1-stores*) over store alphabet Γ is the set of all sequences $[a_1 \dots a_l] \in [\Gamma^*]$. For $n \geq 2$, the set $\mathcal{S}_n$ of* level n stores *(or* n-stores*) over Γ is the set of all sequences $[s_1 \dots s_l] \in [{\mathcal{S}_{n-1}}^+]$.*

The following operations are defined on 1-stores:

$$\begin{aligned} push_1^w([a_1 \dots a_l]) &= [wa_2 \dots a_l] && \text{for all } w \in \Gamma^*, \\ top_1([a_1 \dots a_l]) &= a_1. \end{aligned}$$

We will sometimes abbreviate $push_1^\varepsilon$ as pop_1. The following operations are defined on n-stores ($n > 1$):

$$\begin{aligned} push_1^w([s_1 \dots s_l]) &= [push_1^w(s_1) \dots s_l] \\ push_k([s_1 \dots s_l]) &= [push_k(s_1) \dots s_l] && \text{if } k \in [2, n[, \\ push_n([s_1 \dots s_l]) &= [s_1 s_1 \dots s_l] \\ pop_k([s_1 \dots s_l]) &= [pop_k(s_1) \dots s_l] && \text{if } k \in [2, n[, \\ pop_n([s_1 \dots s_l]) &= [s_2 \dots s_l] && \text{if } l > 1, \text{ else undefined}, \\ top_k([s_1 \dots s_l]) &= top_k(s_1) && \text{if } k \in [1, n[, \\ top_n([s_1 \dots s_l]) &= s_1. \end{aligned}$$

[1] available at http://www.liafa.jussieu.fr/~ameyer/.

We denote by O_n the set of operations consisting of:

$$\{\, push_k, pop_k \mid k \in [2,n] \,\} \cup \{\, push_1^w \mid w \in \Gamma^* \,\}.$$

We say that operation o is of level n, written $l(o) = n$, if o is either $push_n$ or pop_n, or $push_1^w$ if $n = 1$. We can now define the model studied in this paper.

Definition 2.2. *A* higher-order context-free process *of level n (or n-HCFP) is a pair $\mathcal{H} = (\Gamma, \Delta)$, where Γ is a finite alphabet and $\Delta \in \Gamma \times O_n$ is a finite set of transitions. A configuration of $\mathcal{H}$ is a n-store over Γ. $\mathcal{H}$ defines a transition relation $\underset{\mathcal{H}}{\hookrightarrow}$ between n-stores (or $\hookrightarrow$ when $\mathcal{H}$ is clear from the context), where*

$$s \underset{\mathcal{H}}{\hookrightarrow} s' \iff \exists (a, o) \in \Delta \text{ such that } top_1(s) = a \text{ and } s' = o(s).$$

The level $l(d)$ of a transition $d = (a, o)$ is simply the level of o. Let us give a few more notations concerning HCFP computations. Let $H = (\Gamma, \Delta)$ be a n-HCFP. A *run* of $\mathcal{H}$ starting from some store s_0 is a sequence $s_0 s_1 s_2 \ldots$ such that for all $i \geq 0$, $s_i \hookrightarrow s_{i+1}$. The reflexive and transitive closure of $\hookrightarrow$ is written $\overset{*}{\hookrightarrow}$ and called the *reachability* relation. For a given set C of n-stores, we also define the *constrained transition* relation $\hookrightarrow_C = \hookrightarrow \cap (C \times C)$, and its reflexive and transitive closure $\overset{*}{\hookrightarrow}_C$. Now for any set of n-stores S, we consider the sets:

$$post_{\mathcal{H}}[C](S) = \{\, s \mid \exists s' \in S,\ s' \hookrightarrow_C s \,\},$$

$$post^*_{\mathcal{H}}[C](S) = \{\, s \mid \exists s' \in S,\ s' \overset{*}{\hookrightarrow}_C s \,\},$$

$$pre_{\mathcal{H}}[C](S) = \{\, s \mid \exists s' \in S,\ s \hookrightarrow_C s' \,\},$$

$$pre^*_{\mathcal{H}}[C](S) = \{\, s \mid \exists s' \in S,\ s \overset{*}{\hookrightarrow}_C s' \,\}.$$

When C is the set $\mathcal{S}_n$ of all n-stores, we omit it in notations and simply write for instance $pre_{\mathcal{H}}(S)$ instead of $pre_{\mathcal{H}}[C](S)$. We will also omit $\mathcal{H}$ when it is clear from the context. When $\mathcal{H}$ consists of a single transition d, we may write $pre_d(S)$ instead of $pre_{\mathcal{H}}(S)$.

