Verification of Higher-Order Computation: A Game-Semantic Approach

C.-H. L. Ong

Oxford University Computing Laboratory
users.comlab.ox.ac.uk/luke.ong/

Abstract. We survey recent developments in an approach to the verification of higher-order computation based on game semantics. Higher-order recursion schemes are in essence (programs of) the simply-typed lambda calculus with recursion, generated from uninterpreted first-order symbols. They are a highly expressive definitional device for infinite structures such as word languages and infinite ranked trees. As applications of a representation theory of innocent strategies based on *traversals*, we present a recent advance in the model checking of trees generated by recursion schemes, and the first machine characterization of recursion schemes (by a new variant class of higher-order pushdown automata called *collapsible pushdown automata*). We conclude with some speculative remarks about reachability checking of functional programs. A theme of the work is the fruitful interplay of ideas between the neighbouring fields of semantics and verification.

Game semantics has emerged as a powerful paradigm for giving semantics to a variety of programming languages and logical systems. It has been used to construct the first syntax-independent fully abstract models for a spectrum of programming languages ranging from purely functional languages to languages with non-functional features such as control operators and locally-scoped references [3, 26, 4, 5, 25, 2, 30] etc. In this extended abstract, we present in brief recent developments in *algorithmic game semantics*, which is concerned with applying game semantics to computer-assisted verification and program analysis [22, 19, 36, 33, 34].

Game semantics has several features which make it very promising for such applications. It provides a very *concrete* way of building *fully abstract* models. It has a clear operational content, which admits *compositional methods* in the style of denotational semantics. The basic objects studied in game semantics are games (between two players, called P and O), and strategies on games. As strategies can be seen as certain kinds of highly-constrained processes, they admit the same kind of automata-theoretic representations central to model checking and allied methods in computer-assisted verification [43, 14]. Moreover games and strategies naturally form themselves into intricate mathematical structures that give very accurate models of advanced high-level programming languages, as the various full abstraction results show. For an introduction to game semantics, see for example [6].

Traversal: a representation theory of innocent strategies

In game semantics, programs are modelled as P-strategies. Strategies, which are certain sets of *plays* (or *legal positions*), are typically composed by *parallel composition plus hiding*, in the sense of the process algebra CSP [24]. The starting point of our work is a *representation theory* of the game semantics of higher-type programs (such as recursion schemes, PCF and Idealized Algol) that is very concrete, involving combinatorics over infinite structures defined by the abstract syntax trees of the programs being modelled. Take a program M which may be open. In this approach the strategy-denotation of M, written $[\![M]\!]$, is represented by a set $\mathcal{T}r(M)$ of *traversals* over a possibly infinite tree – called the *computation tree* of M – which is generated from (a souped up version of) the abstract syntax tree of M. (Formally a traversal over a tree is a sequence of nodes starting from the root; quite unlike a path in the tree, a traversal can "jump" all over the tree, and may visit certain nodes infinitely often.) A traversal over the computation tree of M does not correspond to a play in $[\![M]\!]$, but rather to an *interaction sequence* that is obtained by *uncovering* [26] a play in $[\![M]\!]$ in a hereditary fashion; and a suitable projection of $\mathcal{T}r(M)$ – corresponding to the operation of hiding – gives the strategy-denotation $[\![M]\!]$. We call such a result a *Path-Traversal Correspondence Theorem*. (Denoting programs by sets of interaction sequences obtained by hereditary uncovering was first considered by Greenland in his DPhil thesis [20], which he has called *revealed semantics*.) The set $\mathcal{T}r(M)$ is defined by recursion over the syntax of M and by rule induction. Intuitively these formation rules define what amounts to the composition algorithm of innocent strategies (less the hiding) but expressed in a setting in which moves (of the innocent game) are mapped to nodes of the computation tree. In [12] (see also Blum's forthcoming DPhil thesis [10]) we give a self-contained account of the traversal-based representation theory and establish Path-Traversal Correspondence Theorems for a number of higher-order languages including recursion schemes and PCF.

