Annotated Stack Trees*

M. Hague[1] and V. Penelle[2]

[1] Royal Holloway, University of London
[2] LIGM, Université Paris-Est

Abstract

Annotated pushdown automata provide an automaton model of higher-order recursion schemes, which may in turn be used to model higher-order programs for the purposes of verification. We study Ground Annotated Stack Tree Rewrite Systems – a tree rewrite system where each node is labelled by the configuration of an annotated pushdown automaton. This allows the modelling of fork and join constructs in higher-order programs and is a generalisation of higher-order stack trees recently introduced by Penelle.

We show that, given a regular set of annotated stack trees, the set of trees that can reach this set is also regular, and constructible in n-EXPTIME for an order-n system, which is optimal. We also show that our construction can be extended to allow a global state through which unrelated nodes of the tree may communicate, provided the number of communications is subject to a fixed bound.

Modern day programming increasingly embraces higher-order programming, both via the inclusion of higher-order constructs in languages such as C++, JavaScript and Python, but also via the importance of *callbacks* in highly popular technologies such as jQuery and Node.js. For example, to read a file in Node.js, one would write

```
fs.readFile('f.txt', function (err, data) { ..use data.. });
```

In this code, the call to `readFile` spawns a new thread that asynchronously reads `f.txt` and sends the `data` to the function argument. This function will have access to, and frequently use, the closure information of the scope in which it appears. The rest of the program runs *in parallel* with this call. This style of programming is fundamental to both jQuery and Node.js programming, as well as being a popular for programs handling input events or slow IO operations such as fetching remote data or querying databases (e.g. HTML5's indexedDB).

Analysing such programs is a challenge for verification tools which usually do not model higher-order recursion, or closures, accurately. However, several higher-order model-checking tools have been recently developed. This trend was pioneered by Kobayashi *et al.* [14] who developed an *intersection type* technique for analysing *higher-order recursion schemes* – a model of higher-order computation. This was implemented in the TRecS tool [13] which demonstrated the feasibility of higher-order model-checking in practice, despite the high theoretical complexities ($(n-1)$-EXPTIME for an order-n recursion scheme). This success has led to the development of several new tools for analysing recursion schemes: GTRecS [15, 17], TravMC [21], C-SHORe [5], HorSat [6], and Preface [25].

In particular, the C-SHORe tool is based on an automata model of recursion schemes called *annotated (or collapsible) pushdown systems* [11]. This is a generalisation of pushdown systems – which accurately model first-order recursion – to the higher-order case. C-SHORe implements a *saturation* algorithm to perform a backwards reachability analysis, which first appeared in ICALP 2012 [4]. Saturation was popularised by Bouajjani *et al.* [1] for the analysis of pushdown systems, which was implemented in the successful Moped tool [28, 30].

*This work was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1].

Contributions In this work we introduce a generalisation of annotated pushdown systems: *ground annotated stack tree rewrite systems (GASTRS)*. A configuration of a GASTRS is an *annotated stack tree* – that is, a tree where each node is labelled by the configuration of an annotated pushdown system. Operations may update the leaf nodes of the tree, either by updating the configuration, creating new leaf nodes, or destroying them. Nodes are created and destroyed using

$$p \xrightarrow{+} (p_1, \ldots, p_m) \text{ and } (p'_1, \ldots, p'_m) \xrightarrow{-} p'$$

which can be seen as spawning m copies of the current process (including closure information) using the first rule, and then later joining these processes with the second rule, returning control to the previous execution (parent node). Alternatively, we can just use $p \xrightarrow{+} (p_1, p_2)$ for a basic fork that does not join.

This model is a generalisation of *higher-order stack trees* recently introduced by Penelle [24], where the tree nodes are labelled by a restriction of annotated pushdown automata called *higher-order pushdown automata*.

As our main contribution, we show that the global backwards reachability problem for GASTRSs can be solved via a saturation technique. That is, given a regular target set of annotated stack trees, we compute a regular representation of all trees from which there is a run of the system to the target set. Note that being able to specify a target set of trees allows us to identify error states such as race conditions between threads. Our result is a generalisation of the ICALP 2012 algorithm, and as such, may be implemented as part of the C-SHORe tool.

Moreover, we define a notion of regularity amenable to saturation which is also closed under the standard boolean operations.

As a final contribution, we show that the model can be extended to allow a bounded amount of communication between separate nodes of the tree. I.e., we add a global state to the system and perform a "context-bounded" analysis [26], where the global state can only be changed an *a priori* fixed number of times.

Related Work Annotated pushdown systems are a generalisation of higher-order pushdown systems that provide a model of recursion schemes subject to a technical constraint called *safety* [20, 12] and are closely related to the Caucal hierarchy [7]. Parys has shown that safety is a genuine constraint on definable traces [23]. Panic automata provided the first model of order-2 schemes, while annotated pushdown systems model schemes of arbitrary order. These formalisms have good model-checking properties. E.g. μ-calculus decidability [22, 11]. Krivine machines can also be used to model recursion schemes [27].