3 Sets of Stores and Symbolic Representation

To be able to design symbolic verification techniques over higher-order context-free processes, we need a way to finitely represent infinite sets (or languages) of configurations. In this section we present the sets of configurations (i.e. sets of stores) we consider, as well as the family of automata which recognize them.

A n-store $s = [s_1 \ldots s_l]$ over Γ is associated to a word $w(s) = [w(s_1) \ldots w(s_l)]$, in which store letters in Γ only appear at nesting depth n. A set of stores over Γ is called *regular* if its set of associated words is accepted by a finite automaton over $\Gamma' = \Gamma \cup \{\, [\, , \,]\, \}$, which in this case we call a *store automaton*. We will often make no distinction between a store s and its associated word $w(s)$. Due to the nested structure of pushdown stores, it will sometimes be more convenient to characterize sets of stores using *nested store automata*.

Definition 3.1. *A level* 1 *nested store automaton is a finite automaton whose transitions have labels in* Γ*. A nested store automaton of level* $n \geq 2$ *is a finite automaton whose transitions are labelled by level* $n-1$ *nested automata over* Γ*.*

The existence of a transition labelled by $\mathcal{B}$ between two control states p and q in a finite automaton $\mathcal{A}$ is written $p \xrightarrow[\mathcal{A}]{\mathcal{B}} q$, or simply $p \xrightarrow{\mathcal{B}} q$ when $\mathcal{A}$ is clear from the context. Let $\mathcal{A} = (Q, \Gamma, \delta, q_0, q_f)$ be a level n nested automaton[2] with $n \geq 2$. The level k language of $\mathcal{A}$ for $k \in [1, n]$ is defined recursively as:

$$\begin{aligned} L_k(\mathcal{A}) &= \{ [L_k(\mathcal{A}_1) \ldots L_k(\mathcal{A}_l)] \mid [\mathcal{A}_1 \ldots \mathcal{A}_l] \in L_n(\mathcal{A}) \} && \text{if } k < n, \\ L_k(\mathcal{A}) &= \{ [\mathcal{A}_1 \ldots \mathcal{A}_l] \mid q_0 \xrightarrow[\mathcal{A}]{\mathcal{A}_1} \ldots \xrightarrow[\mathcal{A}]{\mathcal{A}_l} q_f \} && \text{if } k = n. \end{aligned}$$

For simplicity, we often abbreviate $L_1(\mathcal{A})$ as $L(\mathcal{A})$. We say a nested automaton $\mathcal{B}$ occurs in $\mathcal{A}$ if $\mathcal{B}$ labels a transition of $\mathcal{A}$, or occurs in the label of one. Level n automata are well suited to representing sets of n-stores, but have the same expressive power as standard level 1 store automata.

Proposition 3.2. *The store languages accepted by nested store automata are the regular store languages.*

Moreover, regular n-store languages are closed under union, intersection and complement in $\mathcal{S}_n$. We define for later use the set of automata $\{ \mathcal{A}_a^n \mid a \in \Gamma,\ n \in \mathbb{N} \}$ such that for all a and n, $L(\mathcal{A}_a^n) = \{ s \in \mathcal{S}_n \mid top_1(s) = a \}$. We also write $\mathcal{A} \times \mathcal{B}$ the product operation over automata such that $L(\mathcal{A} \times \mathcal{B}) = L(\mathcal{A}) \cap L(\mathcal{B})$.

4 Symbolic Reachability Analysis

Our goal in this section is to investigate effective techniques to compute the sets $pre(S)$, $post(S)$, $pre^*(S)$ and $post^*(S)$ for a given n-HCFP $\mathcal{H}$, in the case where S is a regular set of stores. For level 1 pushdown systems, it is a well-known result that both $pre^*_{\mathcal{H}}(S)$ and $post^*_{\mathcal{H}}(S)$ are regular. We will see that this is still the case for $pre(S)$ and $pre^*(S)$ in the higher-order case, but not for $post(S)$ (hence not for $post^*(S)$ either).