In the following we consider (higher-order) recursion schemes as a definitional device for infinite structures (mainly ranked trees, but also word languages and directed graphs). We sketch two applications of a Path-Correspondence Theorem for recursion schemes: the first concerns the verification of (possibly infinite) ranked trees generated by recursion schemes, and the second is a machine characterization of recursion schemes.

Recursion schemes of order 1, originally known as *recursive program schemes*, were first formalized and studied in the early 70's [17, 35] (although the basic ideas of program schemes and fixpoint theory go further back to David Park in the late 60's); they were an influential formalism for the semantical analysis of both imperative and functional programs [35, 15]. We fix a (ranked) alphabet Σ. *Types* are generated from a base type o using the arrow constructor $\rightarrow$. A (higher-order) *recursion scheme* is a finite set of equations of the form $F\, x_1 \cdots x_n = e$, where $F : A_1 \rightarrow \cdots \rightarrow A_n \rightarrow o$ is a typed non-terminal, each $x_i : A_i$ is a typed variable, and e is an applicative term of type o constructed from the non-terminals (which include a distinguished *start symbol*), terminals (which

are the Σ-symbols), and variables $x_1, \cdots, x_n$. The scheme is said to be *order-k* if the highest order of the non-terminals is k. We use (deterministic) recursion schemes here as generators of possibly infinite term-trees. The *tree* generated by a recursion scheme is defined to be the (possibly infinite) term-tree built up from the first-order terminal symbols by applying the (equations *qua*) rewrite rules *ad infinitum*, replacing the formal parameters by the actual parameters, starting from the start symbol. Note that in essence, recursion schemes are programs of the *simply-typed lambda calculus with recursion* (generated from uninterpreted 1st-order symbols).

Model-checking trees generated by recursion schemes

In a FOSSACS'02 paper [28], Knapik, Niwiński and Urzyczyn studied the infinite hierarchy of term-trees generated by higher-order recursion schemes that are *homogeneously typed* and satisfy a syntactic constraint called *safety*[1]. They showed that for every $n \geq 0$, the trees that are generated by order-n safe schemes have decidable monadic second-order (MSO) theories. Later in the year at MFCS'02 [13], Caucal introduced a tree hierarchy and a graph hierarchy that are defined by mutual recursion, using a pair of powerful transformations that preserve decidability of MSO theories. Caucal's tree hierarchy coincides with the hierarchy of trees generated by higher-order safe recursion schemes. In [28] Knapik *et al.* asked if the safety assumption is really necessary for their MSO decidability result. A partial answer was subsequently obtained by Aehlig, de Miranda and Ong; in a TLCA'05 paper [7], they showed that trees that are generated by order-2 recursion schemes, whether safe or not, have decidable MSO theories. Independently, Knapik, Niwiński, Urzyczyn and Walukiewicz obtained a sharper result: in an ICALP'05 paper [29], they proved that the modal mu-calculus model-checking problem for trees generated by order-2 recursion schemes (whether safe or not) is 2-EXPTIME complete. A year later in a LICS'06 paper [37], we gave a complete answer to the question:

Theorem 1 (Decidability). *The modal mu-calculus model-checking problem for trees generated by order-n recursion schemes (whether safe or not, and whether homogeneously typed or not) is* n-EXPTIME *complete, for every* $n \geq 0$. *Thus these trees have decidable MSO theories.*

Our approach to the decidability result is to transfer the algorithmic analysis from the tree generated by a recursion scheme, which we call *value tree*, to the *computation tree*, which is itself a tree generated by a related order-0 recursion scheme (equivalently, a regular tree). The computation tree recovers useful intensional information about the computational process behind the construction of the value tree. Paths in the value tree correspond exactly to plays in the game semantics of the recursion scheme; a traversal is then (a representation of) the

