2011 26th Annual IEEE Symposium on Logic in Computer Science

Higher-Order Model Checking: From Theory to Practice

Naoki Kobayashi
Tohoku University

***Abstract*—The model checking of higher-order recursion schemes (higher-order model checking for short) has been actively studied in the last decade, and has seen significant progress in both theory and practice. From a practical perspective, higher-order model checking provides a foundation for software model checkers for functional programming languages such as ML and Haskell. This short article aims to provide an overview of the recent progress in higher-order model checking and discuss future directions.**

I. Introduction

A higher-order recursion scheme (a recursion scheme for short) is a kind of tree grammar for generating a single, possibly infinite tree. From a programming language point of view, a recursion scheme is a term (of tree type) of the call-by-name simply-typed λ-calculus with recursion and tree constructors. The goal of the model checking of higher-order recursion schemes (higher-order model checking for short) is, given a recursion scheme $\mathcal{G}$ and a property φ, to decide whether the tree generated by $\mathcal{G}$ satisfies φ. To our knowledge, this model-checking problem has been posed in the present form by Knapik et al. [19, 20], although related problems have been studied since 70's (see Section III). The decidability of higher-order model checking (for arbitrary recursion schemes and properties expressed by modal μ-calculus) has been open for some time, until Ong [36] provided a positive answer in 2006.

Higher-order model checking has attracted interests of the theoretical community in the last decade [1–3, 16, 19–21, 28, 36], as recursion schemes are very expressive grammars for trees, subsuming the previously known classes of trees with MSO-decidable theories, such as regular trees, algebraic trees, and the trees generated by higher-order pushdown automata. Higher-order model checking can also be considered a natural extension of finite-state and pushdown model checking.

Until recently, however, higher-order model checking did not appear to be of interest to researchers on automated program analysis or verification. The main reason is probably that the complexity of higher-order model checking is n-EXPTIME complete (where n is the largest order of functions) [36]. The algorithm given in Ong's decidability proof in fact always suffers from this n-EXPTIME bottleneck, so that it is not runnable even for very small recursion schemes.

The situation above has dramatically changed in the last few years. First, it turned out that various verification problems for higher-order functional programs, such as reachability and resource usage verification [18], can be naturally reduced to higher-order model checking problems [24]. Thus, higher-order model checking provides a universal tool for automated analysis or verification of functional programs. Secondly, a new algorithm for higher-order model checking has been found [22], which does not always suffer from the n-EXPTIME bottleneck. A higher-order model checker TRecS [23] has been implemented based on the new algorithm, and is used as the core of recent verification tools for functional programs [30, 31].

The purpose of this paper is to give a brief overview of higher-order model checking, especially highlighting the recent transition from theory to practice. The paper is by no means a complete survey of the field; it just aims to provide (non-exhaustive) references, from which the reader can find more information. Nevertheless, we hope that this short survey helps more people to get acquainted with and interested in this exciting, but relatively unexplored research field.

II. Higher-Order Recursion Schemes and Model Checking

This section provides informal definitions of higher-order recursion schemes and model checking problems. For more formal definitions, the reader is referred to [36].

We consider *simple types* (also called *sorts*) constructed from a unique base type $\circ$, describing trees, and the standard function type constructor $\rightarrow$. We assume a finite set of tree constructors $c_1, \ldots, c_k$ of type $\circ \rightarrow \cdots \rightarrow \circ \rightarrow \circ$. We write $\widetilde{x}$ for a sequence $x_1, \ldots, x_m$.

A *higher-order recursion scheme* (a recursion scheme for short) $\mathcal{G}$ is a pair (D, t) where D is a set of top-level function definitions, of the form $\{F_1\,\widetilde{x}_1 = t_1, \ldots, F_\ell\,\widetilde{x}_\ell = t_\ell\}$, and t is a main program, whose syntax is given by:

$$t ::= x \mid F_i \mid c_j \mid t_1 t_2.$$

We consider the standard simple type system, and require that the term t and the body t_i of each function definition have the tree type $\circ$.