4.1 Forward Reachability

Proposition 4.1. *Given a n-HCFP* $\mathcal{H}$ *and a regular set of n-stores* S*, the set* $post(S)$ *is in general not regular. This set is a context-sensitive language.*

Proof. Let $post_{(a,o)}(S)$ denote the set $\{ s' \mid \exists s \in S,\ top_1(s) = a \ \wedge\ s' = o(s) \}$. Suppose S is a regular set of n-stores, then if $d = (a, push_1^w)$ or $d = (a, pop_k)$, it is not difficult to see that $post_{(a,o)}(S)$ is regular. However, if $d = (a, push_k)$ with $k > 1$, then $post_{(a,o)}(S)$ is the set $\{ [^{n-k+1} t\, t\, w \mid [^{n-k+1} t\, w \in S \}$. It can be shown using the usual pumping arguments that this set is not regular, because of the duplication of t. However, one can straightforwardly build a linearly bounded Turing machine recognizing this set. $\square$

[2] Note that we only consider automata with a single final state.

4.2 Backward Reachability

We first propose a transformation on automata which corresponds to the *pre* operation on their language. In a second time, we extend this construction to deal with the more difficult computation of pre^* sets.

Proposition 4.2. *Given a n-HCFP $\mathcal{H}$ and a regular set of n-stores S, the set $pre(S)$ is regular and effectively computable.*

We introduce a construction which, for a given HCFP transition d and a given regular set of n-stores S recognized by a level n nested automaton $\mathcal{A}$, allows us to compute a nested automaton $\mathcal{A}'_d$ recognizing the set $pre(S)$ of direct predecessors of S by d. This construction is a transformation over nested automata, which we call T_d. We define $\mathcal{A}'_d = T_d(\mathcal{A}) = (Q', \Gamma, \delta', q'_0, q_f)$ as follows.
If $l(d) < n$, we propagate the transformation to the first level $n-1$ automaton encountered along each path. We thus have $Q' = Q$, $q'_0 = q_0$ and

$$\delta' = \{\, q_0 \xrightarrow{T_d(\mathcal{A}_1)} q_1 \mid q_0 \xrightarrow[\mathcal{A}]{\mathcal{A}_1} q_1 \,\} \cup \{\, q \xrightarrow{\mathcal{B}} q' \mid q \xrightarrow[\mathcal{A}]{\mathcal{B}} q' \wedge q \neq q_0 \,\}.$$

If $l(d) = n$, we distinguish three cases according to the nature of d:

1. If $d = (a, push_1^w)$, then $Q' = Q \cup \{q'_0\}$ and $\delta' = \delta \cup \{\, q'_0 \xrightarrow{a} q_1 \mid q_0 \xrightarrow[\mathcal{A}]{w} q_1 \,\}$.
2. If $d = (a, push_n)$ and $n > 1$, then $Q' = Q \cup \{q'_0\}$ and
$\delta' = \delta \cup \{\, q'_0 \xrightarrow{\mathcal{B}} q_2 \mid \exists q_1,\ q_0 \xrightarrow[\mathcal{A}]{\mathcal{A}_1} q_1 \xrightarrow[\mathcal{A}]{\mathcal{A}_2} q_2 \,\}$ where $\mathcal{B} = \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_a^{(n-1)}$.
3. If $d = (a, pop_n)$, then $Q' = Q \cup \{q'_0\}$ and $\delta' = \delta \cup \{\, q'_0 \xrightarrow{\mathcal{A}_a^{(n-1)}} q_0 \,\}$.

It is not difficult to prove that $L(\mathcal{A}'_d) = pre_d(L(\mathcal{A}))$. Hence, if Δ is the set of transitions of $\mathcal{H}$, then we have $pre(S) = pre(L(\mathcal{A})) = \bigcup_{d\in\Delta} L(\mathcal{A}'_d)$.

This technique can be extended to compute the set $pre^*(S)$ of all predecessors of a regular set of stores S.

Theorem 4.3. *Given a n-HCFP $\mathcal{H}$ and a regular set of n-stores S, the set $pre^*(S)$ is regular and effectively computable.*

To compute $pre^*(S)$, we have to deal with the problem of termination. A simple iteration of our previous construction will in general not terminate, as each step would add control states to the automaton. As a matter of fact, even the sequence $(pre^i(S))_{i\geq 0}$, defined as $pre^0(S) = S$ and for all $n \geq 1$ $pre^n(S) = pre^{n-1}(S) \cup pre(pre^{n-1}(S))$, does not reach a fix-point in general. For instance, if $d = (a, pop_1)$, then for all n, $pre^n([a]) = \{\, [a^i] \mid i \leq n \,\} \neq pre^{n+1}([a])$.