[1] The *safety condition* may be presented as a set of rules that determine where a variable may occur as a subterm of a term, depending on both the order of the variable and the order of the term (see [11, 10]).

uncovering of such a play. By appealing to the Path-Traversal Correspondence Theorem, we prove that a given alternating parity tree automaton (APT) [18] has an accepting run-tree over the value tree if and only if it has an accepting *traversal-tree* over the computation tree. Our problem is then reduced to finding an effective way of recognizing a set of infinite traversals (over a given computation tree) that satisfy the parity condition. This requires a new idea as a traversal is most unlike a path. Our solution again exploits the game-semantic connection. It is a property of traversals that their *P-views* are paths (in the computation tree). This allows us to simulate a traversal over a computation tree by (the P-views of its prefixes, which are) annotated paths of a certain kind in the same tree. The simulation is made precise in the notion of *traversal-simulating* APT. We establish the correctness of the simulation by proving that a given *property*[2] APT has an accepting traversal-tree over the computation tree if and only if the associated *traversal-simulating* APT has an accepting run-tree over the computation tree. Note that the decidability of the modal mu-calculus model-checking problem for trees generated by recursion schemes follows at once since computation trees are regular, and the APT acceptance problem for regular trees is decidable [40, 18].

A machine characterization of higher-order recursion schemes

Another application of the Path-Traversal Correspondence Theorem concerns a fundamental question about higher-order recursion schemes: *Can we characterize their expressivity by a class of machine models?* Knapik, Niwiński and Urzyczyn [28] have shown that as generators of ranked trees, higher-order *safe* recursion schemes are equi-expressive with *higher-order pushdown automata* [31]. Their result and an earlier result by Damm and Goerdt [16] may be viewed as attempts to answer the question; they both had to impose somewhat unnatural syntactic constraints (of safety and derived types respectively) on recursion schemes in order to establish their characterizations.

A partial answer was recently obtained by Knapik, Niwiński, Urzyczyn and Walukiewicz. In an ICALP'05 paper [29], they proved that order-2 homogeneously-typed (but not necessarily safe) recursion schemes are equi-expressive with a variant class of order-2 pushdown automata called *panic automata*. In a preprint [21], we give a complete answer to the question. We introduce a new kind of higher-order pushdown automata (which generalize *pushdown automata with links* [8], or equivalently panic automata, to all finite orders), called *collapsible pushdown automata* (CPDA), in which every symbol in the stack has a link to a (necessarily lower-ordered) stack situated somewhere below it. In addition to the higher-order stack operations $push_i$ and pop_i, CPDA have an important operation called *collapse*, whose effect is to "collapse" a stack s to the prefix as indicated by the link from the top_1-symbol of s. In [21] we prove the following result:

[2] *Property* APT because the APT corresponds to the property described by a given modal mu-calculus formula.

Theorem 2 (Equi-Expressivity). *CPDA are equi-expressive with* recursion schemes *as generators of (possibly infinite) ranked trees.*

In one direction, we give a simple algorithm that transforms an order-n CPDA to an order-n recursion scheme that generates the same tree, uniformly for all $n \geq 0$. In the other direction, using ideas from game semantics, we give an effective transformation of order-n recursion schemes (not assumed to be *homogeneously typed*, and hence not necessarily *safe*) to order-n CPDA that compute *traversals* over the computation tree of the scheme, and hence paths in the tree generated by the scheme. Our equi-expressivity result is the first automata-theoretic characterization of higher-order recursion schemes. Thus CPDA are also a characterization of the *simply-typed lambda calculus with recursion* (generated from uninterpreted 1st-order symbols) and of (pure) *innocent strategies*.