The *order* of a recursion scheme is the highest order of the simple types of function symbols, where the order of a simple type κ is given by:

$$order(\circ) = 0$$
$$order(\kappa_1 \rightarrow \kappa_2) = max(order(\kappa_1) + 1, order(\kappa_2))$$

1043-6871/11 $26.00 © 2011 IEEE
DOI 10.1109/LICS.2011.15

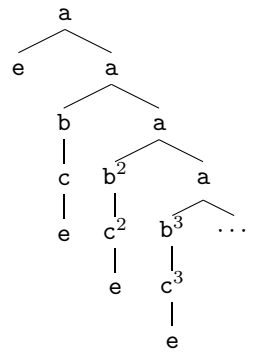

Figure 1. The tree generated by $\mathcal{G}_0$

The *tree generated by* a recursion scheme $\mathcal{G} = (D, t)$, written $\mathbf{Tree}(\mathcal{G})$, is the (possibly infinite) tree obtained by reducing t according to D (possibly) infinitely often in a fair manner.

Example 2.1: Consider tree constructors $\mathtt{a} : \circ \to \circ \to \circ$, $\mathtt{b} : \circ \to \circ$, $\mathtt{c} : \circ \to \circ$, and $\mathtt{e} : \circ$. Let $\mathcal{G}_0$ be (D, S), where D is the set of function definitions:

$$\begin{array}{rcl} S & = & F\ I\ \mathtt{e} \\ F\ f\ x & = & \mathtt{a}\ (f\ x)\ (F\ (B\ f)\ (\mathtt{c}\ x)) \\ B\ f\ x & = & \mathtt{b}\ (f\ x) \\ I\ x & = & x \end{array}$$

$\mathcal{G}_0$ is an order-2 recursion scheme, where F and B have an order-2 type $(\circ \to \circ) \to \circ \to \circ$. S is reduced as follows:

$$\begin{array}{l} S \longrightarrow F\ I\ \mathtt{e} \longrightarrow \mathtt{a}\ (I\ \mathtt{e})\ (F\ (B\ I)\ (\mathtt{c}\ \mathtt{e})) \\ \longrightarrow \mathtt{a}\ \mathtt{e}\ (\mathtt{a}\ (B\ I\ (\mathtt{c}\ \mathtt{e}))\ (F\ (B\ (B\ I))\ (\mathtt{c}\ (\mathtt{c}\ \mathtt{e})))) \longrightarrow \cdots \end{array}$$

The tree generated by $\mathcal{G}_0$ is shown in Figure 1, which consists of paths of the form $\mathtt{a}^{n+1}\mathtt{b}^n\mathtt{c}^n\mathtt{e}$.

The following is the key result, due to Ong.

Theorem 2.1 ([36]): Given an order-n higher-order recursion scheme $\mathcal{G}$ and a modal μ-calculus formula φ, the problem of deciding whether $\mathbf{Tree}(\mathcal{G})$ satisfies φ is n-EXPTIME in the size of $\mathcal{G}$ and φ.

Remark 2.1: Restricted forms of higher-order model checking problems are often considered in the literature. Knapik et al. [20] considered a restricted class of recursion schemes called *safe* higher-order recursion schemes and proved the decidability for the restricted class. Aehlig [1] considered a restricted class of properties, expressed by *trivial automata*, which corresponds to a fragment of the modal μ-calculus with only (positive occurrences of) the greatest fixedpoint operator ν. Trivial automata are sufficient [24] for applications to verification of safety properties of functional programs.

III. Historical Background

This section briefly reviews some historical background about the model checking of higher-order recursion schemes. De Miranda's PhD thesis [13] and Ong's tutorial paper [37] provide a more comprehensive survey.

Order-1 recursion schemes have been studied in 70's under the name of *recursive program schemes* [10, 35]. A recursive program scheme provides a tree representation of the control flow graph of a program, with the semantics of data and primitive instructions left unspecified. Thus, the semantics of a program can be represented as a pair of a recursive program scheme and an interpretation function for the instructions. The reader is referred to other books or papers [9, 14] on the historical background of program schemes.

Higher-order grammars, where non-terminals can take functions as parameters, or higher type program schemes have been introduced also in 70's [12, 42, 45] and extensively studied since then [11]. As already mentioned, Knapik et al. [19, 20] formalized the model-checking problem for higher-order recursion schemes in the present form, and proved the decidability for the class of *safe* higher-order recursion schemes. To our knowledge, most of the earlier results for higher-order languages [11] are also under the equivalent restriction of *derived types*. The reader is referred to [13] for discussions on the safety condition and derived types. The first result on the model checking of unrestricted recursion schemes was provided independently by Knapik et al. [21] and Aehlig et al. [2], who proved that the model checking of order-2 recursion schemes is decidable. Ong [36] then extended the result to arbitrary orders.