To build $pre^*(S)$ for some regular S, we modify the previous construction in order to keep constant the number of states in the nested automaton we manipulate. The idea, instead of creating new control states, is to add edges to the automaton until saturation, eventually creating loops to represent at once multiple applications of a HCFP transition. Then, we prove that this new algorithm terminates and is correct.

Let us first define operation T_d for any n-HCFP transition d (see Figure 1 for an illustration). Let $\mathcal{A} = (Q, \Gamma, \delta, q_0, q_f)$ and $\mathcal{A}' = (Q, \Gamma, \delta', q_0, q_f)$ be nested n-store automata over $\Gamma' = \Gamma \cup \{ [,] \}$, and d a n-HCFP transition. We define $\mathcal{A}' = T_d(\mathcal{A})$ as follows.
If the level of d is less than n, then we simply propagate the transformation to the first level $n-1$ automaton encountered along each path:

$$\delta' = \{ q_0 \xrightarrow{T_d(\mathcal{A}_1)} q_1 \mid q_0 \xrightarrow[\mathcal{A}]{\mathcal{A}_1} q_1 \} \cup \{ q \xrightarrow{\mathcal{B}} q' \mid q \xrightarrow[\mathcal{A}]{\mathcal{B}} q' \wedge q \neq q_0 \}.$$

If $l(d) = n$ then as previously we distinguish three cases according to d:

1. If $n = 1$ and $d = (a, push_1^w)$, then $\delta' = \delta \cup \{ q_0 \xrightarrow{a} q_1 \mid q_0 \xrightarrow[\mathcal{A}]{w} q_1 \}$.
2. If $d = (a, push_n)$ for some $n > 1$, then
$\delta' = \delta \cup \{ q_0 \xrightarrow{\mathcal{B}} q_2 \mid \exists q_1,\ q_0 \xrightarrow[\mathcal{A}]{\mathcal{A}_1} q_1 \xrightarrow[\mathcal{A}]{\mathcal{A}_2} q_2 \}$ where $\mathcal{B} = \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_a^{(n-1)}$.
3. If $d = (a, pop_n)$, then $\delta' = \delta \cup \{ q_0 \xrightarrow{\mathcal{A}_a^{(n-1)}} q_0 \}$

Suppose $H = (\Gamma, \Delta)$ with $\Delta = \{ d_0, \ldots, d_{l-1} \}$. Given an automaton $\mathcal{A}$ such that $S = L(\mathcal{A})$, consider the sequence $(\mathcal{A}_i)_{i \geq 0}$ defined as $\mathcal{A}_0 = \mathcal{A}$ and for all $i \geq 0$ and $j = i \mod l$, $\mathcal{A}_{i+1} = T_{d_j}(\mathcal{A}_i)$. In order to obtain the result, we have to prove that this sequence always reaches a fix-point (Lemma 4.4) and this fix-point is an automaton actually recognizing $pre^*(S)$ (Lemmas 4.5 and 4.6).

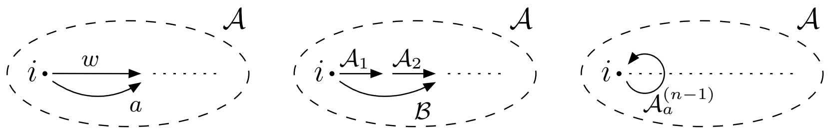

Fig. 1. transformation $T_d(\mathcal{A})$ for $d = (a, push_1^w)$, $(a, push_k)$ and (a, pop_k).

Lemma 4.4 (Termination). *For all nested n-store automaton $\mathcal{A}$ and n-HCFP $\mathcal{H} = (\Gamma, \Delta)$, the sequence $(\mathcal{A}_i)_{i \geq 0}$ defined with respect to $\mathcal{A}$ eventually stabilizes: $\exists k \geq 0,\ \forall k' \in \Delta,\ \mathcal{A}_{k'} = \mathcal{A}_k$, which implies $L(\mathcal{A}_k) = \bigcup_{i \geq 0} L(\mathcal{A}_i)$.*

Proof. First, notice that for all d, T_d does not change the set of control states of any automaton occurring in $\mathcal{A}$, and only adds transitions. This means $(\mathcal{A}_i)_{i \geq 0}$ is monotonous in the size of each $\mathcal{A}_i$.