Verifying PCF programs: reachability checking

As a further direction (and a possible application of path-traversal correspondence), we consider the problem of reachability checking of higher-order computation. In the simplest form, reachability is the problem: Given a state of a transition system, is it reachable from the start state? Reachability is arguably the most important test in the computer-assisted verification of computing systems. Reachability (in its various forms) is expressible in standard temporal logics such as EF, LTL, CTL, etc., but it is typically computationally more tractable than the model checking of any of these logics (e.g. for pushdown systems, reachability is polytime [1], whereas EF-, LTL- and CTL-model checking are respectively PSPACE-complete, EXPTIME-complete and EXPTIME-complete [27]). In recent years, reachability checkers (such as SLAM [9], Blast [23], etc.) for first-order imperative programs have had a major impact in the verification community. Perhaps because of its simplicity and ease of use, reachability is now a standard approach to checking safety properties in the industry. It is therefore somewhat surprising that no reachability checker has been developed for higher-order programming languages such as Ocaml, Haskell and $F\#$. Indeed, to our knowledge, reachability of higher-order computation does not appear to have been studied in the literature.

The simplest (though already challenging) setting is PCF (generated from finite base types). We propose the following decision problem:

> PCF-Reachability: *Given a (possibly open) PCF term M and a subterm N of M, is there a program context $C[\,]$ such that the evaluation of $C[M]$ entails the evaluation of N? (Precisely, is there a program context $C[\,]$ such that $C[M] \longrightarrow^* E[N]$ for some evaluation context $E[\,]$?)*

For which fragment of PCF is the problem decidable? If there are positive answers, it would be interesting to consider the "global version" of the problem i.e. is it possible to compute a finite description of the set of contexts $C[\,]$ for a given pair of M and N?

An approach that seems promising is to appeal to the Path-Traversal Theorem for PCF [10], and consider traversals over the computation tree of M. The idea is to use appropriate alternating tree automata to "guess" a set of paths in the computation tree simulating traversals that witness yes-instances of the problem (see [37]). If this works out, it would be interesting to present the algorithm in terms that functional programmers can readily understand and appreciate.

Remark 1. (i) It is not clear if there is any connection between reachability (in our sense) and control flow analysis (e.g. [42]) of functional programs. In the past couple of years there have been several interesting developments in the verification and flow analysis of functional language. Xu and Peyton Jones have studied contract checking in Haskell (see Xu's forthcoming PhD thesis). A recent project of Shivers *et al.* [32] used abstract interpretation (specifically *abstract counting*) to build more precise flow analysers by garbage collecting "dead" environment structure in the abstract state space traversed by the functional programs.

(ii) When restricted to *finitary* (i.e. recursion-free) PCF, the problem is related to the atoms case of the Interpolation Problem, which is decidable [38]. (The Interpolation Problem is equivalent to the Higher-Order Matching Problem [41, 39].)

References

1. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *International Conference on Concurrency Theory*, pages 135–150, 1997.
2. S. Abramsky, K. Honda, and G. McCusker. Fully abstract game semantics for general reference. In *Proceedings of IEEE Symposium on Logic in Computer Science, 1998*. Computer Society Press, 1998.
3. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163, 2000.
4. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*. Birkhaüser, 1997.
5. S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL '97*. Springer-Verlag, 1998. Lecture Notes in Computer Science.
6. S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Logic and Computation: Proceedings of the 1997 Marktoberdorf Summer School*. Springer-Verlag, 1998.
7. 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 *Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications (TLCA'05)*, pages 39–54, 2005. LNCS Volume 3461.
8. K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*, pages 490–501, 2005. LNCS Volume 3411.