IV. State of the Art

This section summarizes the state-of-the-art in the field of higher-order model checking. The results are classified into theory and practice, though they are of course related to each other.

A. Theory

1) Decidability: As already mentioned, the most general decidability result is due to Ong [36]. At present, there are at least three proofs of the decidability: Ong's original proof, Hague et al.'s reduction to model checking problems on collapsible higher-order pushdown automata [16], and Kobayashi and Ong's proof based on intersection types [28]. The first two are based on game semantics, and the third one, which is relatively more elementary, is based on type theory. Simpler proofs are available [1, 24] for the restricted class of properties expressed by trivial automata.

Broadbent et al. [6] gave an algorithm for *global* model checking, which, given a recursion scheme $\mathcal{G}$ and a formula φ, computes all the subtrees of $\mathbf{Tree}(\mathcal{G})$ that satisfy φ. As there may be infinitely many such trees, the output is another recursion scheme $\mathcal{G}'$ such that $\mathbf{Tree}(\mathcal{G}')$ is identical to $\mathbf{Tree}(\mathcal{G})$ except that all the (roots of) subtrees satisfying φ are marked.

2) Expressive Power: As already mentioned, the restricted class of *safe* recursion schemes was considered in earlier studies. It has been an open question whether the safety is a genuine restriction until recently. Parys [39] answered the question by proving that there is a tree that is generated by an order-2 unsafe recursion scheme but not by any order-2 safe recursion scheme. For *word* languages generated by (non-deterministic) order-2 recursion schemes, however, safety is not a fundamental restriction [3].

Connections to other models of higher-order computation have also been actively studied. The class of trees generated by safe higher-order recursion schemes coincides with those generated by higher-order pushdown automata [20] and also with the class of trees in Caucal hierarchy [7]. Recursion schemes without the safety condition are equivalent to higher-order pushdown automata extended with collapse operations [16].

3) Complexity: As already mentioned, the complexity of the modal μ-calculus model checking of higher-order recursion schemes is n-EXPTIME complete [36]. It is the case even for *safe* higher-order recursion schemes. If the largest size of types of function symbols and the size of the formula are fixed, however, the time complexity is polynomial in the size of recursion schemes [28]. Under the same condition, for the class of trivial automata, the time complexity is linear in the size of recursion schemes [24]. For other results on the complexity of higher-order model checking, we refer the reader to [17, 27].

B. Practice

1) Model-Checking Algorithms and Implementation: All the decidability proofs mentioned in Section IV-A1 are constructive in the sense that they provide model checking algorithms. Unfortunately, however, none of them can be used in practice because they always suffer from n-EXPTIME bottleneck. To our knowledge, the first practical algorithm and its implementation are due to Kobayashi [22, 23], where the class of properties is restricted to those expressed by deterministic trivial automata. Lester et al. [32] recently extended Kobayashi's algorithm to deal with properties expressed by alternating weak tree automata, and implemented another model checker THORS. These algorithms are based on a reduction from model checking to intersection type inference problems [24, 28], and guess intersection types by partially constructing the tree generated by a recursion scheme, and running automata over the partial tree. Although the worst-case time complexity of the algorithms is not polynomial in the size of recursion schemes (recall Section IV-A3), the algorithms run fast for many realistic inputs according to experiments.

Kobayashi [26] has recently proposed yet another practical algorithm for trivial automata model checking, which runs in time linear in the size of recursion schemes, under the assumption that the largest size of types of function symbols and the size of the formula are fixed.

2) Applications to Program Verification: A promising application of higher-order model checking is automated verification of functional programs. Given that higher-order model checking subsumes finite state model checking and pushdown model checking, which have been successfully applied to software model checkers for imperative languages [4, 5], it is natural to expect that higher-order model checking can be applied to construct software model checkers for functional languages like ML and Haskell. As mentioned below, initial results towards such directions have been obtained.

For the simply-typed λ-calculus with recursion and finite base types, Kobayashi [22, 24, 25] showed that a variety of program analysis or verification problems, such as reachability, flow analysis, and resource usage verification can be reduced to higher-order model checking. Thus, a higher-order model checker can be used as a universal tool for solving those problems.

