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 [26]). Collapsible pushdown systems (CPDSs) are an automaton model of higher-order recursion schemes (HORSs) [10, 21], which allow reasoning about higher-order recursion.

Collapsible pushdown systems are a generalisation of higher-order pushdown systems (HOPDSs). HOPDSs provide a model of HORSs subject to a technical constraint called *safety* [20, 17] 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 [23]. Hence, to model higher-order recursion fully, we require CPDSs, which — using an idea from *panic automata* [18] — 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 HORS [21] (or CPDS [13]). Although, the complexity of such analyses is high — for an order-n CPDS, 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 CPDSs when the number of control states is bounded. Furthermore, when translating from a HORS to a CPDS, it is the arity that determines the number of control states [13]. It has been shown by Kobayashi *et al.* [19] 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 [26] 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 (APDSs), which replace the collapse links of a CPDS with annotations containing the

stack the link pointed to. This model allows a more natural handling than collapse links and we prove that configuration graphs of this model are isomorphic to the configuration graphs of CPDSs when restricted to configurations reachable from a given 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 CPDS. This is a global model-checking algorithm that is based on saturation techniques for higher-order pushdown automata [5, 14, 27]. Our algorithm handles alternating (or "two-player") as well as non-alternating systems.

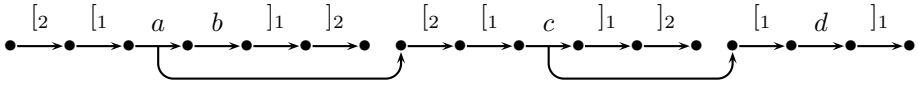
In addition to the extension to annotated pushdown systems, the algorithm improves on Hague and Ong's construction for higher-order pushdown systems [14] since the number of states introduced by the construction is no longer multiplied by the number of iterations it takes to reach a fixed point, potentially 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.

Our full ICALP 2012 submission is available from `http://igm.univ-mlv.fr/~hague/icalp2012/`.

2 Technical Overview

Annotated stacks An order-1 stack is simply a sequence of characters from a given stack alphabet Σ, e.g. $[abc]_1$, where a is the top of the stack. A second-order stack is a stack of order-1 stacks, e.g. $[[abc]_1[de]_1]_2$. This nesting structure continues to build order-n stacks for any order n.

An *annotated* order-n stack is an order-n stack where the stack characters are annotated with order-k stacks (for $1 \leq k \leq n$). E.g., the order-3 stack $[[[a^u b^v]_1]_2]_3$ where the stacks u and v are the annotations on a and b. Often we will omit annotations for readability. Ignoring v annotating b and letting $u = [[[c^{[[d]_1]_2}]_1]_2]_3$, we can represent this stack as the edge-labelled tree shown below.



Annotated stacks are based on collapsible stacks, which follow a similar definition, except the annotations are replaced by pointers to positions further down the stack structure.

One can perform several operations annotated stacks: pop_k removes the uppermost order-$(k-1)$ stack; $push_k$ duplicates the uppermost order-$(k-1)$ stack; $push^k_a$ adds a to the top of the stack, annotated by the uppermost order-k stack with its uppermost order-$(k-1)$ stack removed; $collapse_k$ replaces the top order-k stack with the (order-k) annotation of the uppermost stack character; and rew_a changes the uppermost character to a. Some examples are given below.

$$
\begin{aligned}
pop_3([[[a]_1]_2[[b]_1]_2]_3) &= [[[b]_1]_2]_3 \\
push_2([[[a^u]_1]_2[[b]_1]_2]_3) &= [[[a^u]_1[a^u]_1]_2[[b]_1]_2]_3 \\
push^3_c([[[a]_1]_2[[b]_1]_2]_3) &= [[[c^{[[[b]_1]_2]_3}a]_1]_2[[b]_1]_2]_3 \\
collapse_2\big([[[c^{[[d]_1]_2}a]_1]_2[[b]_1]_2]_3\big) &= [[[d]_1]_2[[b]_1]_2]_3 \\
rew_c([[[a^u]_1]_2[[b]_1]_2]_3) &= [[[c^u]_1]_2[[b]_1]_2]_3
\end{aligned}
$$