To establish the termination of the conctruction, we prove that the number of transitions which can be added to $\mathcal{A}_0$ is finite. Note that by definition of T_d, the number of states of each $\mathcal{A}_i$ is constant. Moreover, each new transition originates from the initial state of the automaton it is added to. Hence, the total number of transitions which can be added to a given automaton is equal to $|V_n| \cdot |Q|$, where V_n is the level n vocabulary and Q its set of states. Since $|Q|$ does not change, we only have to prove that V_n is finite for all n. If $n = 1$, $V_1 = \Gamma$, and the property holds. Now suppose $n > 1$ and the property holds up to level $n-1$. By induction hypothesis, V_{n-1} is finite. With this set of labels, one can build a finite number N of different level $n-1$ automata which is exponential in $|V_{n-1}| \cdot K$, where K depends on the number of level $n-1$ automata in $\mathcal{A}_0$ and of their sets of control states. As each transition of a level n automaton is labelled by a product of level $n-1$ automata, then $|V_n|$ is itself exponential in N, and thus doubly exponential in $|V_{n-1}|$. Remark that, as a consequence, the number of steps of the construction is non-elementary in n. $\square$

Lemma 4.5 (Soundness). $\bigcup_{i \geq 0} L(\mathcal{A}_i) \subseteq pre^*_{\mathcal{H}}(S)$.

Proof (sketch). We prove by induction on i the equivalent result that $\forall i,\ L(\mathcal{A}_i) \subseteq pre^*_{\mathcal{H}}(S)$. The base case is trivial since by definition $\mathcal{A}_0 = \mathcal{A}$ and $L(\mathcal{A}) = S \subseteq pre^*_{\mathcal{H}}(S)$. For the inductive step, we consider a store s accepted by a run in $\mathcal{A}_{i+1}$ and reason by induction on the number m of new level k transitions used in this run, where k is the level of the operation d such that $\mathcal{A}_{i+1} = T_d(\mathcal{A}_i)$. The idea is to decompose each run containing m new transitions into a first part with less than m new transitions, one new transition, and a second part also containing less than m new transitions. Then, by induction hypothesis on m and i, one can re-compose a path in $\mathcal{A}_i$ recognizing some store s' such that $s' \in pre^*_{\mathcal{H}}(S)$ and $s \in pre^*_{\mathcal{H}}(s')$. $\square$

Lemma 4.6 (Completeness). $pre^*_{\mathcal{H}}(S) \subseteq \bigcup_{i \geq 0} L(\mathcal{A}_i)$.

Proof (sketch). We prove the sufficient property that for all nested store automaton $\mathcal{A}$ and HCFP transition d, $pre_d(L(\mathcal{A})) \subseteq L(T_d(\mathcal{A}))$. We consider automata $\mathcal{A}$ and $\mathcal{A}'$ such that $\mathcal{A}' = T_d(\mathcal{A})$, and any pair of stores $s \in L(\mathcal{A})$ and $s' \in pre_{d_j}(s)$. It suffices to isolate a run in $\mathcal{A}$ recognizing s and enumerate the possible forms of s' with respect to s and d to be able to exhibit a possible run in $\mathcal{A}'$ accepting s', by definition of T_d. This establishes the fact that T_d adds to the language L of its argument *at least* the set of direct predecessors of stores of L by d. $\square$

As a direct consequence of Proposition 4.2 and Theorem 4.3, we obtain a symbolic model checking algorithm for the logic $\mathsf{E}(\mathsf{F},\mathsf{X})$ with regular store languages as atomic predicates, i.e. the fragment of the temporal logic CTL for the modal operators EF (there exists a path where eventually a property holds) and EX (there exist an immediate successor satisfying a property).

Theorem 4.7. *For every HCFP $\mathcal{H}$ and formula φ of $\mathsf{E}(\mathsf{F},\mathsf{X})$, the set of configurations (stores) satisfying φ is regular and effectively computable.*

5 Constraining Reachability

In this section we address the more general problem of computing a finite automaton recognizing $pre^*_{\mathcal{H}}[C](S)$ for any HCFP H and pair of regular store languages C and S. We provide an extension of the construction of Proposition

4.3 allowing us to ensure that we only consider runs of H whose configurations all belong to C. Again, from a given automaton $\mathcal{A}$, we construct a sequence of automata whose limit recognizes exactly $pre^*_{\mathcal{H}}[C](L(\mathcal{A}))$. The main (and only) difference with the previous case is that we need to compute language intersections at each iteration without invalidating our termination arguments (i.e. without adding any new states to the original automaton). For this reason, we use a class of *alternating* automata, which we call *constrained* nested automata.