Recent studies aim to deal with a larger class of functional programs, with a sacrifice of completeness. Kobayashi et al. [31] introduced a restricted class of higher-order tree-processing programs called higher-order multi-parameter tree transducers, and proposed a method to verify that tree-processing programs conform to input and output specifications. They [43] later extended the method to handle arbitrary tree-processing functional programs, by requiring user annotations on certain invariants. Ramsay and Ong [38] combined a conventional static analysis and higher-order model checking to deal with programs manipulating recursive data types. Kobayashi et al. [29, 30] combined predicate abstraction and CEGAR with higher-order model checking, and constructed a software model checker MOCHI for a tiny subset of ML. MOCHI takes functional programs annotated with assertions on integer values, and verifies the lack of assertion failures fully automatically.

V. FUTURE DIRECTIONS

Despite the long history of higher-order recursion schemes and the recent progress in higher-order model checking, a lot of work is still left to be done in this research area. We discuss below some future directions, especially for applications to program verification.

A. Theory

1) Open problems: There remain some open problems about higher-order recursion schemes. The most notable one is whether the equivalence of the trees generated by two recursion schemes is decidable. This may have an application to automated verification of program equivalence. Another open problem is whether the results on the safety condition [3, 39] for order-2 recursion schemes extend to arbitrary orders.

2) Extensions of higher-order model checking: Another challenge is to find a useful extension of the decidability of higher-order model checking [36]. There are two possible directions: extending the class of models, or the class of properties. For example, in view of applications to program verification, it would be very useful to relax the restriction that terms of recursion schemes must be simply-typed. As discussed in [41], however, extensions of the type system can easily make the model checking problem undecidable.

3) Theoretical analysis of model checking algorithms: We need a better understanding of the nature of the high complexity of higher-order model checking. Ong [36] has shown that higher-order model checking is n-EXPTIME complete, but recent model checking algorithms [22, 26] run reasonably fast for many inputs. This cannot be explained by the complexity of fixed-parameter PTIME (recall Section IV-A3) alone, as the constant factor is too large in the worst case. As discussed in [25], the high complexity of higher-order model checking seems to be related to the ability of higher-order functions to express a long computation compactly. For example, consider the following two recursion schemes $\mathcal{G}_0$ and $\mathcal{G}_1$:

$$\mathcal{G}_0 = (\{F_{2^m} = \mathtt{a}\ F_{2^m-1}, \ldots, F_1 = \mathtt{a}\ F_0, F_0 = \mathtt{c}\}, F_{2^m})$$
$$\mathcal{G}_1 = (\{G_m\ x = G_{m-1}(G_{m-1}\ x), \ldots, G_1\ x = G_0(G_0\ x), G_0\ x = \mathtt{a}\ x\}, G_m\ \mathtt{c})$$

$\mathcal{G}_0$ and $\mathcal{G}_1$ generate the same tree $\mathtt{a}^{2^m}\mathtt{c}$, but the size of $\mathcal{G}_0$ is exponential in that of $\mathcal{G}_1$. Thus, an exponential time algorithm for $\mathcal{G}_1$ can be as fast as a polynomial time algorithm for $\mathcal{G}_0$. Similarly, there are finite trees that can be expressed by order-n recursion schemes of size $O(m)$ but can only be expressed by order-0 recursion schemes of size $O(2^{\overbrace{2^{\cdots^{2^m}}}^{n}})$. If this is the main source of the n-EXPTIME hardness, higher-order model checking may work for typical programs that do not fully exploit the expressive power of higher-order functions. Furthermore, the recent fixed-parameter linear time algorithm [26] runs in time linear in the size of recursion schemes of arbitrary order, so that the model checking of well-structured higher-order programs can be faster than that of equivalent first-order programs. More theoretical justifications are necessary however.

B. Practice

1) Better higher-order model checkers: Better algorithms and implementation techniques for higher-order model checking are required. According to experiments, the state-of-the-art higher-order model checker TRecS [23] can check many recursion schemes consisting of a few hundred function definitions in a few seconds, but that is not sufficient in the context of applications to program verification [30, 31], as higher-order recursion schemes obtained from source programs are typically larger than the source programs. As for algorithms, the new fixed-parameter linear time algorithm [26] is attractive but requires further investigation, as the current implementation is actually much slower than TRecS for many inputs. As for implementation techniques, BDD-like implementation techniques would be important, especially because boolean values are heavily used in recursion schemes obtained from program verification problems [30].

The current higher-order model checkers support only a very restricted fragment of the properties expressed by the modal μ-calculus. Implementing an efficient full modal μ-calculus model checker is a challenge.