There has been some work studying concurrent variants of recursion scheme model checking, including a context-bounded algorithm for recursion schemes [16], and further underapproximation methods such as phase-bounded, ordered, and scope-bounding [10, 29]. These works allow only a fixed number of threads.

Dynamic thread creation is permitted by both Yasukata *et al.* [31] and by Chadha and Viswanathan [8]. In Yasukata *et al.*'s model, recursion schemes may spawn and join threads. Communication is permitted only via nested locks, whereas in our model we allow shared memory, but only a bounded number of memory updates. Their work is a generalisation of results for order-1 pushdown systems [9]. Chadha and Viswanathan allow threads to be spawned, but only one thread runs at a time, and must run to completion. Moreover, the tree structure is not maintained.

Saturation methods have also been developed for *ground tree rewrite systems* and related systems [18, 3, 19], though these use quite different techniques to the algorithm developed here.

A saturation technique has also been developed for dynamic trees of pushdown processes [2]. These are trees where each process on each node is active (in our model, only the leaf nodes are active). However, their spawn operations do not create a copy of the current process, which would lose closure information in our setting. It would be interesting and non-trivial to study the combination of their approach and ours.

Penelle proves decidability of first order logic with reachabilty over rewriting graphs of ground stack tree rewriting systems [24]. This may be used for a context-bounded reachability result for higher-order stack trees. This result relies on MSO decidability over the configuration graphs of higher-order pushdown automata, through a finite set interpretation of any rewriting graph of a ground stack tree rewriting system into a configuration graph of a higher pushdown automaton. This does not hold for annotated pushdown automata.

References

[1] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.

[2] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, pages 473–487, 2005.

[3] W. S. Brainerd. Tree generating regular systems. *Information and Control*, 14(2):217–231, 1969.

[4] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 165–176, 2012.

[5] 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.

[6] C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, pages 129–148, 2013.

[7] 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.

[8] R. Chadha and M. Viswanathan. Decidability results for well-structured transition systems with auxiliary storage. In *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, pages 136–150, 2007.

[9] T. M. Gawlitza, P. Lammich, M. Müller-Olm, H. Seidl, and A. Wenner. Join-lock-sensitive forward reachability analysis for concurrent programs with dynamic process creation. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, pages 199–213, 2011.

[10] M. Hague. Saturation of concurrent collapsible pushdown systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, pages 313–325, 2013.

[11] M. Hague, A. S. Murawski, C.-H. Luke Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.

[12] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS '02: Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 205–222, London, UK, 2002. Springer-Verlag.

[13] N. Kobayashi. Model-checking higher-order functions. In *PPDP*, pages 25–36, 2009.

[14] N. Kobayashi. Higher-order model checking: From theory to practice. In *LICS*, pages 219–224, 2011.

[15] N. Kobayashi. A practical l inear time algorithm for trivial automata model checking of higher-order recursion schemes. In *FOSSACS*, pages 260–274, 2011.

[16] N. Kobayashi and A. Igarashi. Model-checking higher-order programs with recursive types. In *ESOP*, pages 431–450, 2013.

[17] Naoki Kobayashi. GTRecS2: A model checker for recursion schemes based on games and types. A tool available at `http://www-kb.is.s.u-tokyo.ac.jp/~koba/gtrecs2/`, 2012.

[18] C. Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.

[19] D. Lugiez and P. Schnoebelen. The regular viewpoint on pa-processes. In *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, pages 50–66, 1998.

[20] A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 15:1170–1174, 1976.

[21] R. P. Neatherway, S. J. Ramsay, and C.-H. L. Ong. A traversal-based algorithm for higher-order model checking. In *ICFP*, pages 353–364, 2012.

[22] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

[23] P. Parys. Collapse operation increases expressive power of deterministic higher order pushdown automata. In *STACS*, pages 603–614, 2011.

[24] V. Penelle. Rewriting higher-order stack trees, 2015. arXiv:1311.4915 [cs.FL], to appear in CSR 15.

[25] S. J. Ramsay, R. P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72, 2014.

[26] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, pages 93–107, 2005.

[27] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP (2)*, pages 162–173, 2011.

[28] S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.

[29] A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, pages 203–216, 2009.

[30] D. Suwimonteerabuth, F. Berger, S. Schwoon, and J. Esparza. jmoped: A test environment for java programs. In *CAV*, pages 164–167, 2007.

[31] K. Yasukata, N. Kobayashi, and K. Matsuda. Pairwise reachability analysis for higher order concurrent programs by higher-order model checking. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 312–326, 2014.