Definition 5.1 (Constrained nested automata). *Let $\mathcal{B}$ be a non-nested m-store automaton*[3] *(with $m \geq n$). A level n $\mathcal{B}$-constrained nested automaton $\mathcal{A}$ is a nested automaton $(Q_{\mathcal{A}}, \Gamma, \delta_{\mathcal{A}}, i_{\mathcal{A}}, f_{\mathcal{A}})$ with special transitions of the form $p \xrightarrow[\mathcal{A}]{\mathcal{C}} (q, r)$ where $p, q \in Q_{\mathcal{A}}$, r is a control state of $\mathcal{B}$ and $\mathcal{C}$ is a level $n-1$ $\mathcal{B}$-constrained nested automaton.*

For lack of space, we are not able to provide here the complete semantics of these automata. However, the intuitive idea is quite simple. Suppose $\mathcal{A}$ is a $\mathcal{B}$-constrained nested n-store automaton, and $\mathcal{B}$ also recognizes n-stores. First, we require all the words accepted by $\mathcal{A}$ to be also accepted by $\mathcal{B}$: $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Then, in any run of $\mathcal{A}$ where a transition of the form $p \xrightarrow{\mathcal{D}} (q, r)$ occurs, the remaining part of the input word should be accepted both by $\mathcal{A}$ when resuming from state q and by $\mathcal{B}$ when starting from state r. Of course, when expanding $\mathcal{D}$ into a word of its language, it may require additional checks in $\mathcal{B}$. As a matter of fact, constrained nested automata can be transformed into equivalent level 1 alternating automata. As such, the languages they accept are all regular.

Proposition 5.2. *Constrained nested automata accept regular languages.*

The construction we want to provide needs to refer to whole sets of paths in a level 1 store automaton recognizing the constraint language. To do this, we need to introduce a couple of additional definitions and notations.

Definition 5.3. *Let $\mathcal{A}$ be a finite store automaton over $\Gamma' = \Gamma \cup \{ [\, , \,] \}$. A state p of $\mathcal{A}$ is of level 0 if it has no successor by $[$ and no predecessor by $]$. It is of level k if all its successors by $[$ and predecessors by $]$ are of level $k-1$. The level of p is written $l(p)$.*

We can show that any automaton recognizing only n-stores is equivalent to an automaton whose control states all have a well-defined level. A notion of level can also be defined for paths. A *level n path* in a store automaton is a path $p_1 \dots p_k$ with $l(p_1) = l(p_k) = n$ and $\forall i \in [2, k-1],\ l(p_i) < n$. All such paths are labelled by n-stores. Now, to concisely refer to the whole set of level n paths between two level n control states, we introduce the following notation. Let

$$Q = \{ q \in Q_{\mathcal{A}} \mid l(q) < n \wedge p_1 \xrightarrow[\mathcal{A}]{+} q \xrightarrow[\mathcal{A}]{+} p_2 \}$$

[3] i.e. a standard, level 1 finite state automaton.

be the set of all states of $\mathcal{A}$ occurring on a level n path between p_1 and p_2. If Q is not empty, we write $p_1 \overset{\mathcal{B}}{\underset{\mathcal{A}}{\rightsquigarrow}} p_2$, where $\mathcal{B}$ is defined as:

$$\mathcal{B} = \big(\, Q_{\mathcal{B}} = Q \cup \{p_1, p_2\},\ \Gamma',\ \delta_{\mathcal{B}} = \delta_{\mathcal{A}} \cap (Q_{\mathcal{B}} \times \Gamma' \times Q_{\mathcal{B}}),\ p_1,\ p_2 \,\big).$$

Thanks to these few notions, we can state our result:

Theorem 5.4. *Given a n-HCFP $\mathcal{H}$ and regular sets of n-stores S and C, the set $pre^*_{\mathcal{H}}[C](S)$ is regular and effectively computable.*