2) Scalable program verification tools: Program verification tools [30, 31] based on higher-order model checking should be improved in terms of both efficiency and supported language features. For the efficiency, besides the improvement of higher-order model checkers as discussed above, abstraction techniques should also be improved. As fully automated verification of large programs is difficult, finding a good combination with user annotations [43] would also be important. As for the language features, techniques for supporting integers [30] and algebraic data structures [31, 38] have been proposed, but they should be improved and integrated. Supporting objects and concurrency also remains a big challenge.

3) Applications to other program verification problems: Although a variety of program verification or analysis problems can be reduced to higher-order model checking [22, 24], many of them have not been implemented yet, so that the effectiveness and the scalability of the resulting methods are not clear. Particularly interesting for further investigation is the reduction from control flow analysis (CFA) to higher-order model checking. Flow analyses for functional languages such as k-CFA [40] compute over-approximations of the actual flow, except Mossin's exact flow analysis [34], which has non-elementary time complexity and has never been implemented. Interestingly, with the reduction to higher-order model checking, a single flow query (of whether the expression at program point ℓ evaluates to a value created at ℓ') can be answered in time linear in the program size, under the assumption that the largest size of the simple types of functions is fixed. Under the same assumption, therefore, all the flow queries can be computed in cubic time. Thus, if the constant factor is ignored, the time complexity is similar to that of 0-CFA [33], unlike other flow analyses more precise than 0-CFA, like k-CFA [40] and CFA2 [44]. It would be interesting to know how much higher-order model checking-based CFA scales; note that it is sometimes observed that the precision of program analysis also improves the efficiency.

Higher-order model checking [24, 30, 31, 38] has so far been applied to verification of safety properties. It should also be possible to apply higher-order model checking to verification of liveness properties such as termination, following the success of first-order model checking [8].

4) Other applications: There are many other potential applications of higher-order model checking. Given that higher-order functions are used in recent hardware description languages [15], an application of higher-order model checking to hardware verification may be useful. Another interesting application is data compression. As mentioned in Section V-A3, higher-order recursion schemes can express certain trees extremely compactly. Higher-order model checking can then be used to check whether such compressed trees match a given pattern without decompressing them. The time complexity of the pattern matching is linear in the size of the compressed data (expressed in the form of recursion schemes), if the size of the pattern is fixed.

VI. Conclusion

We have briefly reviewed the recent progress of higher-order model checking, and discussed future directions. Although its scalability is unclear at the moment, we believe that higher-order model checking is an exciting research topic for both theoretical and practical research communities and hope that more researchers get interested and involved in this topic.

Acknowledgment: We would like to thank Luke Ong for comments on this paper.

References

[1] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

[2] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *TLCA 2005*, volume 3461 of *Lecture Notes in Computer Science*, pages 39–54. Springer-Verlag, 2005.

[3] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 490–504. Springer-Verlag, 2005.

[4] T. Ball and S. K. Rajamani. The SLAM project: debugging system software via static analysis. In *POPL*, pages 1–3, 2002.

[5] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker Blast. *International Journal on Software Tools for Technology Transfer*, 9(5-6):505–525, 2007.

[6] C. H. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre. Recursion schemes and logical reflection. In *Proceedings of LICS 2010*, pages 120–129. IEEE Computer Society Press, 2010.

[7] D. Caucal. On infinite terms having a decidable monadic theory. In *Proceedings of MFCS 2002*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer-Verlag, 2002.

[8] B. Cook, A. Gotsman, A. Podelski, A. Rybalchenko, and M. Y. Vardi. Proving that programs eventually do something good. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 265–276. ACM Press, 2007.

[9] B. Coucelle. *Handbook of Theoretical Computer Science, Volume B*, chapter Recursive Applicative Program Schemes, pages 459–492. The MIT Press, 1990.

[10] B. Courcelle and M. Nivat. The algebraic semantics of recursive program schemes. In *Proceedings of MFCS 1978*, volume 64 of *Lecture Notes in Computer Science*, pages 16–30, 1978.

[11] W. Dam. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

[12] W. Damm. Higher type program schemes and their tree languages. In *Theoretical Computer Science, 3rd GI-Conference*, volume 48 of *Lecture Notes in Computer Science*, pages 51–72, 1977.

[13] J. de Miranda. *Structures generated by Higher-Order Grammars and the Safety Constraint*. PhD thesis, University of Oxford, 2006.