Definition 1 (Annotated Pushdown Systems). *An order-n* alternating annotated pushdown system (APDS) *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.*

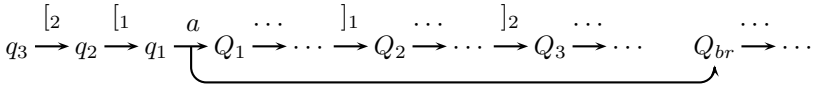
A *configuration* of an APDS is a pair $\langle p, w\rangle$ where $p \in \mathcal{P}$ and w is an order-n annotated stack. We have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ from a rule (p, a, o, p') when a is the uppermost character in w and $w' = o(w)$, and a transition $\langle p, w\rangle \longrightarrow \{\ \langle p', w\rangle \mid p' \in P\ \}$ from a rule $p \to P$. A non-alternating APDS has no rules of this second form. We write C to denote a set of configurations.

Regularity of Annotated Stacks We represent sets of configurations with order-n stack automata based on 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 2 (Order-n Stack Automata). *For a finite stack alphabet Σ, an* order-n stack automaton *is a tuple $A = (\mathcal{Q}_n, \ldots, \mathcal{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1)$ where*

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 such that order-k stacks are recognised from states in $\mathcal{Q}_k$. A transition $(q, q', Q) \in \Delta_k$ for some $k > 1$ can be fired when the uppermost order-$(k-1)$ stack is accepted from $q' \in \mathcal{Q}_{(k-1)}$ and the rest of the stack is accepted from all states in Q. At order-1, a transition (q, a, Q_{br}, Q) is an 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. 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.



Algorithm Given an APDS $\mathcal{C}$ 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 $Pre^*_{\mathcal{C}}(A_0) \supseteq \left\{ \langle p, w \rangle \mid w \in \mathcal{L}_{q_p}(A_0) \right\}$, and

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

recalling C denotes a set of configurations. In our ICALP submission, we show how to 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 reach a fixed point, 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 is n-EXPTIME for alternating APDS and CPDS — which is optimal — and can be improved to $(n-1)$-EXPTIME for non-alternating CPDSs when the stack automaton satisfies a certain notion of *non-alternation*.

3 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 [15]. 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 a 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 [29, 1]. We would like to consider versions of our algorithm for higher-order concurrent systems.

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 APDS is a challenging problem.

We may also study the question of FO decidability over the configuration graphs of APDSs. Note that the undecidability results for CPDSs do not transfer here. In fact we conjecture that FO should be decidable.

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. W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.
11. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV*, pages 232–247, 2000.
12. 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.
13. M. Hague, A. S. Murawski, C.-H. Luke Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LiCS*, pages 452–461, 2008.
14. M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. *Logical Methods in Computer Science*, 4(4), 2008.
15. M. Hague and C.-H. L. Ong. A saturation method for the modal -calculus over pushdown systems. *Inf. Comput.*, 209(5):799–821, 2010.
16. M. Hague and C.-H. L. Ong. Analysing mu-calculus properties of pushdown systems. In *SPIN*, pages 187–192, 2010.
17. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, pages 205–222, 2002.
18. T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, pages 1450–1461, 2005.
19. N. Kobayashi. Higher-order model checking: From theory to practice. In *LiCS*, pages 219–224, 2011.
20. A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 15:1170–1174, 1976.
21. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LiCS*, pages 81–90, 2006.
22. 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.
23. P. Parys. Collapse operation increases expressive power of deterministic higher order pushdown automata. In *STACS*, pages 603–614, 2011.
24. 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.
25. S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP (2)*, pages 162–173, 2011.
26. S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.
27. A. Seth. An alternative construction in symbolic reachability analysis of second order pushdown systems. In *RP* pages 80–95, 2007.
28. A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, pages 203–216, 2009.
29. D. Suwimonteerabuth, J. Esparza, and S. Schwoon. Symbolic context-bounded analysis of multithreaded java programs. In *SPIN*, pages 270–287, 2008.
30. 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.