To address this problem, we propose a modified version of the construction of the previous section, which uses constrained nested automata. Let $d = (a, o)$ be a HCFP transition rule, $\mathcal{A} = (Q_{\mathcal{A}}, \Gamma, \delta, i, f)$ and $\mathcal{A}' = (Q_{\mathcal{A}}, \Gamma, \delta', i, f)$ two nested k-store automata constrained by a level 1 n-store automaton $\mathcal{B} = (Q_{\mathcal{B}}, \Gamma', \delta_{\mathcal{B}}, i_{\mathcal{B}}, f_{\mathcal{B}})$ accepting C (with $n \geq k$). We define a transformation $T^{\mathcal{B}}_{d_j}(\mathcal{A})$, which is very similar to T_{d_j}, except that we need to add alternating transitions to ensure that no new store is accepted by $\mathcal{A}'$ unless it is the transformation of a store previously accepted by $\mathcal{B}$ (Cf. Figure 2). If $l(d) < k$, we propagate the transformation to the first level $k-1$ automaton along each path:

$$\delta' = \{\, i \xrightarrow{T^{\mathcal{B}}_d(\mathcal{C})} (p, q) \mid i \xrightarrow[\mathcal{A}]{\mathcal{C}} (p, q) \,\} \cup \{\, p \xrightarrow{\mathcal{C}} (p', q') \in \delta \mid p \neq i \,\}.$$

If $l(d) = n$, we distinguish three cases according to the nature of d:

1. If $d = (a, push^w_1)$, then
$$\delta' = \delta \cup \Big\{\, i \xrightarrow{a} (p, q) \mid i \xrightarrow[\mathcal{A}_i]{w} (p, q') \quad \wedge \quad \exists q_1, q \in Q_{\mathcal{B}},$$
$$l(q_1) = l(q) = 0,\ i_{\mathcal{B}} \xrightarrow[\mathcal{B}]{[^n} q_1 \xrightarrow[\mathcal{B}]{w} q \,\Big\}.$$

2. If $d = (a, push_k)$, then for $m = n-k+1$ and $\mathcal{C} = (\mathcal{C}_1 \times \mathcal{C}_2) \times (\mathcal{B}_1 \times \mathcal{B}_2) \times \mathcal{A}^{(k-1)}_a$,
$$\delta' = \delta \cup \Big\{\, i \xrightarrow{\mathcal{C}} (p, q) \mid i \xrightarrow[\mathcal{A}_i]{\mathcal{C}_1} \xrightarrow[\mathcal{A}_i]{\mathcal{C}_2} (p, q') \quad \wedge \quad \exists\, q_1,\, q_2,\, q \in Q_{\mathcal{B}},$$
$$l(q_1) = l(q_2) = l(q) = k-1,\ i_{\mathcal{B}} \xrightarrow[\mathcal{B}]{[^m} q_1 \overset{\mathcal{B}_1}{\underset{\mathcal{B}}{\rightsquigarrow}} q_2 \overset{\mathcal{B}_2}{\underset{\mathcal{B}}{\rightsquigarrow}} q \,\Big\}.$$

3. If $d = (a, pop_k)$, then for $m = n-k+1$,
$$\delta' = \delta \cup \big\{\, i \xrightarrow{\mathcal{A}^{(k-1)}_a} (i, q) \mid \exists q \in Q_{\mathcal{B}},\ l(q) = k-1,\ i_{\mathcal{B}} \xrightarrow[\mathcal{B}_1]{[^m} q \,\big\}.$$

Suppose $H = (\Gamma, \Delta)$ with $\Delta = \{\, d_0, \ldots, d_{l-1} \,\}$. Given an automaton $\mathcal{A}$ such that $S = L(\mathcal{A})$, consider the sequence $(\mathcal{A}_i)_{i \geq 0}$ defined as $\mathcal{A}_0 = \mathcal{A}^{\mathcal{B}}$ (the $\mathcal{B}$-constrained automaton with the same set of states and transitions as $\mathcal{A}$, whose language is $L(\mathcal{A}) \cap L(\mathcal{B})$) and for all $i \geq 0$ and $j = i \mod l$, $\mathcal{A}_{i+1} = T^{\mathcal{B}}_{d_j}(\mathcal{A}_i)$. By definition of $T^{\mathcal{B}}_d$, the number of states in each $\mathcal{A}_i$ does not vary, and since the number of

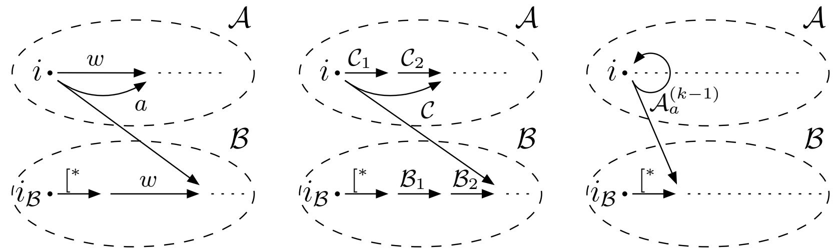

Fig. 2. transformation $T_d^{\mathcal{B}}(\mathcal{A})$ for $d = (a, push_1^w)$, $(a, push_k)$ and (a, pop_k).

control states of $\mathcal{B}$ is finite the same termination arguments as in Lemma 4.4 still hold. It is then quite straightforward to extend the proofs of Lemma 4.5 and Lemma 4.6 to the constrained case.

