Saturation of Concurrent Collapsible Pushdown Systems (Extended Abstract)

M. Hague

Royal Holloway University of London

Abstract. Multi-stack pushdown systems are a well-studied model of concurrent computation using threads with first-order procedure calls. While, in general, reachability is undecidable, there are numerous restrictions on stack behaviour that lead to decidability. To model higher-order procedures calls, a generalisation of pushdown stacks called collapsible pushdown stacks are required. Reachability problems for multi-stack collapsible pushdown systems have been little studied. In a recent FSTTCS article [12] we studied ordered, phase-bounded and scope-bounded multi-stack collapsible pushdown systems using saturation techniques and showed decidability of control state reachability, as well as giving a regular representation of all configurations that can reach a given control state.
This is an extended abstract of the FSTTCS 2013 article [12]. A full version of the article with proofs and additional material is also available [13].

1 Introduction

Pushdown systems augment a finite-state machine with a stack and accurately model first-order recursion. Such systems then are ideal for the analysis of sequential first-order programs and several successful tools, such as Moped [25] and SLAM [2], exist for their analysis. However, the domination of multi- and many-core machines means that programmers must be prepared to work in concurrent environments, with several interacting execution threads.

Unfortunately, the analysis of concurrent pushdown systems is well-known to be undecidable. However, most concurrent programs don't interact pathologically and many restrictions on interaction have been discovered that give decidability (e.g. [4,5,26,15,16]).

One particularly successful approach is *context-bounding*. This underapproximates a concurrent system by bounding the number of context switches that may occur [24]. It is based on the observation that most real-world bugs require only a small number of thread interactions [23]. Additionally, a number of more relaxed restrictions on stack behaviour have been introduced. In particular phase-bounded [28], scope-bounded [29], and ordered [6] (corrected in [1]) systems. There are also generic frameworks — that bound the tree- [20] or split-width [9] of the interactions between communication and storage — that give decidability for all communication architectures that can be defined within them.

Languages such as C++, Haskell, Javascript, Python, or Scala increasingly embrace higher-order procedure calls, which present a challenge to verification. A popular approach to modelling higher-order languages for verification is that of higher-order recursion schemes [10,21,17]. Collapsible pushdown systems (CPDS) are an extension of pushdown systems [14] with a "stack-of-stacks" structure. The "collapse" operation allows a CPDS to retrieve information about the context in which a stack character was created. These features give CPDS equivalent modelling power to schemes [14].

These two formalisms have good model-checking properties. E.g, it is decidable whether a μ-calculus formula holds on the execution graph of a scheme [21] (or CPDS [14]). Although, the complexity of such analyses is high, it has been shown by Kobayashi [16] (and Broadbent *et al.* for CPDS [8]) that they can be performed in practice on real code examples.

However concurrency for these models has been little studied. Work by Seth considers phase-bounding for CPDS without collapse [27] by reduction to a finite state parity game. Recent work by Kobayashi and Igarashi studies context-bounded recursion schemes [18].

In a recent FSTTCS article [12], we studied global reachability problems for ordered, phase-bounded, and scope-bounded CPDS. We used *saturation* methods, which have been successfully implemented by e.g. Moped [25] for pushdown systems and C-SHORe [8] for CPDS. Saturation was first applied to model-checking by Bouajjani *et al.* [3] and Finkel *et al.* [11]. We presented a saturation technique for CPDS in ICALP 2012 [7]. In the FSTTCS article [12] and its full version [13], we presented the following advances.

1. Global reachability for ordered CPDSs.
2. Global reachability for phase-bounded CPDSs.
3. Global reachability for scope-bounded CPDSs.

Because the naive encoding of a single second-order stack has an undecidable MSO theory (we show this folklore result in the full paper [13]) it remains a challenging open problem to generalise the generic frameworks above ([20,9]) to CPDSs, since these frameworks rely on MSO decidability over graph representations of the storage and communication structure.

2 Preliminaries

2.1 Concurrent Collapsible Pushdown Systems (CPDS)

For a readable introduction to CPDS we defer to a survey by Ong [22]. Here we will describe Concurrent CPDS only informally, and point the reader to the full article for an exact definition [12]. We use a notion of collapsible stacks called *annotated stacks* (which we refer to as collapsible stacks). These were introduced in ICALP 2012, and are essentially equivalent to the classical model [7].

Higher-Order Collapsible Stacks An order-1 stack is a stack of symbols from a stack alphabet Σ, an order-n stack is a stack of order-$(n-1)$ stacks.