9. T. Ball and S. K. Rajamani. The SLAM Project: Debugging system software via static analysis. In *Proc. POPL*, pages 1–3. ACM Press, 2002.
10. W. Blum. *The Safe Lambda Calculus*. PhD thesis, University of Oxford, 2008. In preparation.
11. W. Blum and C.-H. L. Ong. Safe lambda calculus. In *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications (TLCA07)*, pages 39–53. Springer-Verlag, 2007. LNCS 4583.
12. W. Blum and C.-H. L. Ong. Path-correspondence theorems and their applications. Preprint, 2008.
13. D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. MFCS'02*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176, 2002.
14. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
15. W. Damm. The IO- and OI-hierarchy. *Theoretical Computer Science*, 20:95–207, 1982.
16. W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71:1–32, 1986.
17. W.-P. de Roever and J. W. de Bakker. A calculus for recursive program schemes. In M. Nivat, editor, *Proc. IRIA symposium on Automata, Languages and Programming*. North Holland, 1972.
18. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of FOCS'91*, pages 368–377, 1991.
19. D. R. Ghica and G. McCusker. Reasoning about idealized algol using regular languages. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming ICALP 2000*, pages 103–116. Springer-Verlag, 2000. LNCS Vol. 1853.
20. W. Greenland. *Game semantics for region analysis*. PhD thesis, Oxford University Computing Laboratory, 2005.
21. M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. Technical report, Oxford University Computing Laboratory, 2007. Preprint, 59 pages, downloadable from `users.comlab.ox.ac.uk/luke.ong/`.
22. C. Hankin and P. Malacaria. A new approach to control flow analysis. In *Proceedings of Compiler Construction*, pages 95–108. Springer-Verlag, 1998.
23. T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with BLAST. In *Proc. 10th SPIN Workshop*, 2003.
24. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
25. K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation (extended abstract). In *Proc. of ICALP'97, Borogna, Italy, July, 1997*. Springer-Verlag, 1997. LNCS.
26. J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163:285–408, 2000.
27. I. Walukiewicz. Model checking CTL properties of pushdown systems. *Lecture Notes in Computer Science*, 1974, 2000.
28. T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, pages 205–222. Springer, 2002. LNCS Vol. 2303.
29. T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP'05*, volume 3580 of *Lecture Notes in Computer Science*, pages 1450–1461. Springer-Verlag, 2005.

30. J. Laird. *A semantic analysis of control*. PhD thesis, University of Edinburgh, 1998.
31. A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.
32. M. Might, B. Chambers, and O. Shivers. Model checking via ΓCFA. In *Proceedings of the 8th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2007)*, pages 59–73, 2007.
33. A. Murawski and I. Walukiewicz. Third-order Idealized Algol with iteration is decidable. In *Proc. FOSSACS*, pages 202–218. Springer, 2005. LNCS Vol. 3441.
34. A. S. Murawski, C.-H. L. Ong, and I. Walukiewicz. Idealized Algol with ground recursion and DPDA equivalence. In *Proceedings 32nd International Colloquium Automata, Languages and Programming*, LNCS Volume 3580, pages 917–929. Springer-Verlag, 2005.
35. M. Nivat. On the interpretation of recursive polyadic program schemes. *Symp. Math.*, XV:255–281, 1975.
36. C.-H. L. Ong. Observational equivalence of third-order Idealized Algol is decidable. In *Proceedings of IEEE Symposium on Logic in Computer Science, 22-25 July 2002, Copenhagen Denmark*, pages 245–256. Computer Society Press, 2002.
37. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proceedings 21st Annual IEEE Symposium on Logic in Computer Science, Seattle*, pages 81–90. Computer Society Press, 2006. Long version (55 pp.) downloadable at `users.comlab.ox.ac.uk/luke.ong/`.
38. V. Padovani. Decidability of all minimal models. pages 201–215, 1996. LNCS 1158.
39. V. Padovani. Decidability of fourth-order matching. *Math. Struct. in Comp. Science*, 10:361–372, 2000.
40. M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Maths. Soc*, 141:1–35, 1969.
41. A. Schubert. A linear interpolation for the higher-order matching problem. pages 441–452, 1997. LNCS 1214.
42. O. Shivers. *Control-flow analysis of higher-order languages*. PhD thesis, Carnegie-Mellon University, 1991.
43. M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc.* IEEE *Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1986.