This more general construction also allows us to extend Theorem 4.7 to the larger fragment $\mathsf{E}(\mathsf{U},\mathsf{X})$ of CTL, where formulas can now contain the modal operator EU (there exists a path along which a first property continuously holds until a second property eventually holds) instead of just EF.

Theorem 5.5. *Given a HCFP $\mathcal{H}$ and formula φ of $\mathsf{E}(\mathsf{U},\mathsf{X})$, the set of configurations (stores) satisfying φ is regular and effectively computable.*

6 Conclusion

We have provided an automata-based symbolic technique for backward reachability analysis of higher-order context-free processes. This technique can be used to check temporal properties expressed in the logic $\mathsf{E}(\mathsf{U},\mathsf{X})$. In this respect, our results provide a first step toward developing symbolic techniques for the model-checking of higher-order context-free or pushdown processes.

Several important questions remain open and are left for future investigation. In particular, it would be interesting to extend our approach to the more general case of higher-order pushdown systems, i.e. by taking into account a set of control states. This does not seem to be technically trivial, and naive extensions of our construction lead to procedures which are not guaranteed to terminate.

Another interesting issue is to generalize our symbolic approach to more general properties than reachability and/or safety, including liveness properties. Finally, it would also be very interesting to extend our symbolic techniques in order to solve games (such as safety and parity games) and to compute representations of the sets of all winning configurations for these games.

References

1. P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *10th CAV*, volume 1427 of *LNCS*, pages 305–318, 1998.

2. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *10th TACAS*, volume 2988 of *LNCS*, pages 467–481, 2004.
3. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of qdds. In *4th SAS*, volume 1302 of *LNCS*, pages 172–186, 1997.
4. A. Bouajjani. Languages, rewriting systems, and verification of infinite-state systems. In *28th ICALP*, volume 2076 of *LNCS*, pages 24–39, 2001.
5. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *8th CONCUR*, volume 1243 of *LNCS*, pages 135–150, 1997.
6. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *12th CAV*, volume 1855 of *LNCS*, pages 403–418, 2000.
7. O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *7th CONCUR*, volume 1119 of *LNCS*, pages 247–262, 1996.
8. T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *29th ICALP*, volume 2380 of *LNCS*, pages 704–715, 2002.
9. T. Cachat. Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In *30th ICALP*, volume 2719 of *LNCS*, pages 556–569, 2003.
10. A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *23rd FSTTCS*, volume 2914 of *LNCS*, pages 112–123, 2003.
11. D. Caucal. On the regular structure of prefix rewriting. *TCS*, 106:61–86, 1992.
12. D. Caucal. On infinite terms having a decidable monadic theory. In *27th MFCS*, volume 2420 of *LNCS*, pages 165–176, 2002.
13. J. Engelfriet. Iterated pushdown automata and complexity classes. In *15th STOC*, pages 365–373, 1983.
14. J. Esparza. Grammars as processes. In *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 232–247, 2002.
15. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithm for model checking pushdown systems. In *12th CAV*, volume 1885 of *LNCS*, pages 232–247, 2000.
16. J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *FoSSaCS*, volume 1578 of *LNCS*, pages 14–30, 1999.
17. J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *1st TACAS*, volume 1019 of *LNCS*, pages 89–110, 1995.
18. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *9th CAV*, volume 1254 of *LNCS*, pages 424–435, 1997.
19. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *5th FoSSaCS*, volume 2303 of *LNCS*, pages 205–222, 2002.
20. D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *TCS*, 37:51–75, 1985.
21. Stefan Schwoon. *Model-Checking Pushdown Systems.* PhD thesis, Technische Universität München, 2002.
22. I. Walukiewicz. Pushdown processes: Games and model checking. In *8th CAV*, volume 1102 of *LNCS*, pages 62–74, 1996.
23. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *10th CAV*, volume 1427 of *LNCS*, pages 88–97, 1998.