A collapsible stack of order n is an order-n stack in which the stack symbols are annotated with collapsible stacks which may be of any order $\leq n$. Note, in examples we will omit some annotations for clarity.

An example stack is given below

$$[[c^{[[b]_1]_2} a]_1 [b]_1]_2 \ .$$

This is an order-2 stack containing two order-1 stacks (the order of the stack is indicated by the subscripts). The stack is read from left to right, hence the top-most order-1 stack contains the character c on top of the character a. The bottom-most order-1 stack contains only the character b. Only one annotation is shown: the c character is annotated with an order-2 stack containing a single order-1 stack consisting of a single b character.

Stack Operations The following operations can be performed on an order-n stack.

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

We describe each of these operations below.

- The *noop* operation leaves the stack unchanged.
- The pop_1 operation removes the top-most character from the top-most order-1 stack. For example

  $$pop_1\left([[c^{[[b]_1]_2} a]_1 [b]_1]_2\right) = [[a]_1 [b]_1]_2 \ .$$

  That is, we popped the c (and its annotation).
- The rew_a operation rewrites the top-most character of the top-most order-1 stack to a, while leaving its annotation unchanged. For example

  $$rew_a\left([[c^{[[b]_1]_2} a]_1 [b]_1]_2\right) = [[a^{[[b]_1]_2} a]_1 [b]_1]_2 \ .$$

- The $push^k_a$ operation is the most involved. It adds a new character to the top of the top-most stack. In doing so it must also create an annotation. The annotation it creates is the top-most order-k stack with its topmost stack removed. For example

  $$push^2_c([[a]_1 [b]_1]_2) = [[c^{[[b]_1]_2} a]_1 [b]_1]_2 \ .$$

  That is, we put a c character on top of the topmost stack. The annotation of this character is the topmost order-2 stack (in this case there is only one order-2 stack) with its topmost order-1 stack removed.
- The $copy_k$ operation pushes a new stack onto the topmost order-k stack. The stack it pushes is the copy of the top of the stack it is updating. For example

  $$copy_2([[a]_1 [b]_1]_2) = [[a]_1 [a]_1 [b]_1]_2 \ .$$

  That is, the topmost order-1 stack of the topmost order-2 stack has been copied.

– Finally, the pop_k operation removes the topmost stack from the topmost order-k stack. For example

$$pop_2([[a]_1[b]_1]_2) = [[b]_1]_2 \ .$$

Concurrent Collapsible Pushdown Systems We define a general model of concurrent collapsible pushdown systems.

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

A configuration of $\mathcal{C}$ is a tuple $\langle p, w_1, \ldots, w_m \rangle$ where p is a control state and the w_i are stacks. There is a transition

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

via $(p, a, o, p') \in \mathcal{R}_i$ when a is the topmost character of w_i and $w'_i = o(w_i)$.

Consuming and Generating Rules We distinguish two kinds of rule or operation: a rule (p, a, o, p') or operation o is *consuming* if $o = pop_k$ or $o = collapse_k$ for some k. Otherwise, it is *generating.*

2.2 Reachability Problems

We are interested in two problems for a given concurrent CPDS $\mathcal{C}$.

Definition 2 (Control State Reachability Problem). *Given control states p_{in}, p_{out} of $\mathcal{C}$, decide if there is for some $w_1, \ldots, w_m$ a run*

$$\langle p_{in}, \bot_n, \ldots, \bot_n \rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, w_1, \ldots, w_m \rangle \ .$$

Definition 3 (Global Control State Reachability Problem). *Given a control state p_{out} of $\mathcal{C}$, construct a regular representation of the set of configurations $\langle p, w_1, \ldots, w_m \rangle$ such that there exists for some $w'_1, \ldots, w'_m$ a run*

$$\langle p, w_1, \ldots, w_m \rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, w'_1, \ldots, w'_m \rangle \ .$$

We will omit the definition of the representation of stacks. Intuitively, a stack can be considered a word that is the sequence of stack characters read from top to bottom. A stack with annotations can then be considered a tree, where the annotations are where the tree splits into different branches. Regular sets of stacks can then be represented by a suitable kind of finite automata, which we define in the full versions of the article [12,13].