[14] J. Engelfriet. *Simple Program Schemes and Formal Languages*, volume 20 of *Lecture Notes in Computer Science*. Springer-Verlag, 1974.

[15] D. R. Ghica and A. Smith. Verifying higher-order programs with pattern-matching algebraic data types. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 587–598, 2011.

[16] M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE Computer Society, 2008.

[17] M. Hague and A. W. To. The complexity of model checking (collapsible) higher-order pushdown systems. In *Proceedings of FSTTCS 2010*, pages 228–239, 2010.

[18] A. Igarashi and N. Kobayashi. Resource usage analysis. *ACM Transactions on Programming Languages and Systems*, 27(2):264–313, 2005.

[19] T. Knapik, D. Niwinski, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA 2001*, volume 2044 of *Lecture Notes in Computer Science*, pages 253–267. Springer-Verlag, 2001.

[20] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer-Verlag, 2002.

[21] T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 1450–1461. Springer-Verlag, 2005.

[22] N. Kobayashi. Model-checking higher-order functions. In *Proceedings of PPDP 2009*, pages 25–36. ACM Press, 2009. See also [25].

[23] N. Kobayashi. TRecS: A type-based model checker for recursion schemes. http://www.kb.ecei.tohoku.ac.jp/~koba/trecs/, 2009.

[24] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 416–428, 2009. See also [25].

[25] N. Kobayashi. Model checking higher-order programs. Available at http://www.kb.ecei.tohoku.ac.jp/\~{}koba/papers/hmc.pdf. A revised and extended version of [24] and [22], 2010.

[26] N. Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *Proceedings of FoSSaCS 2011*, volume 6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer-Verlag, 2011.

[27] N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In *Proceedings of ICALP 2009*, volume 5556 of *Lecture Notes in Computer Science*, pages 223–234. Springer-Verlag, 2009.

[28] N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.

[29] N. Kobayashi, R. Sato, and H. Unno. MoCHi: A model checker for higher-order programs. http://www.kb.ecei.tohoku.ac.jp/~ryosuke/cegar/, 2011.

[30] N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and cegar for higher-order model checking. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2011.

[31] N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 495–508, 2010.

[32] M. M. Lester, R. P. Neatherway, C.-H. L. Ong, and S. J. Ramsay. Model checking liveness properties of higher-order functional programs. Unpublished manuscript, 2010.

[33] J. Midtgaard and D. V. Horn. Subcubic control flow analysis algorithms. *Higher-Order and Symbolic Computation*.

[34] C. Mossin. Exact flow analysis. *Mathematical Structures in Computer Science*, 13(1):125–156, 2003.

[35] M. Nivat. On the interpretation of recursive program schemes. In *Symposia Mathematica*, pages 255–281, 1975.

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

[37] C.-H. L. Ong. *Logics and Languages for Reliability and Security*, chapter Models of Higher-Order Computation: Recursive Schemes and Collapsible Pushdown Automata, pages 263–299. IOS Press, 2010.

[38] C.-H. L. Ong and S. Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 587–598, 2011.

[39] P. Parys. Collapse operation increases expressive power of deterministic higher order pushdown automata. In *Proceedings of STACS 2011*, 2011.

[40] O. Shivers. *Control-Flow Analysis of Higher-Order Languages*. PhD thesis, Carnegie-Mellon University, May 1991.

[41] T. Tsukada and N. Kobayashi. Untyped recursion schemes and infinite intersection types. In *Proceedings of FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 343–357. Springer-Verlag, 2010.

[42] R. Turner. An infinite hierarchy of term languages - an approach to mathematical complexity. In *Proceedings of ICALP*, pages 593–608, 1972.

[43] H. Unno, N. Tabuchi, and N. Kobayashi. Verification of tree-processing programs via higher-order model checking. In *Proceedings of APLAS 2010*, volume 6461 of *Lecture Notes in Computer Science*, pages 312–327. Springer-Verlag, 2010.

[44] D. Vardoulakis and O. Shivers. Cfa2: A context-free approach to control-flow analysis. In *Proceedings of ESOP 2010*, volume 6012 of *Lecture Notes in Computer Science*, pages 570–589. Springer-Verlag, 2010.

[45] M. Wand. An algebraic formulation of the chomsky hierarchy. In *Category Theory Applied to Computation and Control*, volume 25 of *Lecture Notes in Computer Science*, pages 209–213. Springer-Verlag, 1974.