3 Statement of Results

In general, the reachability problems presented above are undecidable for MCPDS. We present informally three different restrictions on the runs of such systems that allow us to obtain positive results. In particular, we have:

Theorem 1 ([12]). *For ordered, phase-bounded and scope-bounded CPDSs the control state reachability problem and the global control state reachability problem are decidable.*

Ordered MCPDS The runs of an ordered MCPDS are restricted such that consuming operations may only be performed on the leftmost non-empty stack. That is, an order-n *ordered CPDS* is an n-MCPDS $\mathcal{C}$ such that a transition from $\langle p, w_1, \ldots, w_m \rangle$ using the rule r on stack i is permitted iff, when r is consuming, for all $1 \leq j < i$ we have $w_j = \perp_n$.

Phase-Bounded MCPDS The runs of a phase-bounded MCPDS are split into a fixed number ζ of phases. During each phase, consuming actions may only occur on a single stack, while generating operations may be performed on any stack. That is, given a fixed number ζ of phases, an order-n *phase-bounded CPDS* is an n-MCPDS with the restriction that each run σ can be partitioned into $\sigma_1 \ldots \sigma_\zeta$ and for all i, if some transition in σ_i by $r \in \mathcal{R}_j$ on stack j for some j is consuming, then all consuming transitions in σ_i are by some $r' \in \mathcal{R}_j$ on stack j.

Scope-Bounded MCPDS The runs of a scope-bounded MCPDS operate according to a round-robin scheduler. Take a fixed scope-bound ζ. Each run consists of an arbitrary number of rounds. When there are m stacks, each round consists of m partitions. In the first partition, only operations acting on the first stack may be performed, in the second, it is the second stack that can be updated, and so on. When a consuming operation is performed, it must be the case that the stack or character removed was created at most ζ rounds earlier. We refer the reader to the full article for a more precise definition [12].

4 Conclusion

In FSTTCS 2013, we have shown decidability of global reachability for ordered, phase-bounded and scope-bounded collapsible pushdown systems [12].

There are several interesting avenues of future work:

1. Our results thus far are purely theoretical. We would like to discover algorithms that may be implemented in practice.
2. Can the generic frameworks for pushdown systems ([20,9]) be generalised to CPDSs?
3. Recently, a more relaxed notion of scope-bounding has been studied [19]. It would be interesting to see if we can extend our results to this notion.

4. Finally, collapsible pushdown systems provide an automaton model for recursion schemes that can be used to model higher-order programs. We would like to study the effect of the restrictions considered here when applied to this modelling process, and if there are different restrictions that may also be applied.

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

References

1. M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *Developments in Language Theory*, pages 121–133, 2008.
2. T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *POPL*, pages 1–3, Portland, Oregon, Jan. 16–18, 2002.
3. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.
4. A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *SIGPLAN Not.*, 38(1):62–73, 2003.
5. A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. *CONCUR 2005 - Concurrency Theory*, pages 473–487, 2005.
6. L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
7. C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *ICALP*, pages 165–176, 2012.
8. C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-shore: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24, 2013.
9. A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, pages 547–561, 2012.
10. W. Damm. The io- and oi-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.
11. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY*, volume 9, pages 27–37, 1997.
12. M. Hague. Saturation of concurrent collapsible pushdown systems. In *FSTTCS*, pages 313–325, 2013.
13. M. Hague. Saturation of concurrent collapsible pushdown systems, 2013. arXiv:1310.2631 [cs.FL].
14. M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.
15. A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *Proc. 13th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'10), Paphos, Cyprus, Mar. 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010.
16. V. Kahlon. Reasoning about threads with bounded lock chains. In *CONCUR*, pages 450–465, 2011.

17. T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, pages 1450–1461, 2005.
18. N. Kobayashi and A. Igarashi. Model-checking higher-order programs with recursive types. In *ESOP*, pages 431–450, 2013.
19. S. La Torre and M. Napoli. A temporal logic for multi-threaded programs. In *IFIP TCS*, pages 225–239, 2012.
20. P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294, 2011.
21. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.
22. L. Ong. Recursion schemes, collapsible pushdown automata and higher-order model checking. In *LATA*, pages 13–41, 2013.
23. S. Qadeer. The case for context-bounded verification of concurrent programs. In *Proceedings of the 15th international workshop on Model Checking Software*, SPIN '08, pages 3–6, Berlin, Heidelberg, 2008. Springer-Verlag.
24. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, pages 93–107, 2005.
25. S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.
26. K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, pages 300–314, 2006.
27. A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, pages 203–216, 2009.
28. S. L. Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170, 2007.
29. S. L. Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, pages 203–218, 2011.